Capturing Event metadata in the sky:
A Java-based application for receiving astronomical Internet Feeds

A thesis presented in partial fulfilment of the requirements for the degree

of

Master of Computer Science

in

Computer Science

at Massey University, Auckland,

New Zealand.

Feng Jiang

2008

Acknowledgements

I wish to express my sincere thanks to my research supervisor, Dr. Ian Bond, for providing me this opportunity and many thanks for your kind helps.

Thanks for your invaluable support and patient guidance throughout this journey.

Abstract

When an astronomical observer discovers a transient event in the sky, how can the information be immediately shared and delivered to others? Not too long time ago, people shared the information about what they discovered in the sky by books, telegraphs, and telephones. The new generation of transferring the event data is the way by the Internet. The information of astronomical events is able to be packed and put online as an Internet feed. For receiving these packed data, an Internet feed listener software would be required in a terminal computer. In other applications, the listener would connect to an intelligent robotic telescope network and automatically drive a telescope to capture the instant Astrophysical phenomena. However, because the technologies of transferring the astronomical event data are in the initial steps, the only resource available is the Perl-based Internet feed listener developed by the team of eSTAR. In this research, a Java-based Internet feed listener was developed. The application supports more features than the Perl-based application. After applying the rich Java benefits, the application is able to receive, parse and manage the Internet feed data in an efficient way with the friendly user interface.

Keywords: Java, socket programming, VOEvent, real-time astronomy

Table of Contents

ACKNOWLEDGEMENTS 1

ABSTRACT 2

1. INTRODUCTION 5

1.1 A brief Introduction to the astronomical phenomena 5

1.2 Introduction the way to deliver instant astronomical event information 8

1.3 Introduction to current observation works 11

2. BACKGROUND 13

2.1 A closer look at VOEvent 13

2.1.1 History of VOEvent 13

2.1.2 Structure of a typical VOEvent message 14

2.1.3 Schema of the VOEvent 18

2.14 Current Uses of VOEvent 19

2.2 Review existing applications 20

2.2.1 Event broker application 20

2.2.2 Existing RSS news aggregators 21

2.3 Java application on the Internet 22

3. PROJECT DEVELOPMENT 24

3.1 Requirements collections and analysis 24

3.2 Application Architecture Design 25

3.2.1 Architecture with UML 27

3.2.2 The real-time listener architecture 29

3.2.3 The RSS/XML reader architecture 32

3.2.4 The comparision between the RSS/XML reader model and the real-time listener model. 34

3.3 Application Functional Design 35

3.3.1 Main form designs 35

3.3.2 Server setting form designs 38

3.3.3 Real-time Model Designs 39

3.3.3.1 Connecting 41

3.3.3.2 Listening 41

3.3.3.3 Responding.....44
3.3.3.4 Parsing.....45
3.3.3.5 Displaying.....52
3.3.3.6 Recording.....53
3.3.3.7 Thread technique for the time critical programming.....54
3.3.4 RSS Model Designs.....55

4. PROJECT DEPLOYMENT.....**60**

4.1 Enterprise application deployment.....**60**

4.2 Java Web Start Deployment.....**62**

5.PROJECT PERFORMANCE.....**64**

5.1 Multiple resource listening.....**64**

5.2 Real-time parsing time cost.....**65**

5.3 RSS reader parsing time cost.....**66**

5.4 Validation check time cost.....**67**

5.5 Parsing method designs.....**68**

6. CONCLUSION.....**70**

7. FUTURE WORK.....**71**

REFERENCES.....**72**

1. Introduction

In this chapter, a brief introduction to the astronomical phenomena, the ways to deliver the astronomical event information and the current work for astronomical observations are involved.

1.1 A brief Introduction to the astronomical phenomena

Astronomical events, such as solar eclipses and meteor showers, are most visible and discovered by amateur astronomy. In the modern astronomy, astronomers are doing the researches on Gravitational Microlensing, Supernova, Gamma-ray bursts.

Supernova:

One of the most energetic explosive events known is a supernova. These occur at the end of a star's lifetime, when its nuclear fuel is exhausted and it is no longer supported by the release of nuclear energy (NASA's HEASARC, 2003). A supernova is a very powerful event: a star which suddenly brightens to about $10^{9}-10^{10}$ L (Monique Signore, Denis Puy, 2001).

The following image (as shown in Fig 1.1) has been constructed from NASA's Spitzer space telescope, Hubble Space telescope and Chandra X-ray Observatory. (NASA's Image Galley, 2004).

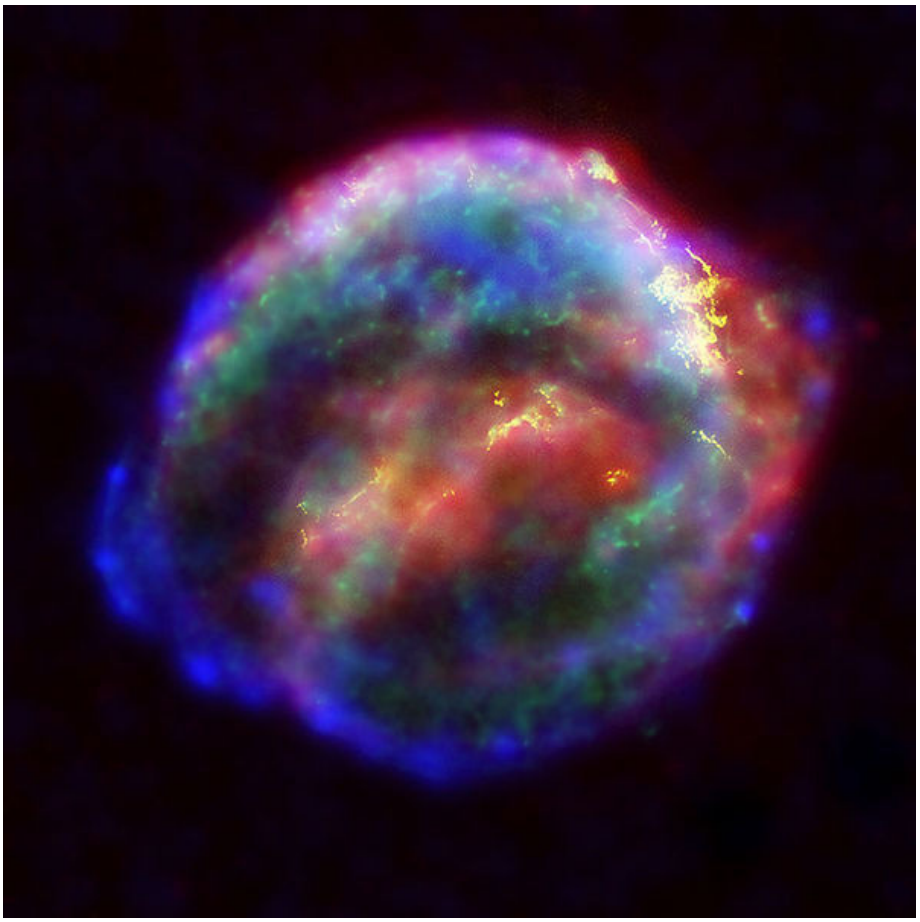

Fig 1.1 the Remnants of Kepler's Supernova.

Gamma-ray bursts:

Gamma-ray bursts is another example of time critical astrophysics is in the study of Gamma-ray bursts. These intense, unpredictable flashes of gamma radiation are first detected by orbiting satellites. Ground based telescopes then respond in real-time to follow-up these detections.

Gamma-ray bursts are short-lived bursts of gamma-ray photons, the most energetic form of light. At least some of them are associated with a special type of supernovae, the explosions marking the deaths of especially massive stars (NASA's Imagine the Universe, 2007).

Gamma-ray bursts may last anywhere from a few milliseconds to several minutes. Gamma-ray bursts are detected at the rate of about once a day, and while they are on, they outshine every other gamma-ray source in the sky, including the sun. (Nikos Drakos, 1993)

The following image (as shown in Fig 1.2) of gamma-ray bursts was detected by Swift spacecraft. The gamma-ray burst may have been caused by a collapsing star. This illustration shows the center of a dying star collapsing just minute before the star implodes and emits a gamma-ray burst (NASA Swift catching Gamma-Ray Bursts, 2006).

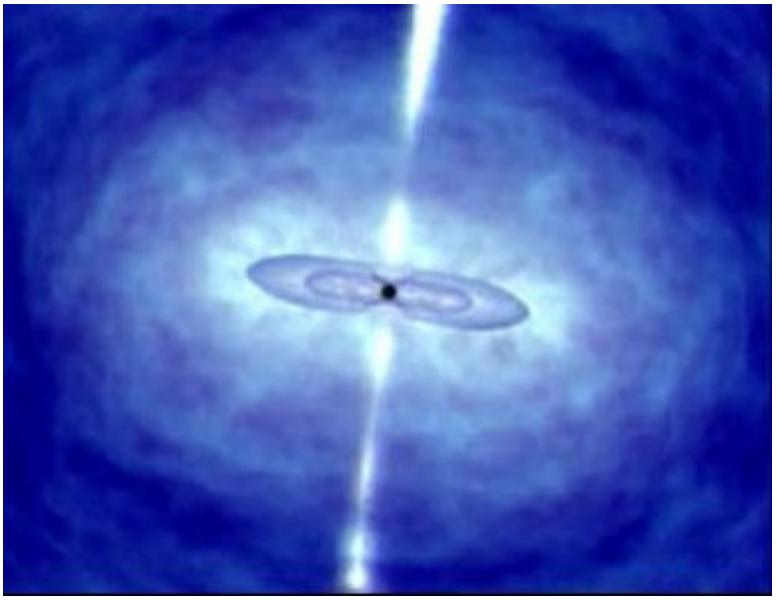

Fig 1.2, the gamma-ray burst detected by the Swift satellite

Gravitational microlensing:

One example relevant to New Zealand is the search for extra solar planets using gravitational microlensing. This technique is used to detect extra solar planets by

studying the time signatures caused when a foreground star together with any orbiting planets amplifies the light of a background star. These time signatures are unpredictable in nature and can occur at any time.

The MOA (Microlensing Observations in Astrophysics) project in New Zealand uses a 1.8 m telescope to detect these lensing events in real time.

MOA is a NZ/Japan collaboration that includes Canterbury and Victoria Universities as well as Auckland and Massey. Using a telescope at Mt John Observatory in Tekapo they image selected regions of the sky towards the galactic centre and measure the brightness of millions of stars every clear night - each year they detect dozens of these rare chance stellar alignments, MOA then co-ordinates it observations with many telescopes throughout the globe in order to maximize the chance of catching the planetary microlensing signatures (MOA, 2006). Also in Massey University, a MOA transient alert webpage has been setup to alert microlensing events, since 2000. (MOA transient alert page, 2000)

1.2 Delivering instant astronomical event information

Astronomy is the oldest science, dating back thousands of years to when primitive people noticed objects in the sky overhead and watched the way the objects moved (Microsoft Encarta, 2007).

In ancient time, delivering the instant astronomical event information seems to be impossible, because the observers only discovered the obvious phenomena by human eyes and informed others by letters, dictation.

In the past few hundred years, astronomical research has been greatly enhanced by the industrial age and the information age. Delivering the transit astronomical event data becomes faster and faster. Technologies, computers, the Internet, robotic telescopes etc. have been widely applied. Modern astronomical telescopes and robotic telescopes, and space telescopes are able to capture the transit phenomenon. Modern physics and modern astronomy support the theoretical principle. Computers, workstation and super computers are able to be used to analyze the captured images and data. The Internet connects most of computers in every part of the world. Information has been delivered faster than ever before.

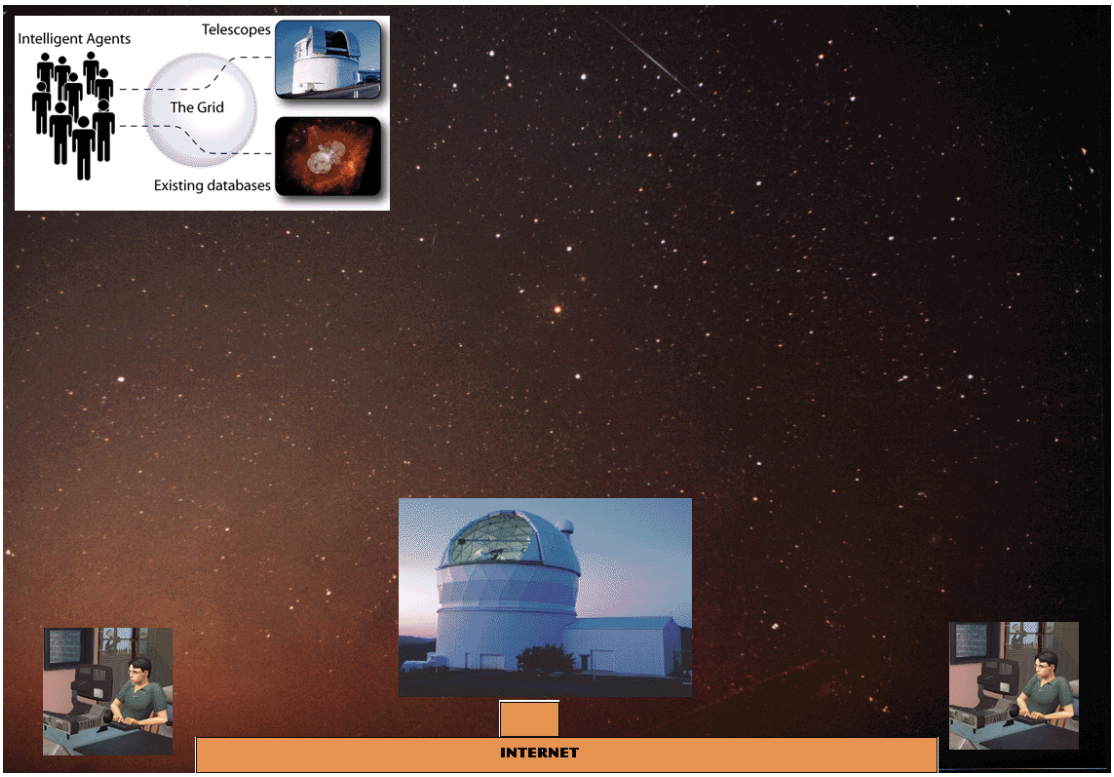

Fig 1.3 the modern way to deliver astronomical event information

When a sky event is captured by an astronomical telescope, the event data would be uploaded the specified server. Other users are able to have the information via the Internet almost at the same time, and then users may decide how to follow it up. For example, users may achieve the information or alert the communication. Astronomers may guild their own telescopes to continue observing and do the follow-up works.

Since the robotic astronomical telescopes were invented, astronomical event information is able to be packed as a data packet with a command. The command is able to automatically guild the robotic telescopes to locate the event direction for further observations. In this way, a transit astronomical event can be imaged and recorded by different angles at almost same time.

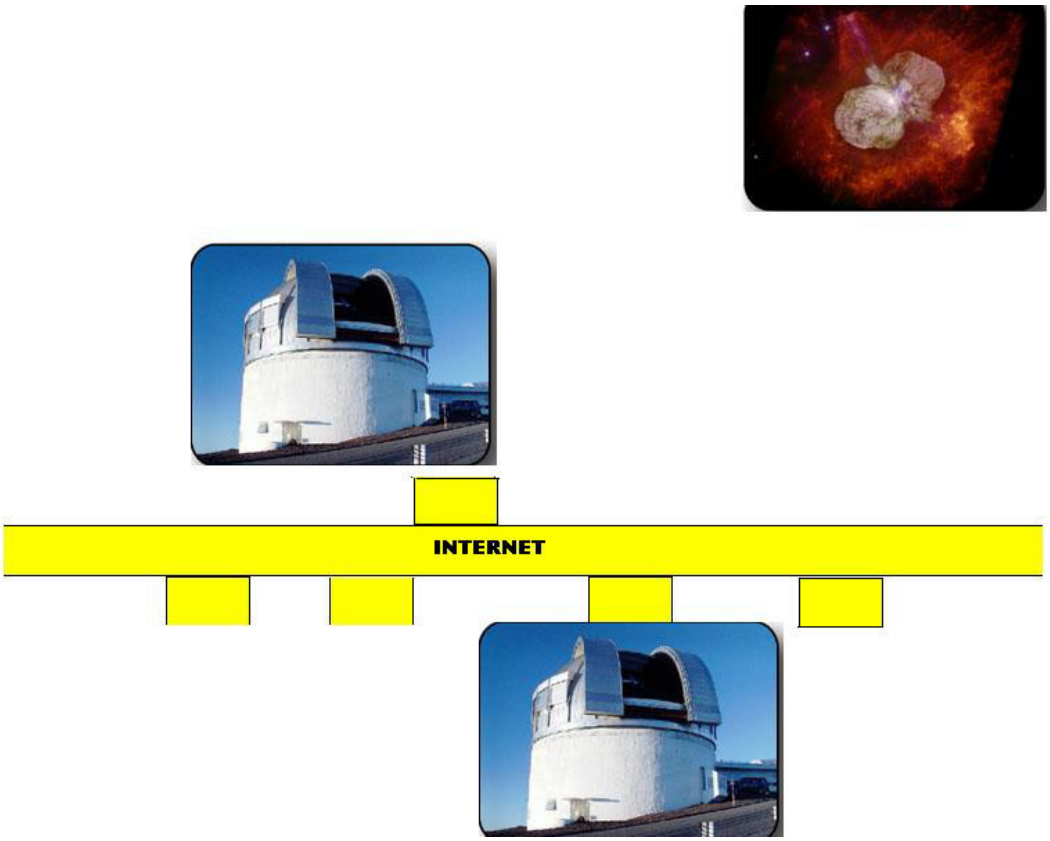

Fig 1.4 Delivering astronomical event information between telescopes

As shown in Fig 1.4, the prompt communication between robotic astronomical telescopes comes true, due to the swift information transmission.

There is now considerable international activity in investigating the best use of current technologies for real-time sharing of astronomical data across the Internet.

1.3 Introduction to current observation works

For the hardware of the current observation, more and more astronomical observing satellites have been launched and ground-based observing points have been setup as well. For example, the INTEGRAL satellite has been successfully launched in October 2002, which is designed by the European Space Agency for capturing high-resolution imaging of the gamma-ray bursts (Winkler, 1999). It was therefore proposed to implement a ‘‘burst alert system’’ in order to allow rapid multi-wavelength follow-ups (Mereghetti et al., 1999).

For the software part of the astronomical observation and delivering event information purposes, in 2002, the International Virtual Observatory Alliance (IVOA) was formed with a mission to "facilitate the international coordination and collaboration necessary for the development and deployment of the tools, systems and organizational structures necessary to enable the international utilization of astronomical archives as an integrated and interoperating virtual observatory" (IVOA, 2002).

The IVOA now comprises 16 VO projects from Armenia, Australia, Canada, China, Europe, France, Germany, Hungary, India, Italy, Japan, Korea, Russia, Spain, the United Kingdom, and the United States. The work of the IVOA focuses on the development of standards.

In 2006, the standard of Virtual Observatory Event (VOEvent) was officially published by IVOA. The VOEvent is a standardized language used to report observations of astronomical events. The VOEvent messages are intended to be general enough to describe all types of observations of astronomical events.

Recently, many projects have built a number of observation networks. For instance,

the RoboNet-1.0 project and the eSTAR project are from United Kingdom. The eSTAR project is a programme to build an intelligent robotic telescope network. It is a joint project between the Astrophysics Research Institute at Liverpool John Moores University and the Astrophysics Research Group at University of Exeter. The VOEventNet project is from USA. Also in New Zealand, a Massey university team is doing the transient alert project called “MOA”. Each of them has their own schemas to encode the event information. A standard information packet would allow event senders and receivers to easily represent, transmit, archive, and publish the discovery of the immediate events in the sky. Most of these observation networks are based on VOEvent standard.

2. BACKGROUND

This research is a Java-based application for receiving astronomical event information via the Internet. All the event information are packed in VOEvent standard and transferred as the Internet feed. In this chapter, the details of VOEvent, the reviews for existing applications, and related Java knowledge are included.

2.1 A closer look at VOEvent

The objective of the VOEvent effort is to define the content and meaning of a standard information packet for representing, transmitting, archiving, and publishing a discovery of an immediate event in the sky (IVOA 2006).

The VOEvent messages are written in XML language. In order to rapidly disseminate the event data, the VOEvent messages are designed to be compact and quickly transmittable over the Internet, rather than natural languages, so that automated systems can be most effective in interpreting VOEvent packets.

2.1.1 History of VOEvent

VOEvent builds upon previous generic publishing schemes such as International Astronomical Union Central Bureau for Astronomical Telegrams (IAUCs and CBETs) and the Astronomer's Telegrams. The main difference is that VOEvent messages are designed to be automatically parsed and filtered whereas ATELs and IAUCs are intended to be read by humans.

The scope of VOEvent is informed by several other messaging schemes used to

facilitate rapid discovery announcements for specialized sub-fields of astronomy (such as OGLE microlensing alerts and Supernova Early Warning System). The closest ancestors of VOEvent are The Telescope Alert Operation Network System, and the Gamma Ray Burst Coordinates Network messages, used extensively by the gamma-ray burst community (Wikipedia VOEvent, 2006).

Though most of VOEvent messages currently issued are related to supernovae, gravitational microlensing, and gamma-ray bursts,

2.1.2 Structure of a typical VOEvent message

By definition, a VOEvent packet contains a single XML VOEvent element. A VOEvent element may contain at most one of each of the following optional tags (VOEvent packet, 2006):

Tags	Description
<Who>	Author or publisher Identification
<What>	Event Characterization
< WhereWhen>	Space-Time Coordinates
< How>	Instrument Configuration
< Why >	Initial Scientific Assessment
< Citations >	Follow-up Observations
< Description >	Human Oriented Content
< Reference >	External Content

Table 2.1 VOEvent Semantics

• <VOEvent>

Every VOEvent packet must begin with this element. It contains the identifiers, roles and version of the VOEvent. Typically a <VOEvent> element has four attributes.

1. Id: Each VOEvent packet has its unique identifier.
2. Role: The optional role attribute accepts three possible enumerated values: “observation” for actual astronomical events observed, “prediction” for predicting, but not yet happened, or “test” for nothing observed, test use only.
3. Version: “1.0” for all VOEvent packets has been required.
4. Schema: A xmlns attribute refers to the xml schema to define the contents of the VOEvent packet

For example, a VOEvent packet from IVOA example2-v1.0.XML starts:

<VOEvent id="ivo://raptor.lanl/VOEvent#23565" role="observation" version="1.0"
xmlns="http://www.ivoa.net/xml/VOEvent/v1.0"
xmlns:stc="http://www.ivoa.net/xml/STC/stc-v1.20.xsd"
http://www.ivoa.net/internal/IVOA/IvoaVOEvent/VOEvent-v1.0.xsd">

The attributes for this VOEvent packet are as following.

Id: ivo://raptor.lanl/VOEvent#23565”

Role: observation,

Version: 1.0

Schema: xmlns:stc=http://www.ivoa.net/xml/STC/stc-v1.20.xsd
http://www.ivoa.net/internal/IVOA/IvoaVOEvent/VOEvent-v1.0.xsd

• <Who>

Describing who is responsible for the information. In this sub element, the details of publishers or groups would be included. Typical <Who> content would include <PublisherID>, <Contact>, and <Date>. The <Contact> element contains the information of the author's phone number, email address, physical address, etc…Also the <Date> element contains the date and time of the creation of the VOEvent packet.

• <What>

The **<What>** and **<Why>** elements work together to characterize the nature of a VOEvent. That is: **<What>** was factually measured or observed to occur, versus **<Why>** in terms of its hypothesized underlying cause or causes.
A **<What>** element contains a list of **<Param>** elements which may be associated and labeled using **<Group>** elements

For example, here are three values:

TRIGGER_NUM = 114299
RATE_SIGNIF = 20.49
GRB_INTEN = 73288

In VOEvent, these can be represented as:

<Param name="TRIGGER_NUM" value="114299" />
<Param name="RATE_SIGNIF" value="20.49" ucd="stat.snr" />
<Param name="GRB_INTEN" value="73288" unit="ct" ucd="phot.count" />

• <WhereWhen>

Location and time are the important points for the VOEvent packet. <WhereWhen> shows where the location for the observation was and when it happened. <WhereWhen> element contains <ObservationLocation>, <TimeFrame>, etc… This draws from the Space-time coordinate (STC) recommendation to the IVOA.

• <How>

It is a description of the instrumental setup on where the data were obtained. It may contain <Reference> and <Descripition>

• <Why>

It explains the event by the view of the author. It may contain <Concept> and <Inference> tags.

• <Citations>

A VOEvent packet without a **<Citations>** element can be assumed to be asserting information about a new celestial discovery. Citations reference previous events to do one of three things:

- Follow-up an event alert with more observations, or
- Supersede a prior event with better information, or
- Issue a complete retraction of a previous event.

In addition, citations allow merging multiple events into a single related thread, or contrarily, allow separating a single event into multiple threads (VOEvent Citations, 2006).

An example of a standard VOEvent message (Fig 2.1),

```
<?xml version="1.0" encoding="UTF-8"?>
<VOEvent id="ivo://raptor.lanl/VOEvent#23564" role="observation" version="1.0"
 xmlns="http://www.ivoa.net/xml/VOEvent/v1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
  "http://www.ivoa.net/xml/VOEvent/v1.0 http://www.ivoa.net/internal/IVOA/IvoaVOEvent/VOEvent-v1.0.xsd">

 <Who>
  <PublisherID>ivo://raptor.lanl/organization</PublisherID>
  <Date>2005-04-15T14:34:16</Date>
 </Who>

 <What>
  <Param name="RA" ucd="pos.eq.ra" unit="deg" value="185.0" />
  <Param name="Dec" ucd="pos.eq.dec" unit="deg" value="13.2" />
  <Param name="magnitude" ucd="phot.mag;em.opt.R" unit="mag" value="18.2" />
 </What>

 <Why>
  <Concept>Fast Orphan Optical Transient</Concept>
 </Why>
</VOEvent>
```

Fig 2.1 an example of the VOEvent packet

In this example, it contains only three sub elements, which are sub-element “Who”, “What”, “Why”.

2.1.3 Schema of the VOEvent

A well-formed VOEvent message must validate against the VOEvent schema (.xsd). A valid message may omit most of the informational tags listed above, but since the creation of VOEvent messages is done automatically, most opt to transmit the fullest content available.

In Fig 2.2, it is a part of the VOEvent schema

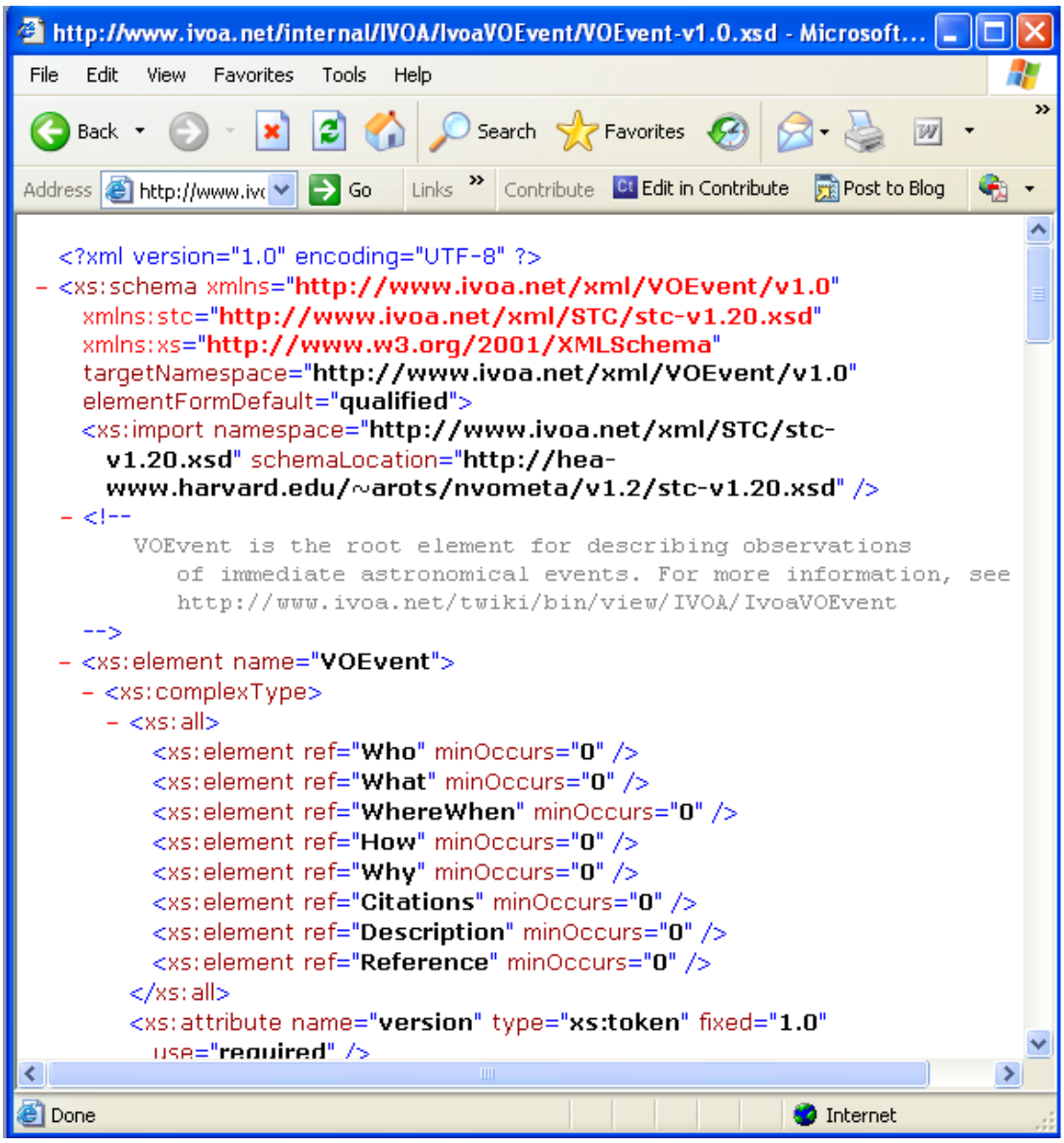

Fig 2.2 a part of the VOEvent schema "VOEvent-v1.0.xsd"

2.14 Current Uses of VOEvent

There are live VOEvent feeds (through a variety of protocols, such as RSS and Jabber) currently available from (Wiki VOEvent 2006): The Caltech VOEvent group and the eSTAR project (Exeter, England).

The goal of this application is to receive the VOEvent feeds based on eStar from England and MOA from New Zealand.

2.2 Review existing applications

According to capturing Internet feeds, technically, they can be identified as a real-time listener or a reader activated by users, like RSS news feed from the Internet. The existing applications for the Internet feed reader include the event broker applications for the real-time listener, and the RSS news aggregators for the static RSS feed updates.

2.2.1 Event broker application

The use of real-time Internet feeds for transmitting astronomical data is a relatively new initiative. As a result, most of software implementations are still under development. For example, only a simple Perl-based VOEvent listener application is available on the eSTAR website for interested users (eSTAR Event Broker, 2006), as shown in Fig 2.3.

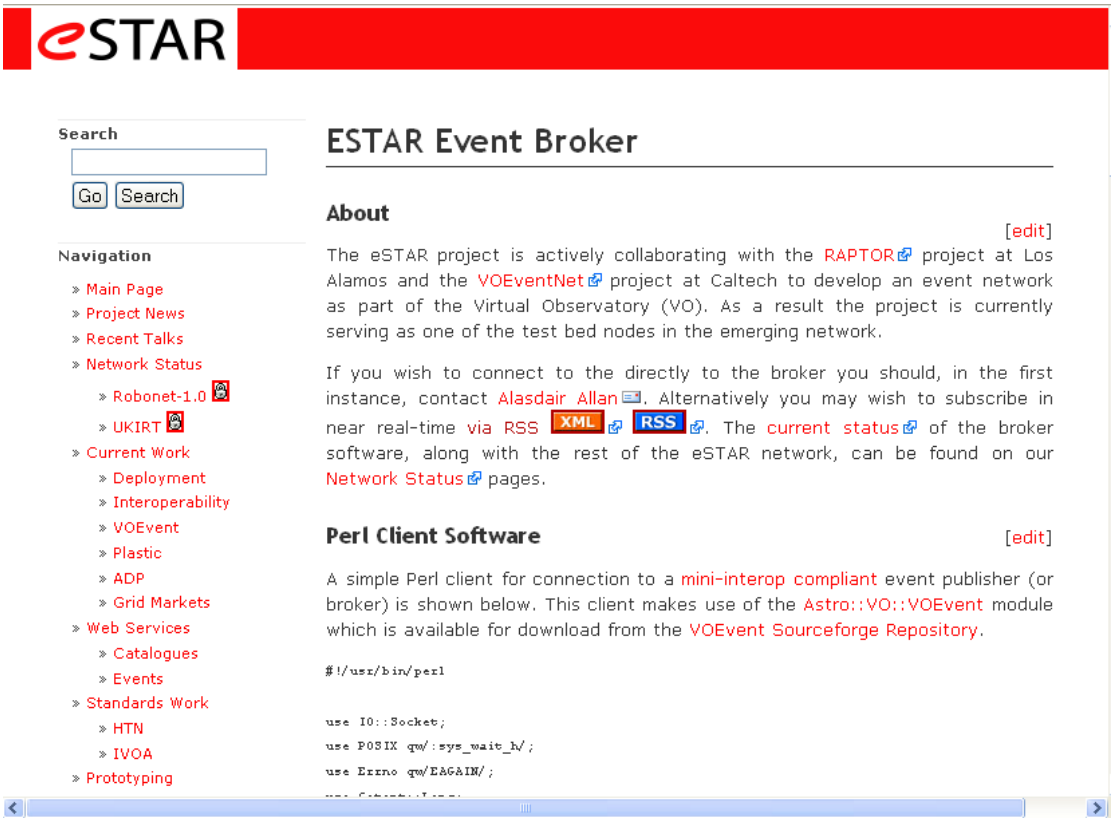

Fig 2.3 the only available Perl-based eSTAR client software

The Perl-based eSTAR application is too simple, which only involves the reading socket programming and it does not include any parsing method designs, user interface design, and etc…So it is not suitable for most of users without professional programming experience.

2.2.2 Existing RSS news aggregators

Web-based aggregators are applications that reside on remote servers and are typically available as Web applications such as Google Reader (as shown in Fig 2.4).

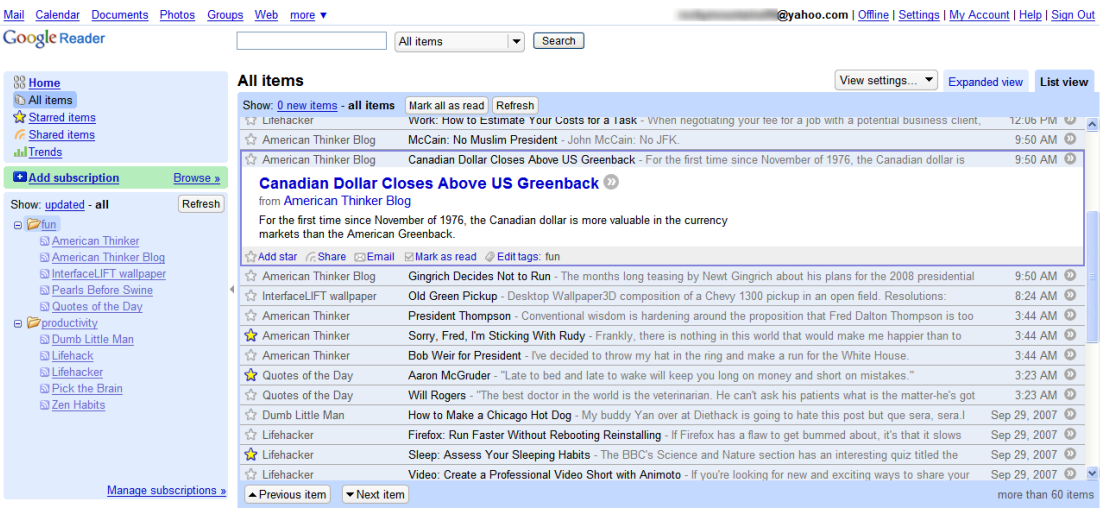

Fig 2.4 a screenshot of Google Reader for RSS news reader

Client software aggregators are installed applications designed to collect Web feed subscriptions and group them together using a user-friendly interface, such as NewzCrawler (as shown in Fig 2.5). Basically, they are developed for the particular purposes, for example, most of the Internet feed readers are doing the reading jobs for the news and weblogs, some of them are collecting emails, like Microsoft Outlook.

For reading and parsing the particular information, like the astronomical event feeds, the VOEvent Internet feed reader software has to be developed independently.

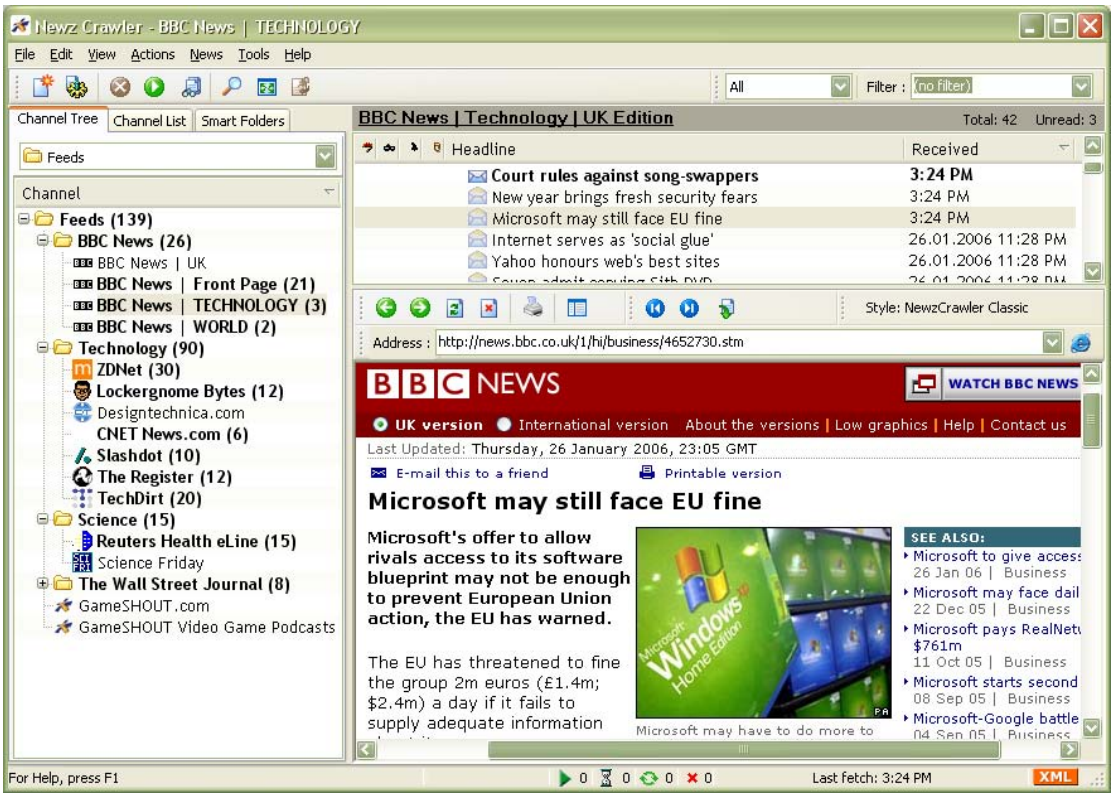

Fig 2.5 a screen shot of the Newz Crawler's application-based RSS feed reader

2.3 Java application on the Internet

The Java-based Internet feed listener application is developed by Java. In this section, the related Java knowledge for the project is presented.

Java was the first object-oriented programming language created for the Internet. Since its arrival on the Internet scene in 1996, it has been a powerful programming tool for web programmers and developers (P.K. Yuen &V.Lau, 2003).

- **Java Socket Basics**

A socket is a connection between two hosts. Sockets are an innovation of Berkeley Unix that allow the programmer to treat a network connection as just another stream onto which bytes can be written and from which bytes can be read (O'Reilly 1996)

In the Java-based Internet listener application, connecting to the remove hosts from the eSTAR and MOA server is the first job. The “java.net.Socket” class allows the application to perform all fundamental socket operations.

- **Java Web Start**

In the deployment of this application, a Java Web Start technology would be used. Standalone Java software applications can be deployed with a single click over the network. Java Web Start ensures the most current version of the application will be deployed, as well as the correct version of the Java Runtime Environment (Java Web Start, 2006). It is a Java-based application that allows full-featured Java 2 client applications to be launched, deployed, and updated from a standard Web server (Steven Kim, 2001).

It is a very convenient way to publish the Java-based Internet feed listener application over the Internet. Users are able to have the entire application with all libraries and classes inside downloaded from the web server. Users only need a web browser and the Internet connected.

- **XML Validation**

In this application, the XML validation check has been applied, because the VOEvent packet is in the XML format and checking the validation of the VOEvent can avoid the unnecessary data lost. The XML validation includes “Well Formed” check and “Valid” check.

XML with correct syntax is “Well Formed” XML, and XML validated against an XML based defined structure (XML Schema) is “Valid” XML (W3C School, 2008).

3. Project Development

In this chapter, it includes the motivation for making this Java-based Internet feed listener application, the architecture designs and the details of the functional designs.

3.1 Requirements collections and analysis

In this research, the motivations for developing a Java-based Internet feed listener are due to the strengths of Java language on the rich language supports for Internet programming, graphical design, and database integration etc... Java now has his over 2.5 million programmers. More and more users are enthusing over writing libraries for it, which means Java also has rich and large numbers of libraries. Opening sources, cross-platform designed and more secure functions make Java become a very popular programming tool (Rogers Cadenhead, Laura Lemay, 2004).

Perl is a stable, cross platform programming language. It is used for mission critical projects in the public and private sectors and is widely used to program web applications of all needs (eSTAR broker, 2006).

Comparing to Java and Perl, we can see the differences from Table 3.1. Java does have some main benefits for doing the application, which may give the better looking user interface, richer classes supported, more secure than Perl. Also Java makes a great effort on net and socket programming as well.

Features	JAVA	PERL
Rich Text Processing	Yes	Yes
Rich class supports	Yes	No
Powerful Graphical API	Yes	No
Excellent Database Integration	Yes	Yes
Centralized Repository of Reusable Code	No	Yes
Overloading Method Support	Yes	No
Numerous Embedded Implementations	Yes	No
Open source, Cross platform	Yes	Yes
More Secure	Yes	No
Widely used	Yes	No

Table 3.1: Comparing Java and Perl

On the other hand, in the field of VOEvent feed listeners so far, there are no Java tools available for receiving and analyzing VOEvent data. It seems that the only resource available is the Perl-based VOEvent listener developed by the eSTAR team. In this research, we also made a few tests on the performance issues with the Java-based Internet feed listener application.

To sum up, we believe that making the Java-based Internet feed listener application for the astronomical event information should be practical and helpful.

3.2 Application Architecture Design

There are two methods for all programs to read Internet feeds: one is real-time detection and listening, which works all times to listen to any event packets without any delay. The other method is to read the RSS feed from official websites of relative

organizations. When the official server is on, they would always update the latest events online as RSS feeds. Users may access this data via a web browser or by a RSS reader application.

Also in order to give a flexible working environment of the application, this Java-based Internet listener application should have a setting interface for updating the information of servers in demand.

Moreover, for giving a user friendly interface, a window format with a multiple document interface for this application is required. In this section, we are going to discuss the Java-based application architecture, the real-time listener model, and the RSS/XML reader model.

In the fig. 3.1, it shows a screenshot of the Java-based Internet Feed reader.

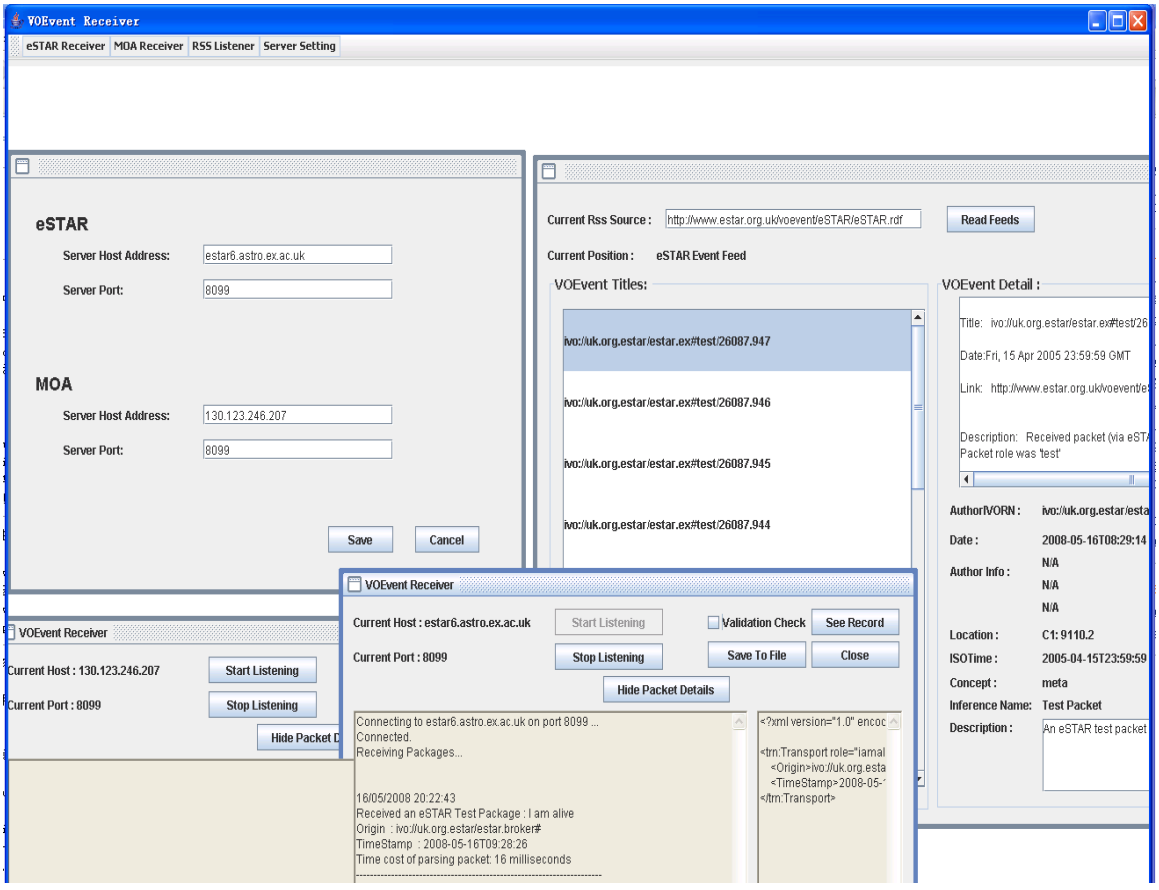

Fig 3.1 a screen shot for the Java-based Internet listener application

3.2.1 Architecture with UML

In the last few decades, Java developers have a new way to visualize the system, modeling software before they build it. The Unified Modeling Language (UML) is OMG's most-used specification, and the way the world models not only application structure, behavior, and architecture, but also business process and data structure (UML 2005). UML is an open standard notation that allows developers to build visual representations of software systems (C.T. Arrington, Syed H. Rayhan, 2003).

In the table 3.2, it gives all the main class names of the application and brief description for each of them. In the figure 3.2, it shows the UML class diagram of the Java-based Internet Feed listener application.

Class Name	Description
MainForm	It is the parent form of the whole application. With the JAVA "swing" class, it gives the graphical user interfaces (GUIs) and it is ready to active the child forms of the application.
RssReader	It is designed for the RSS reader model. It is able to edit the resource of feeds, and show all the latest event information line by line and parsed information show at the right side.
VOEvent	It is designed for the real-time listener model. It is able to accept the real-time event information from both eSTAR and MOA server and able to run parallel
ReadTextFile	System backup and record class.
Setting	Set up the details of servers
ParsePackage	It is a main class to parse each VOEvent packet.

Table 3.2 the classes' details for the application

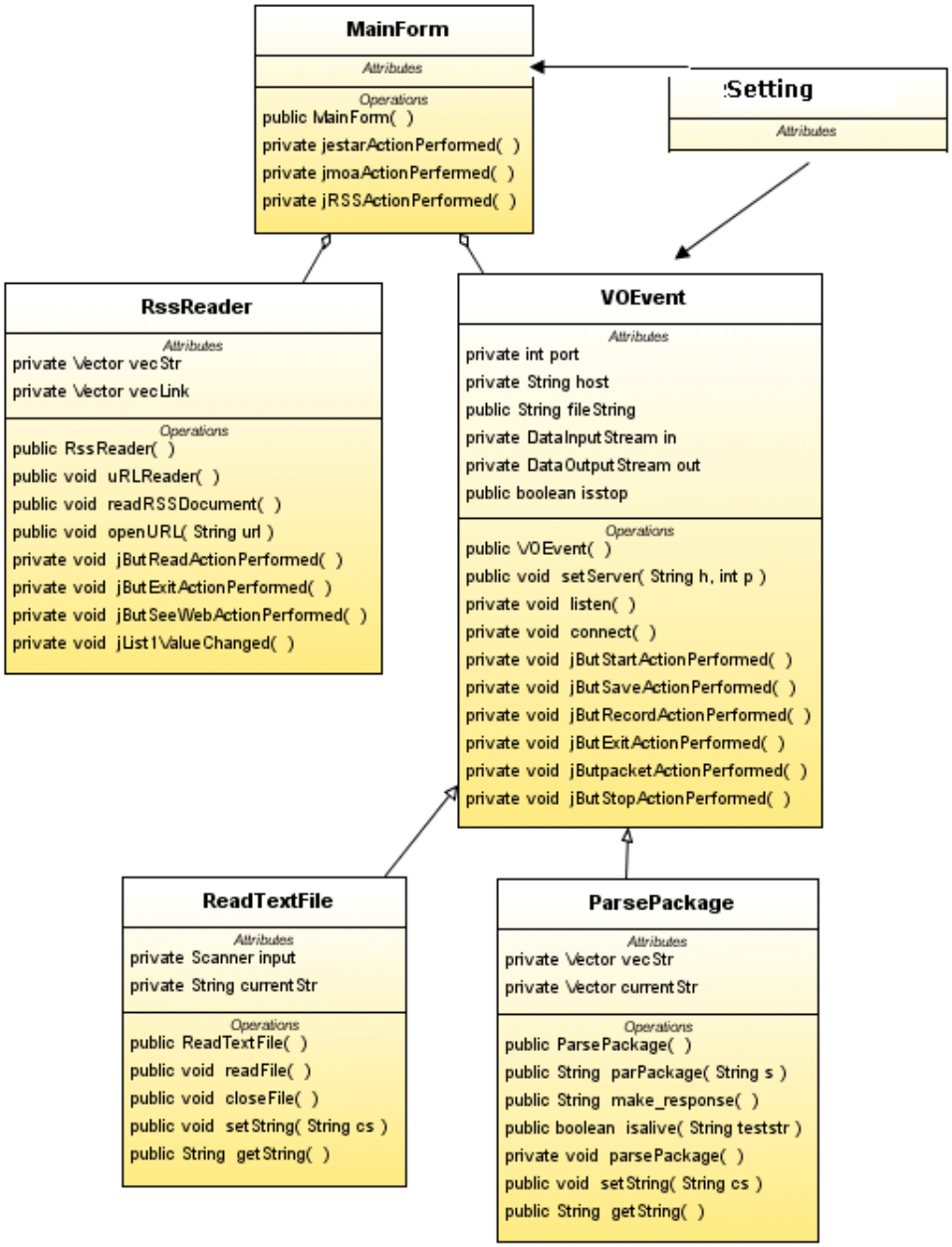

Fig 3.2 the UML class diagram for the application

3.2.2 The real-time listener architecture

In the real-time listener model, the application is setup to connect with the server at all the time for achieving all signals. This is the main part of the application, because the real-time receiving astronomical data is very helpful for better observation and reaction than the static model.

The working principles for the real-time listener model are as following.

1. Connect to the server with the corresponding socket from the server
2. Receive the raw data from the server
3. Send the response data to the server
4. Parse the raw data into user readable information
5. Make reaction to the information. (Display, record, alarm, etc…)

Fig 3.3 shows the working procedure between the Java-based Internet feed listener client and the remote eSTAR Server

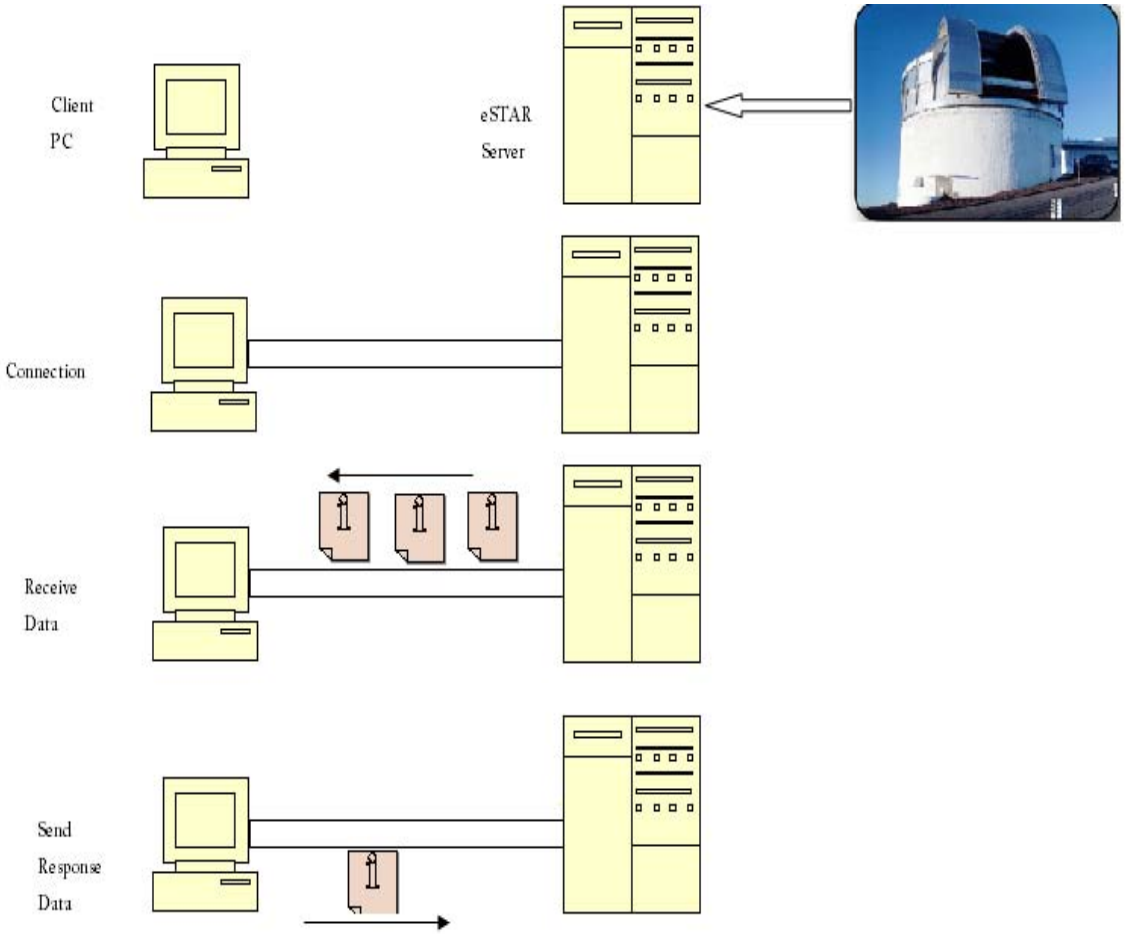

Fig 3.3 the working procedure between the application and the eSTAR Server

All data here are packed with VOEvent standard.

When the application is connecting with the server, there are two working statuses existing. One is the system idle status, and the other is the system busy status. (As shown in Fig 3.4)

When the real-time listener is in the idle status, no astronomical phenomena are detected and no valuable event data can be sent by the server, but the connection between the listener and the server is still on. The server dispatches a small testing VOEvent packet about every 60 seconds. The packet names as “I am alive”. When the application receives this packet, a responding data would be sent to the server, which

informs the server that the routine of the connection is clear and available. A simple parsing procedure is applied to check the “I am alive” packet and display the brief information, like time and titles. Since the astronomical phenomena do not always happen, the application would be in the idle status at most of time.

Once an astronomical phenomenon appears and is detected, the server will send a number of VOEvent packets to the client to describe the situation. Both the server and the real-time listener are in the busy status. Normally the eSTAR sever would send out two or three VOEvent packets, which contains about 1200 Byte data per packet. The listener runs the completed parsing procedure to analyze the whole raw data, and parses out the user readable information. For further operation, the listener would display the essential information on the console, like the time, the location, the details of observers, and the details of astronomical phenomenon. At the same time, the recording function and alerting function also can be triggered.

Fig 3.4 shows the example about the working procedure between the application and eSTAR Server in two statues.

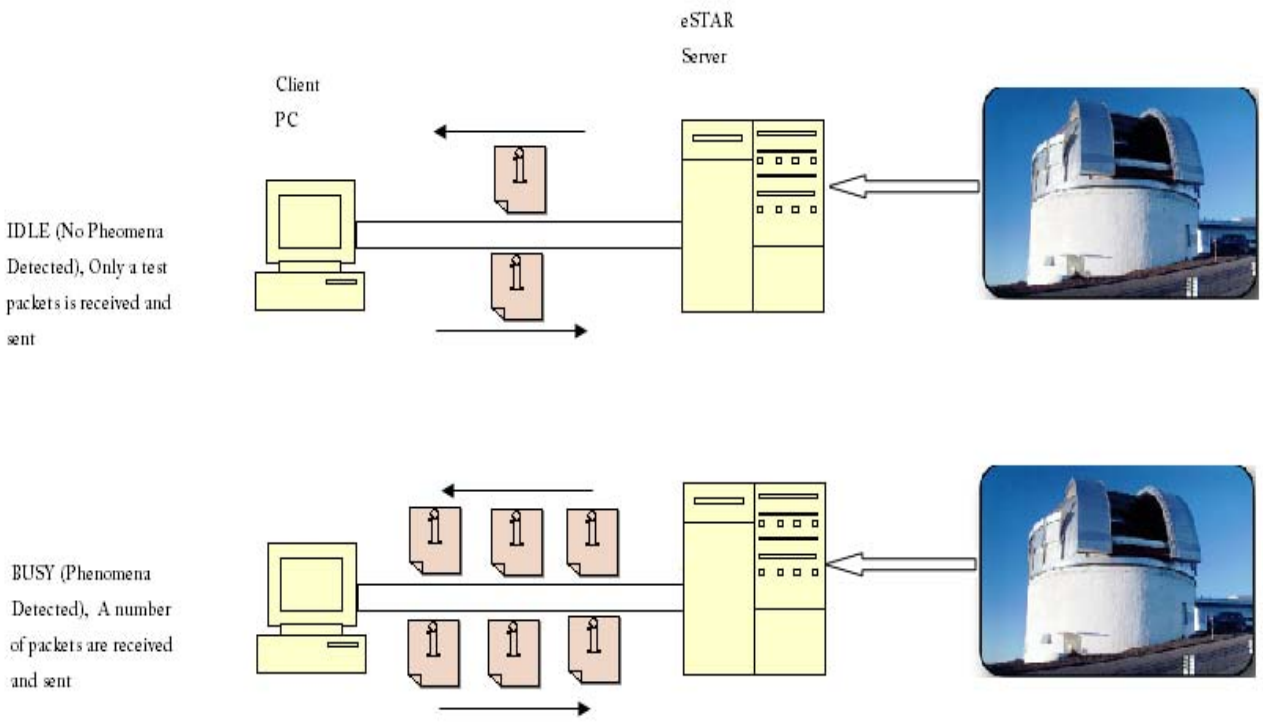

Fig 3.4 the system in idle and the system in busy

3.2.3 The RSS/XML reader architecture

Normally, each feed supplier has its own website, which also supports current event information with the RSS/XML format. For example, in the eSTAR project, the "eSTAR.rdf" file in the eSTAR website contains all the latest astronomical information from observers. This gives us a second idea to have the data through the Internet.

In this RSS reader model, the application connects to the web server with the TCP/IP protocols. The web server is not the main server, which gathers all the event

information at the first time. The web server has to have the data uploaded from the main eSTAR observing servers for the updates.

The working principles for the RSS reader model are as following.

1. Connect to the web server with the TCP/IP protocols
2. Read the source file from the web server
3. Parse the data onto the display panel
4. Make the further actions

Fig 3.5 shows the example about the working procedure between the application and eSTAR Server.

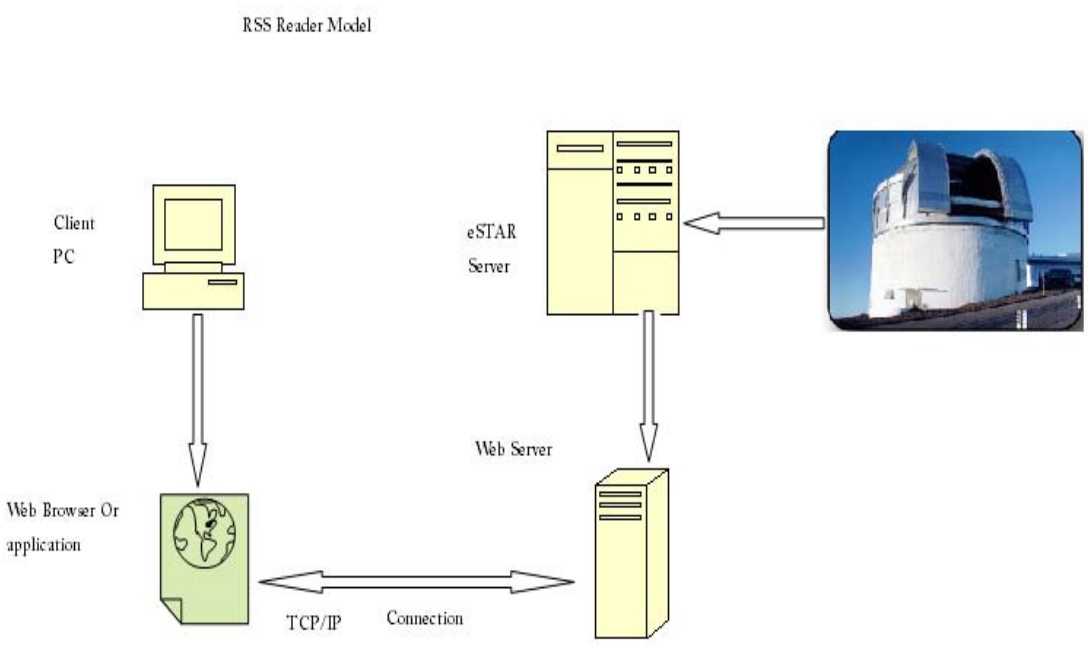

Fig 3.5 the working procedure for the RSS reader model

Users in this reader model just need to press the “reading” button and are able to read

the sorted information instead of going to a web browser and read the information line by line on the webpage. After parsing the RSS file, the final information would be human readable data, like the information of author details, event location, and event description and so on. So everyone can read them.

3.2.4 The comparison between the RSS/XML reader model and the real-time listener model.

The real-time listener model and the RSS/XML reader model, we have these two ways to see what happened in the sky recently, regarding to these two models. As table 3.3, the contrast is remarkable between them.

The real-time listener model is always connected with the server and can do the real-time updates, response, and even automatic further operations. On the contrary, because of the continuous connection, the real-time listener would take up much more system resources than the RSS reader model. Besides, it keeps receiving the test packets at most of time (“I am alive” packets). These unvalued data can be ignored, when recording the event information.

The advantages for reading the RSS feed would be easy access. Users only need a web browser or the RSS reader software to receive the updates from the web site of the Internet feed suppliers. Furthermore, because the updates from the site are sorted and valuable, users may record and file the data with ease. However the disadvantages of this model are delayed updates, manually operation and read only in contrast to the real-time listener. Each time, when users attempt to have the information, they have to manually hit the “receive” butter to trigger the downloading operation from the web server.

	Advantage	Disadvantage
Real-time Reader	Real-time updates, automatic operation, real-time response	Take up system resources for running all the time, continuous receive unvalued data.
RSS Reader	Easy access, easy to file.	Delayed updates, manually operation, and Read information only.

Table 3.3 the comparison between two models

To sum up, for using the real-time reader model and the RSS reader model, the real-time reader is preferred to running all the time to obtain the event data in real time. The RSS reader model is a good way to record the valuable information and compare the results with the one from real-time listener model.

3.3 Application Functional Design

In the details of building up the Internet feed listener application, users are able to open four main windows. They are the main window, the setting server window, the real-time listener window and the RSS reader window. The main window is the parent window, and the rest of three windows are child ones, which are working inside the main window. This design is in order to make the graphical user interface (GUI) with easy working circumstance.

3.3.1 Main form designs

The main window is the first form showing at the front of users after opening the application (As shown in Fig 3.6). It is a Multiple Document Interface design (MDI). The main window is the parent window. It contains two parts, the toolbar area and the working display area. The toolbar area contains four buttons, which are the “eSTAR Receiver” button, the “MOA Receiver” button, the “RSS Listener” button and the “server setting” button. The tool bar is shared between all child windows, which is reducing clutter and increasing efficient use of screen space. The working display area is applying the Java Desktop Pane, which is able to display all child windows on it.

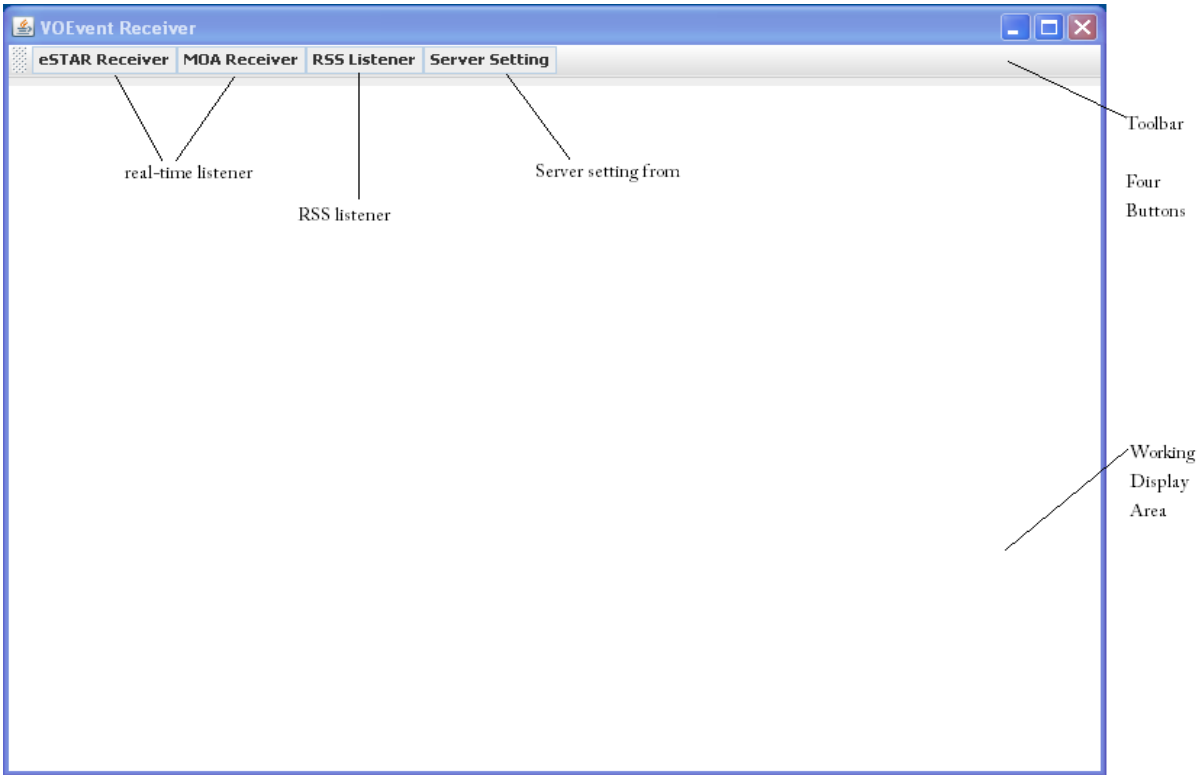

Fig 3.6 the main window of the application

Because this Java-based Internet feed listener application focuses on the real-time listener and the RSS reader, and also both the eSTAR server and the MOA server are sending the real-time signals, the “eSTAR Receiver” button and the “MOA” receiver button on the toolbar are able to trigger the real-time listening module. The “RSS listener” button triggers the static RSS reader module. As shown in Fig 3.7, the

“MainForm” class contains the four action events and one “readsetting()” function.

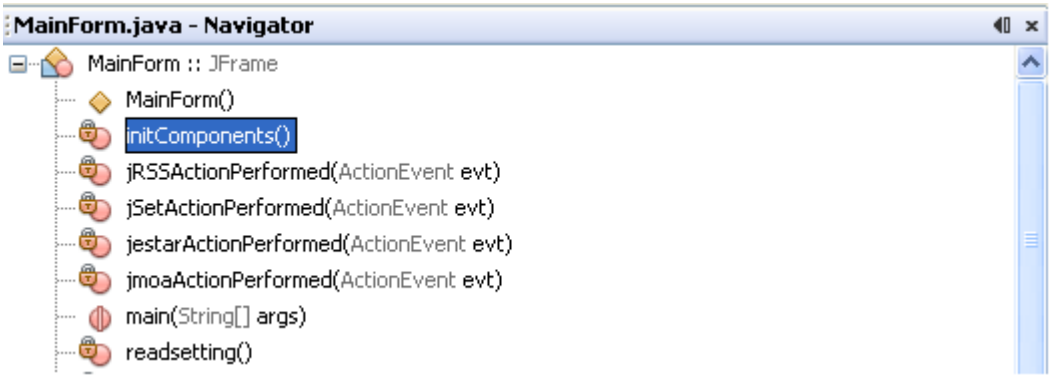

Fig 3.7 the “MainForm()” structure

The programming codes for the main form design are from “MainForm.java”. It applies Java Swing features for graphical windows and the “file” class for reading current server information when opening the application.

Because the server host names and port numbers are always changed, the application keeps the last records for the server information. The “readsetting()” codes are able to do this job. (See fig 3.8)

```
 private void readsetting(){
try{input=new Scanner (new File("setting.txt"));
}catch(FileNotFoundException fileNotFoundException){System.err.println("Error on Opening file.");}
try{
String stemp=""; int tempi=0;
while(input.hasNext()){
stemp=input.nextLine();
if (tempi==0) eh=stemp;
if (tempi==1) ep=Integer.parseInt(stemp);
if (tempi==2) mh=stemp;
if (tempi==3) mp=Integer.parseInt(stemp);
tempi++;
 }

}catch(NoSuchElementException elementException){System.err.println("File Read Errors");}
 }
```

Fig 3.8 the “readsetting()” function

The “readsetting()” uses the Java File class and the Scanner class to obtain both the information from eSTAR and MOA. The variable “eh” and “ep” have the eSTAR host name and port number for the records of last time uses. The variable “mh” and “mp” are for the MOA serer.

3.3.2 Server setting form designs

Due to the server information of eSTAR and MOA are changeable, the application has to make the server setting function for easy accessing to the servers. For example, in the eSTAR project, there are over ten servers available. The nine of them from “estar1.astro.ex.ac.uk” to “estar9.astro.ex.ac.uk” are doing the VOEvent information sending jobs. However some of them may close at sometime for maintaining purpose. Users have to check the latest status of the eSTAR servers from the eSTAR website and configure the settings. (As shown in Fig3.9)

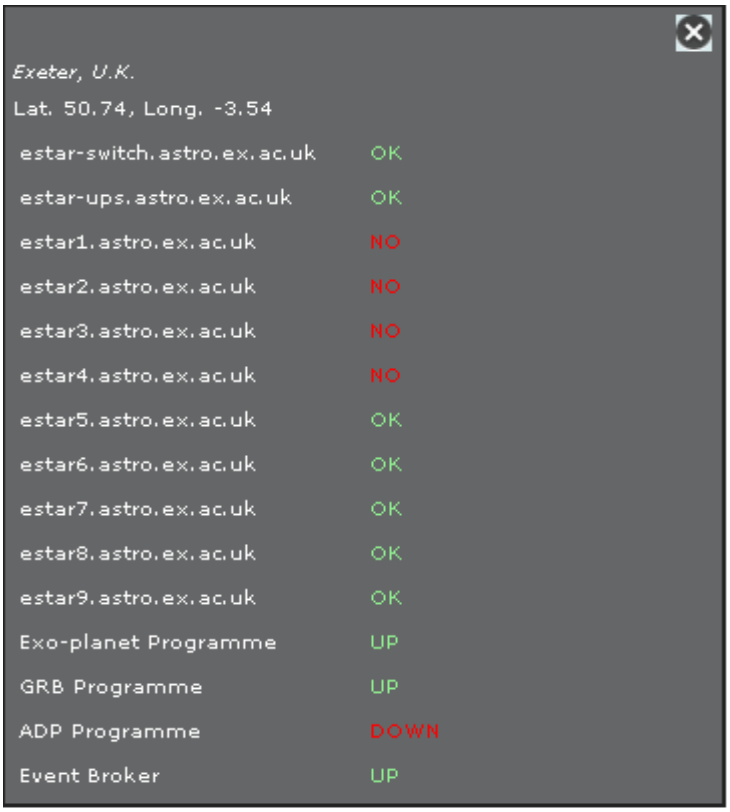

Fig 3.9 the statues of the eSTAR servers (eSTAR server 2008)

On the other hand, in order to receiving multiple server sources at the same time, setting up server configuration is the only way to do it.

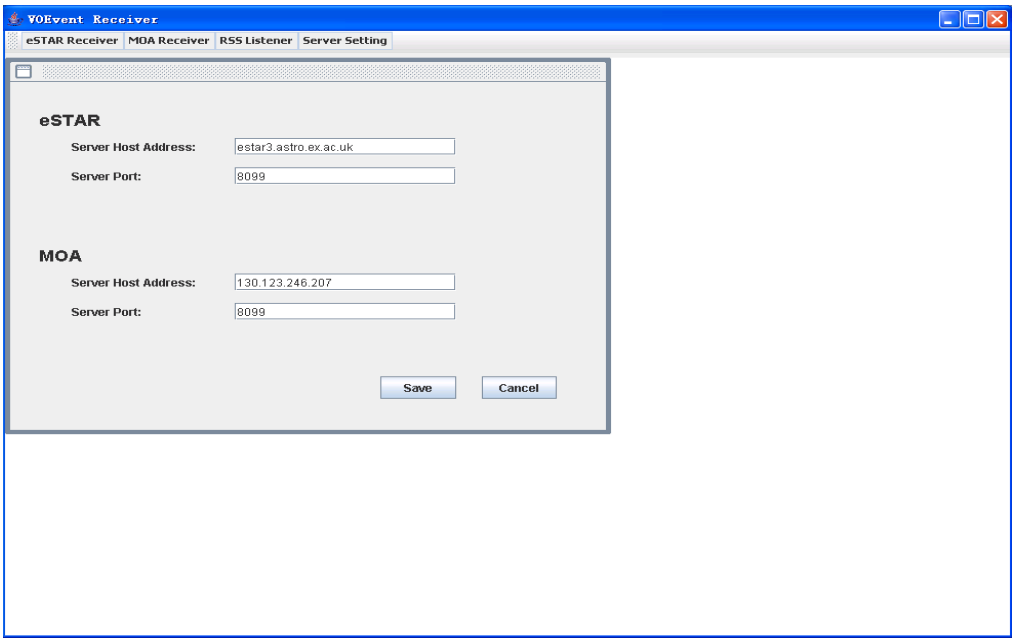

Fig 3.10 the setup window for the eSTAR and MOA servers

After hitting the “Save” button, the user inputted server information for both eSTAR and MOA would be saved into the “setting.txt” file, which is able to be read afterwards for receiving data from the servers.

3.3.3 Real-time Model Designs

The real-time model is the main part of the Internet feed listener application. It includes receiving data from the eSTAR server and the MOA server. (See fig 3.11)

In the VOEvent Receiver window, it separates to user command inputs area on the top and the information display area on the bottom. In the top-left, it shows the current server information, which can be modified in the server setting window. The “Start

Listening” and the “Stop Listening” button control the looping for receiving the data from the server. When the “Start Listening” button is hit, it is disabled until the “Stop Listening” button is pressed. The choice of the “Validation Check” is optional for applying the validation check for the received VOEvent packet. The “Save To File” and the “See Record” button are designed for backup purposes. Users are able to save the current parsed information into a file and see the whole record at anytime on the display area.

In the display part, it contains two text areas. One is on the left side. It shows the connecting information and parsed information. The other one is on the right side, and it displays the original raw VOEvent packet data. This raw data can be hidden by clicking the “Hide Packet Details” button.

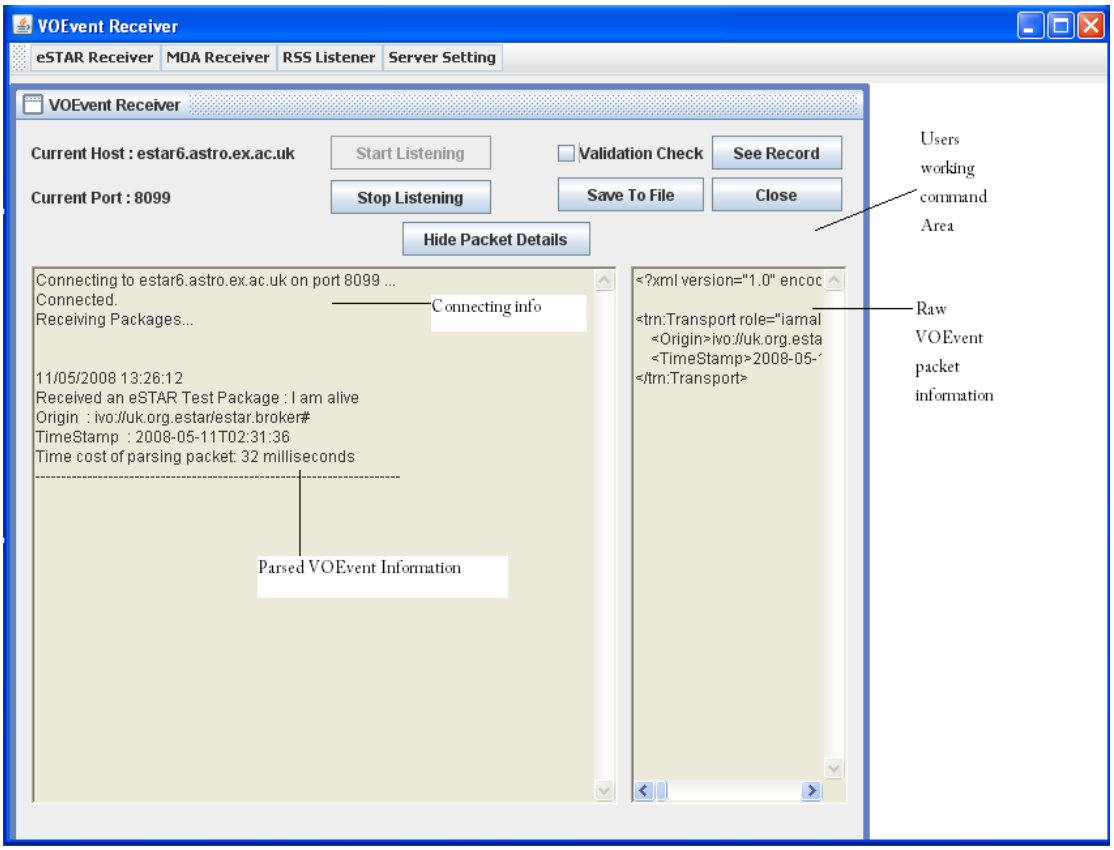

Fig 3.11 the real-time astronomical Internet feed listener

As described on 3.2.2, the real-time model would contain five parts, connecting,

reading, responding, parsing, and displaying. The following content represents them in details.

3.3.3.1 Connecting

The “connect()” method is applying the Java.net.Socket class as shown on fig 3.12. It creates a new socket connection with the string of the “host” name and the integer number of “port” number. After the connection is successful, the input data stream will be hold in variable “in” and any output data stream will be in variable “out”.

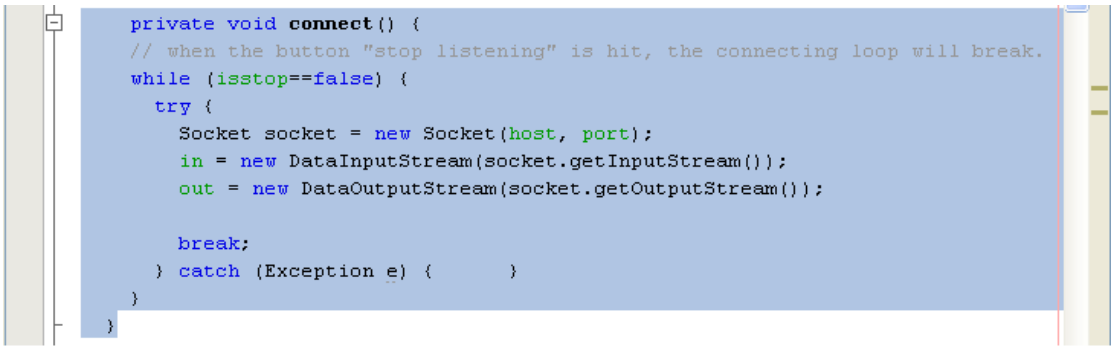

Fig 3.12 the “connect()” method

This is the basic Java socket operation. The program creates a new socket with the “Socket (host, port)” constructor, and the socket attempts to connect to the remote host. Once the connection is established, the local and remote hosts get input and output streams from the socket and use those steams to send data to each other. The data may be transmitted by FTP or HTTP protocols after that.

3.3.3.2 Listening

The “listening” method is the core part of the Internet feed reading application. Its tasks are receiving signals from the server, sending the responds back to the server and controlling the data flow to the parsing class (See fig 3.13 for the part of “listen()” method).

```
private void listen() {
   while (isstop==false) {
    try {
     int available = in.available();
     byte[] byteArray = new byte[available];
     available = in.read(byteArray, 0, available);
   ParsePack pp=new ParsePack();
   DateFormat dateFormat=new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    // when the button "stop listening" is hit, the listening loop will break.
     if ((available > 0)&&(available<5)){strtemp=new String(byteArray);
                        response = strtemp;
                        out.writeChars(response);}
     if (available>5){strtemp=new String(byteArray);
          if(strtemp.endsWith("</trn:Transport>\n")){response = pp.make_response();
strrec=strtemp;
if (jChkVal.isSelected()==true) {pp.parPackage(strrec,true);}
else {pp.parPackage(strrec,false);}
//showing the raw string from server
       textA2.append(strrec+newline);
       Date d= new Date();
     textA1.append( dateFormat.format(d) + pp.getString()+newline+newline);
                        //add to file string, backup the result.
 fileString=fileString+dateFormat.format(d)+pp.getString()+newline;
                        out.writeChars(response);
                        strrec="";}
 if(strtemp.endsWith("</VOEvent>\n")){strrec=strrec+strtemp;
                    //showing the raw string from server
                    textA2.append(strrec+newline);
if (jChkVal.isSelected()==true) {pp.parPackage(strrec,true);}
else {pp.parPackage(strrec,false);}
Date d= new Date();
 textA1.append( dateFormat.format(d) + pp.getString()+newline+newline);
                    fileString=fileString+dateFormat.format(d)+pp.getString()+newline;
                    response = strrec;
                    out.writeChars(response);
                    strrec="";
                  }}
```

Fig 3.13 the "listen ()" method

In the research of developing the application, a rule for the eSTAR server has been discovered. The VOEvent data sent from eSTAR server can be classified to three different data types as the following table (Table 3.4).

Data Received from the eSTAR server	Stream(s)	Length (bytes)	Description
É	One	4	Server Testing String Before sending data
<?xml version="1.0" encoding ="UTF-8"?> <trn:Transport role="iamalive"… … </trn:Transport>□ □	One	<500	“I am alive” String, sending 60 seconds once, when no events are updated.
<?xml version="1.0" encoding ="UTF-8"?> <VOEvent id="… … </VOEvent>	More than one	>500	Useful VOEvent data when events are updated to the server.

Table 3.4 the received data from eSTAR server

Regarding to the types of the data, the listening method has to distinguish all the incoming data, so that it is able to give the corresponding responding strings to the server and also parse the data into user readable information. For example, if the “listen ()” method has a data with the length of greater than four and also ending with “</trn:Transport>, then the data must be a “I am alive” packet.

When the application receives the VOEvent data, most of formal event data are packed into two or more packets in transit, which is for the compact size and quickly transmittable speed over the Internet. It is the reason that the listening method has to keep looping to search the ending signals, like “</trn:Transport>” or “</VOEvent>”, when finding them, the completed VOEvent data can be received successfully, and be ready to pass to the parsing class after that.

3.3.3.3 Responding

The responding part is one of the important parts as well. When the Internet feed listener application receives a VOEvent packet, if the application does not send back a response, then the server will cut off the connection between them and no more data will be received after that. Since the application needs a real-time continuous listening, the responding packet has to be designed in the research.

According to the three types of receiving data, the application has to make a corresponding responding data for each type. See the table 3.5 for details.

Data Received from the eSTAR server	Responding data to the eSTAR server
É	É
<?xml version="1.0" encoding ="UTF-8"?> <trn:Transport role="iamalive"… … </trn:Transport>□ □	<?xml version="1.0" encoding ="UTF-8"?> <trn:Transport role="**ack**"… … </trn:Transport>□ □
<?xml version="1.0" encoding ="UTF-8"?> <VOEvent id="…" role="observation" … </VOEvent>	<?xml version="1.0" encoding ="UTF-8"?> <VOEvent id="…" role="**ack**" … </VOEvent>

Table 3.5 the receiving and sending data formats

When the server receives the packet with the role of "ack" string, the server will recognize the request of the client and continue to transport the data through. To make the responding string, the application applies the “make_response()” method, as shown on fig 3.14.

```
public String make_response(){
String nstrn  = "\"http://www.telescope-networks.org/xml/Transport/v0.1\"";
String nsxsi  = "\"http://www.w3.org/2001/XMLSchema-instance\"";
String schema = "\"http://www.telescope-networks.org/xml/Transport/v0.1 v0.1.xsd\"";
Date d=new Date();
String response = "<?xml version='1.0' encoding='UTF-8'?>" +
          "<trn:Transport role=\"ack\" version=\"0.1\""  +
          " xmlns:trn=" + nstrn  +
          " xmlns:xsi=" + nsxsi  +
          " xsi:schemaLocation=" + schema + ">\n" +
          "<Origin>ivo://uk.org.estar/estar.ex#</Origin>" + "\n" +
          "<TimeStamp>" + d + "</TimeStamp>" + "\n" +
          "</trn:Transport>" + "\n";
return response;
}
```

Fig 3.14 the “make_response()” method

After creating the responding data, the application will send it into the output stream via the opened socket.

3.3.3.4 Parsing

The main feature of this VOEvent Internet feed listener application is the parsing part. At the first stage, the VOEvent packet data that the application receives is the original XML-based data. We call it the raw data. In order to interpret it to standard information for easy reading, a parsing part has to be applied.

On the other hand, for some possibilities, if the application receives an uncompleted VOEvent packet or the VOEvent packet comes with an incorrect format, the parsing method is able to find the problem and alert users. At this point, parsing VOEvent packet is necessary.

1) An example of parsing result

The following figures (As shown in Fig 3.15 and Fig 3.16) are the comparison between the data before parsing and the information after parsing.

The <Who> tag of a VOEvent packet before parsing (As shown in Fig 3.15).

```
<Who>
    <PublisherID>ivo://raptor.lanl/organization</PublisherID>
    <Contact principalContact="true">
      <Name>Robert White</Name>
      <Institution>LANL</Institution>
      <Communication>
        <AddressLine> Los Alamos National Laboratory PO Box 1663 ISR-1,
          MS B244 Los Alamos, NM 87545 </AddressLine>
        <Telephone>+1-505-665-3025</Telephone>
        <Email>rwhite@lanl.gov</Email>
      </Communication>
    </Contact>
    <Date>2005-04-15T14:34:16</Date>
</Who>
```

Fig 3.15 the original <Who> tag before parsing

The information after parsing is shown in Fig 3.16.

```
Publisher ID: ivo://raptor.lanl/organization
Name        : Robert White
Institution : Los Alamos National Laboratory
Address     : Los Alamos National Laboratory PO Box 1663 ISR-1,
              MS B244 Los Alamos, NM 87545
Telephone   : +1-505-665-3025
Email       : rwhite@lanl.gov
Date        : 2005-04-15T14:34:16
```

Fig 3.16 the information from <Who> tag after parsing

2) Parsing principles

The main motivation in the parsing method is that looking for the related tag names. For example, because a standard VOEvent packet must have one or more unique standard tags, like, <Who>, <What>, <WhereWhen> etc… The parsing class is able to find these tags if they are available, and is able to apply the DOM tree searching method to find the child information. The principle of parsing class is as the figure below (in Fig 3.17).

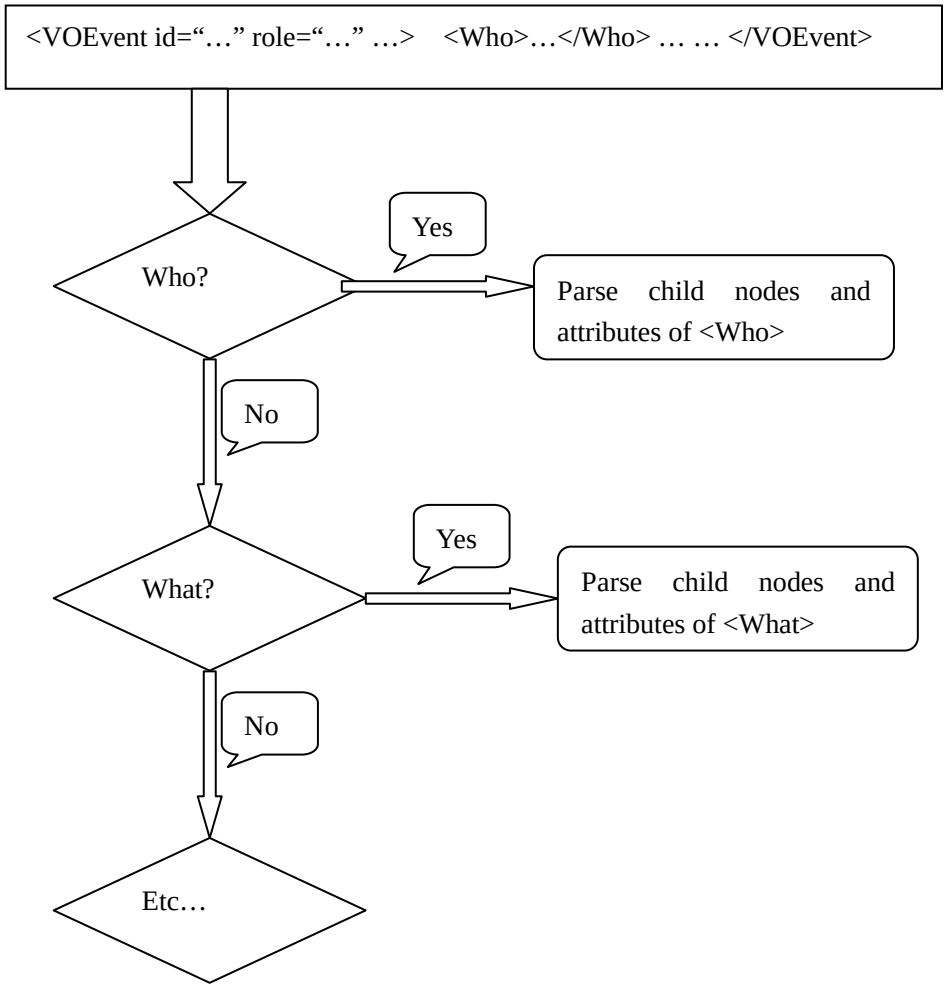

Fig 3.17 the parsing steps for the VOEvent packet

The parsing class is able to find the particular tag firstly, and then make a node list for it, if the tag exists in the VOEvent packet. After that, the parsing class can search every child node under the top parent node. If the child node has its own child node, the parsing class will treat the child node as a new parent node and do the parsing procedure over it again.

In Fig 3.18 and Fig 3.19, they present the tree structure of the <Who> tag of the VOEvent packet and the parsing steps by the parser of the application.

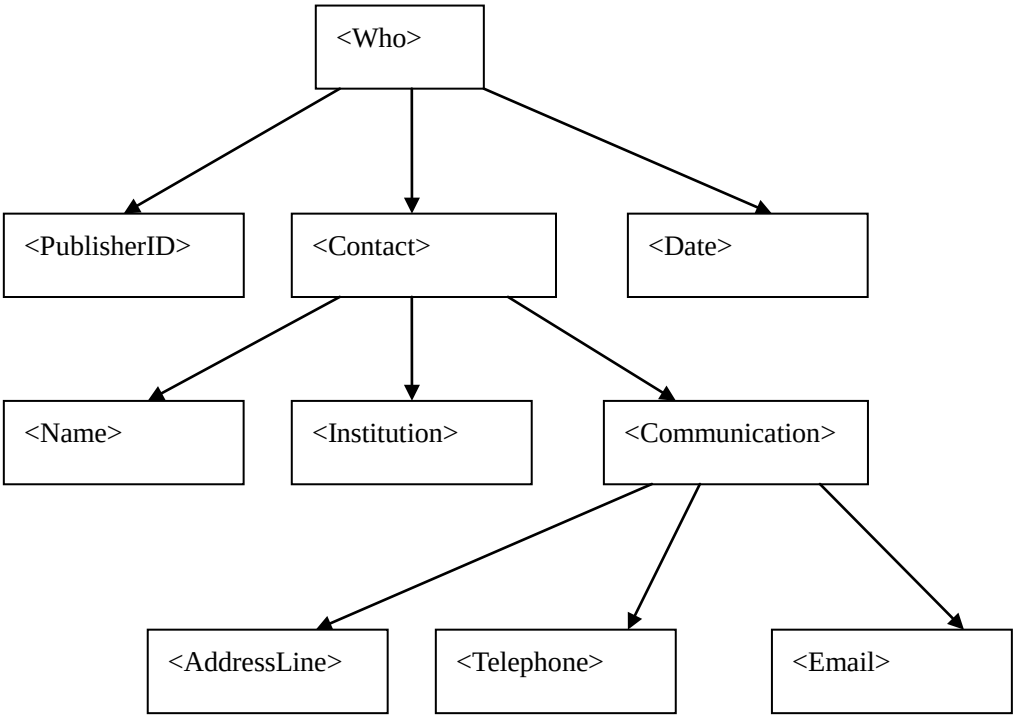

Fig 3.18 the tree structure of the <Who> tag of VOEvent packet

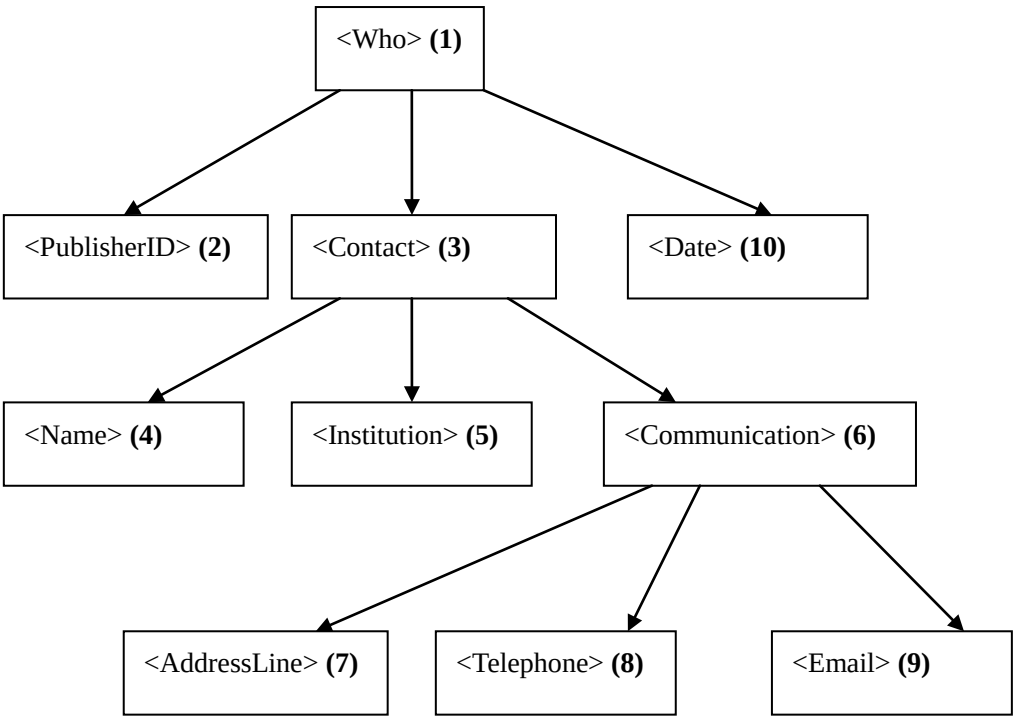

Fig 3.19 the parsing steps for the <Who> tag of VOEvent packet

3) Programming design for the parsing class

In the programming codes, the parsing part involves in the “ParsePack.java” class. The VOEvent Internet feed listener application applies the Java DOM tree searching technique and XML parser class (See Fig 3.20).

```
import java.util.Date;
import java.io.*;
import java.net.URL;
import org.xml.sax.SAXException;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
```

Fig 3.20 the header imports for the “ParsePack.java” class

For the node types in the VOEvent data packet, there are two types available. One is the standard one, which comes with unique name for each tag. Like the VOEvent tag <Who>, <How>, <Why> (See Fig 3.15). The other type comes with parameters in it, which are under the same tag names. Like the VOEvent tag <What>, <WhereWhen> (As shown in Fig 3.21). The parameters here, we call them the attributes of the node. The parsing class is able to deal with them in different ways.

```
<What>
  <Group name="SQUARE_GALAXY_FLUX">
    <Param name="counts" value="73288" unit="ct" ucd="phot.count"/>
    <Param name="peak" value="1310" unit="ct/s" ucd="arith.rate;phot.count"/>
  </Group>
  <Param name="seeing" value="2" unit="arcsec" ucd="instr.obsty.site.seeing"/>
  <Reference uri="http://raptor.lanl.gov/data/lightcurves/235649409"/>
  <Description>This is the light curve associated with the observation.</Description
</What>
```

Fig 3.21 the attributes of the node in the <What> tag

For parsing the first type of VOEvent data, the parsing class is able to seek the unique name following the node list from the top to the bottom of the DOM tree (See Fig 3.22).

```
if(doc.getElementsByTagName("Who").getLength()>0){
System.out.println("*WHO*");
NodeList whoNL = doc.getElementsByTagName("Who").item(0).getChildNodes();
Node whoN;
for (int i = 0; i < whoNL.getLength(); i++) {
whoN = whoNL.item(i);
if (whoN.getNodeName().equals("PublisherID")) { strend=strend+ "\nPublisher ID: "+ whoN.getFirstChild().getNodeValue();}
if (whoN.getNodeName().equals("Date")) { strend=strend+ "\nPublished Date: "+ whoN.getFirstChild().getNodeValue();}
if (whoN.getNodeName().equals("Contact")) {  NodeList contactNL=whoN.getChildNodes();
  for(int j=0;j<contactNL.getLength();j++){
   Node contactN=contactNL.item(j);
   if (contactN.getNodeName().equals("Name")) { strend=strend+ "\nPublisher Name: "+ contactN.getFirstChild().getNodeValue();}
   if (contactN.getNodeName().equals("Institution")) { strend=strend+ "\nPublisher Institution: "+ contactN.getFirstChild().getNodeValue();}
   if (contactN.getNodeName().equals("Communication"))
    {NodeList commuNL=contactN.getChildNodes();
      for (int k=0;k<commuNL.getLength();k++){
       Node commuN=commuNL.item(k);
       if (commuN.getNodeName().equals("AddressLine")) { strend=strend+ "\nPublisher AddressLine: "+ commuN.getFirstChild().getNodeValue();}
       if (commuN.getNodeName().equals("Telephone")) { strend=strend+ "\nPublisher Telephone: "+ commuN.getFirstChild().getNodeValue();}
       if (commuN.getNodeName().equals("Email")) { strend=strend+ "\nPublisher Email: "+ commuN.getFirstChild().getNodeValue();}
                                                  }}
                                          }}
```

Fig 3.22 the parsing method for the type with unique tag names

For parsing the second type of nodes with attributes, the parsing class is able to make a loop to analyse all of elements with the same parameter names (See Fig 3.23).

```
if (whatN.getNodeName().equals("Group")) {
   groupNL=whatN.getChildNodes();
   for(int j=0;j<groupNL.getLength();j++){
   groupN=groupNL.item(j);
    if (groupN.getNodeName().equals("Param"))
    { Element et=(Element) groupN;
      strend=strend+ "\n"+et.getAttribute( "name" )+"  "+et.getAttribute( "value" )+"  "+et.getAttribute( "unit" )+"  "+et.getAttribute( "ucd" );}
                                        } }
```

Fig 3.23 the parsing method for the type with attributes

4) The validation check design in the parsing part

Also the validation check is another main part for the parsing class. It applies the Java XML parser class (javax.xml.parsers) and the Java XML validation class (javax.xml.validation). There are five steps when doing the validation check in this application as following.

- Find the schema
- Load the schema into memory
- Load the VOEvent packet (in XML format)
- Check the packet information against the schema
- Give results

With the Java XML class, the application makes it with ease. The following figure is an example of the validation check (See Fig 3.24).

```
//Check Validation of the packet
// 1. Lookup a factory for the W3C XML Schema language
startTime = System.currentTimeMillis();
SchemaFactory factory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
// 2. Compile the schema.
//1.schema location from file. 6sec
File schemaLocation = new File("D:/Oct/XMLtest/XMLtest/VOEvent-v1.0.xsd");
//2. Schema location from URL. 17sec
Schema schema = factory.newSchema(schemaLocation);
// 3. Get a validator from the schema.
Validator validator = schema.newValidator();
// 4. Parse the document you want to check.
Source source = new StreamSource("D:/Oct/XMLtest/XMLtest/example1-v1.0.xml");
// 5. Check the document
try {
    validator.validate(source);
    strend=strend+ "\nValidation Check: This packet is valid.";
}
catch (SAXException ex) {
    strend=strend+ "\nValidation Check: This packet is not valid because";
    strend=strend+ ex.getMessage();
}
```

Fig 3.24 the validation check in the parsing part

3.3.3.5 Displaying

In this VOEvent Internet feed listener application, the displayed information shows on both text areas. The left side shows the connection information and parsed information by the application. The right side shows the original raw VOEvent packet data.

In the left column, the parsed information also may contain the result of the validation check and time costs for both validation checks and parsing time (As shown in Fig 3.25). By clicking the “Show/Hide Packet Details” button, the right column would be shown or hidden, which is giving a clear easy operating interface (Fig 3.26).

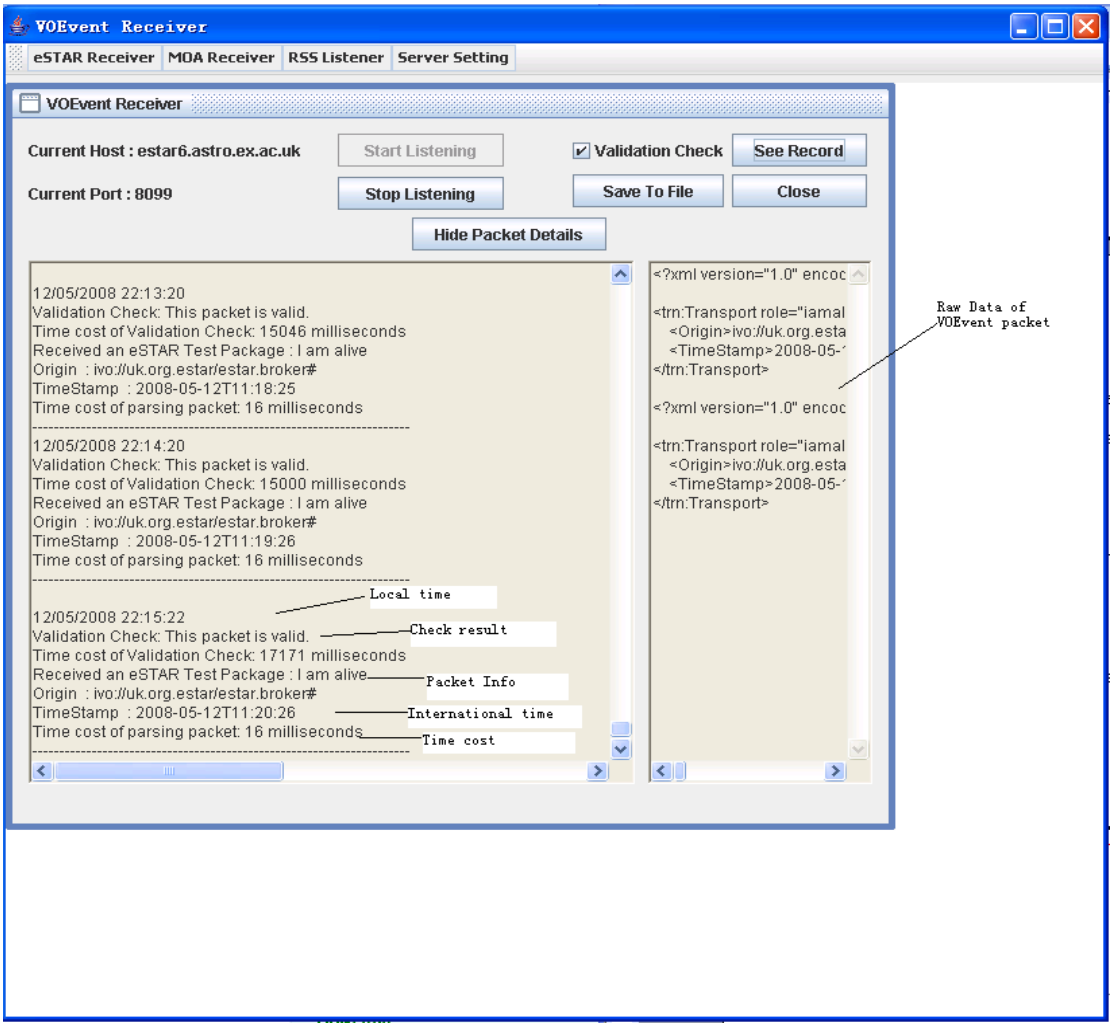

Fig 3.25 the displaying part of the real-time listener

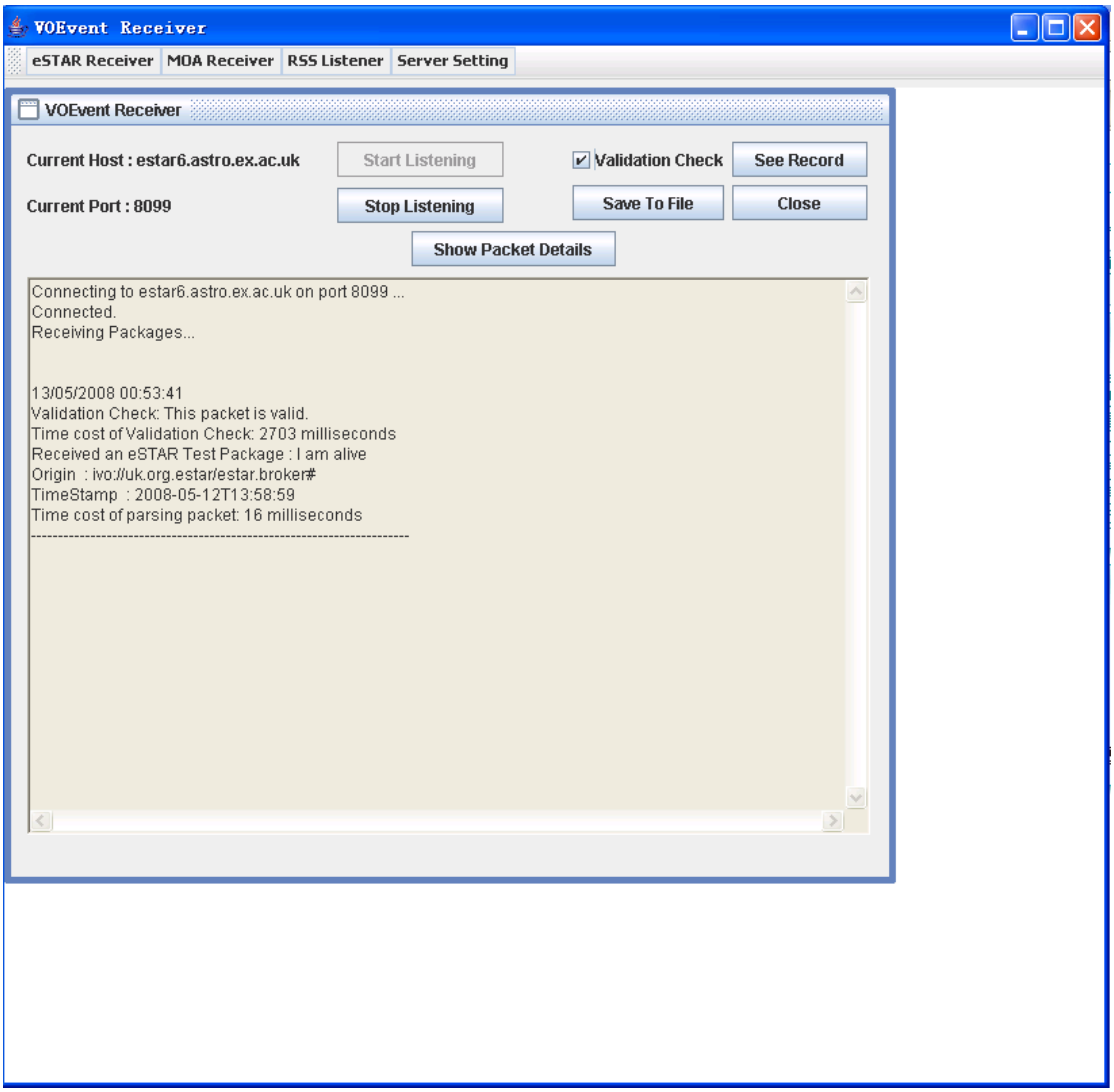

Fig 3.26 the displaying part of the real-time listener without the raw info

3.3.3.6 Recording

Recording what users have received is necessary as well. This VOEvent Internet feed listener application is able to record the parsed information into a file by clicking the "Save To File" button and easy to see the record by clicking the "See Record" button. To do it, the application applies the "ReadTextFile.java" to do the file operation. A text file would be created and updated as the name of "readResults.txt"

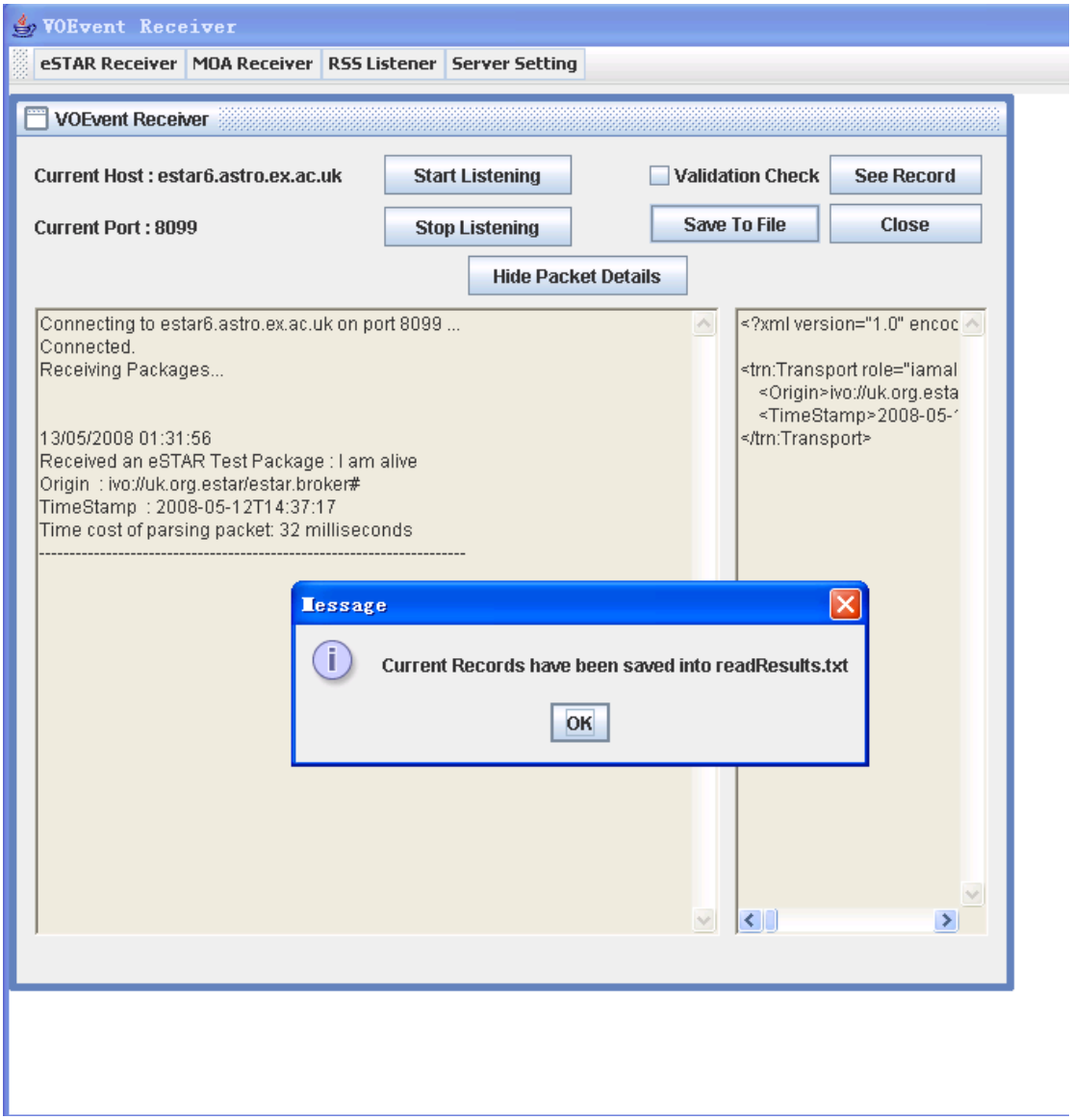

Fig 3.27 the recording part of the real-time listener

3.3.3.7 Thread technique for the time critical programming

Because the real-time model focuses on the time critical programming, the "listening" model has to be always on, which means the application has to keep looping to fetch data from servers. This kind of loop can take up all the system resources, like memory and processor resource. The application even cannot stop the loop, after it starts.

For solving this problem, the thread technique can be applied, and it works very well. What the application does for using the thread technique is as following steps.

- Make a container for the thread
- Put the looping parts (“connect()” and “listen()”) into the container (the thread)
- Run the thread separately from the resource for the application.

```
//create a thread for JForm for easily stopping the listening loop at anytime.

  private class Packer extends Thread
  {
    public void run()
    {
     textAl.setText("");
     textAl.append("Connecting to " + host + " on port " + Integer.toString(port) +
     connect();
     if (isstop==false){
     textAl.append("Connected.\n"+"Receiving Packages..."+newline+newline+newline);
     listen();}else{textAl.append("\n\tStopped by user\n");}
    }
```

Fig 3.28 the thread technique applied in the real-time listener

3.3.4 RSS Model Designs

In addition to running a real-time VOEvent feed listener, the application is able to receive the RSS 2.0 feeds from the website of the eSTAR project. The eSTAR project also makes event notices available via RSS 2.0 feeds and the information in the eSTAR website is keeping updated very fast. Once the eSTAR server has the event information, they would update it as soon as they can. So the RSS model of the application is another way to have the VOEvent packet information over the Internet.

The RSS model has the following features.

- Editable RSS source file
- Preview latest available feeds
- Parse feeds to easy looking information
- Link to the webpage of the original VOEvent packet

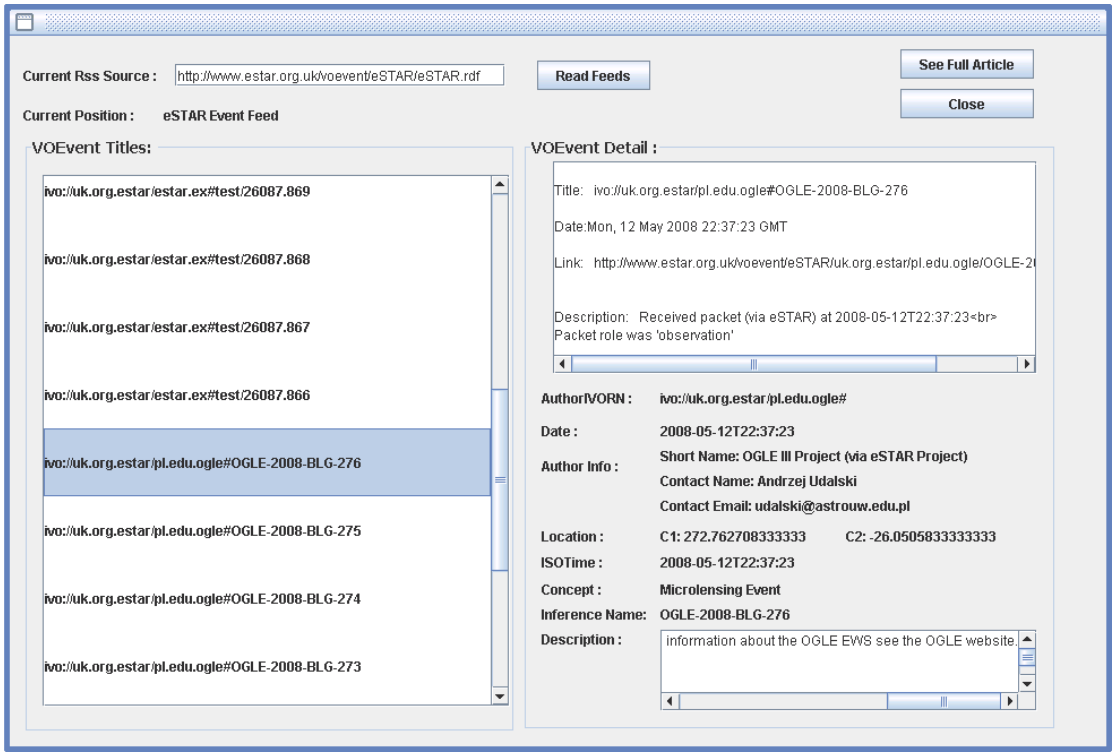

Fig 3.29 the RSS reader

After inputting the RSS source path, (the default URL is the eSTAR VOEvent feed supplier link.) The application is able to show all titles of the latest feeds row by row on the left side of the window. Users may click any of these titles. The corresponding parsed information would be shown at the right side of the window. If users like to see the original VOEvent packet of the current feed, by clicking the “See Full Article” button, a web browser would pop up and show the XML-based VOEvent packet. (As shown in Fig 3.30)

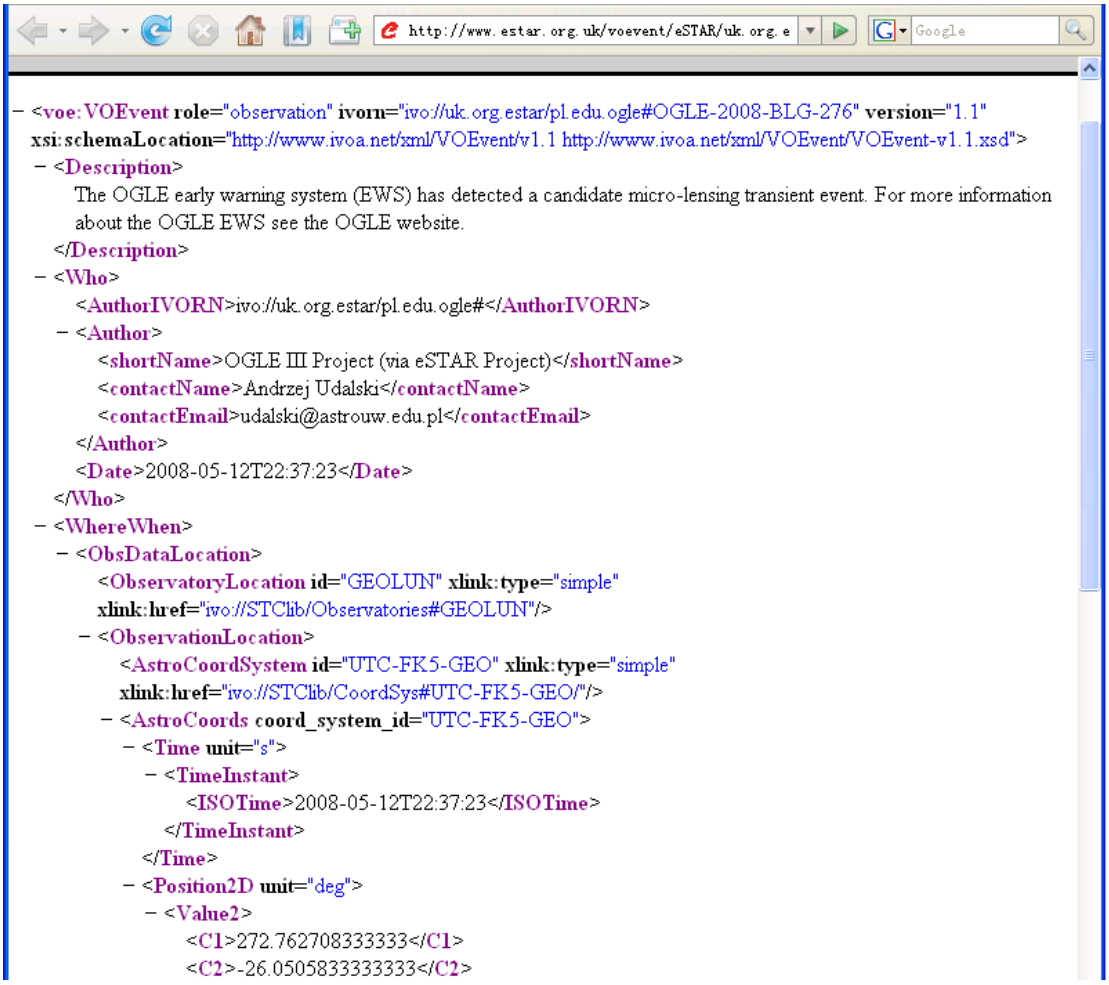

Fig 3.30 the original VOEvent information in the eSTAR webpage

In the programming part, the RSS model applies a Java RSS Utilities package, called “rssutils.jar”, which is developed by Sun developer network. It contains a set of custom JSP tags which make up the RSS Utilities tag library, and a flexible RSS Parser (Java RSS parser, 2004)

After adding the “rssutils.jar” library into the program class path, the application is able to use the specified parser to simplify the parsing procedure. As shown in Fig 3.31, the imported header files started with “com.sun.cnpi.rss” are from the “rssutils” library.

```
package voevent;
import com.sun.cnpi.rss.elements.Category;
import com.sun.cnpi.rss.elements.Item;
import com.sun.cnpi.rss.elements.Rss;
import com.sun.cnpi.rss.parser.RssParser;
import com.sun.cnpi.rss.parser.RssParserFactory;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.lang.reflect.Method;
import java.net.URL;
import java.util.Collection;
import java.util.Iterator;
import java.util.Vector;
import javax.swing.DefaultListModel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import java.awt.event.*;
import java.awt.*;
```

Fig 3.31 the imported header files for the RSS reader model

In Fig 3.32, it shows how the RSS model works. Firstly, create a parser by using the Java rssutils library. Secondly, read the RSS source URL from the website of the feed suppliers. Thirdly, the parser looks for every feed in the source RSS URL. Finally, the parser is able to read the every feed data and show the parsed information on the display area.

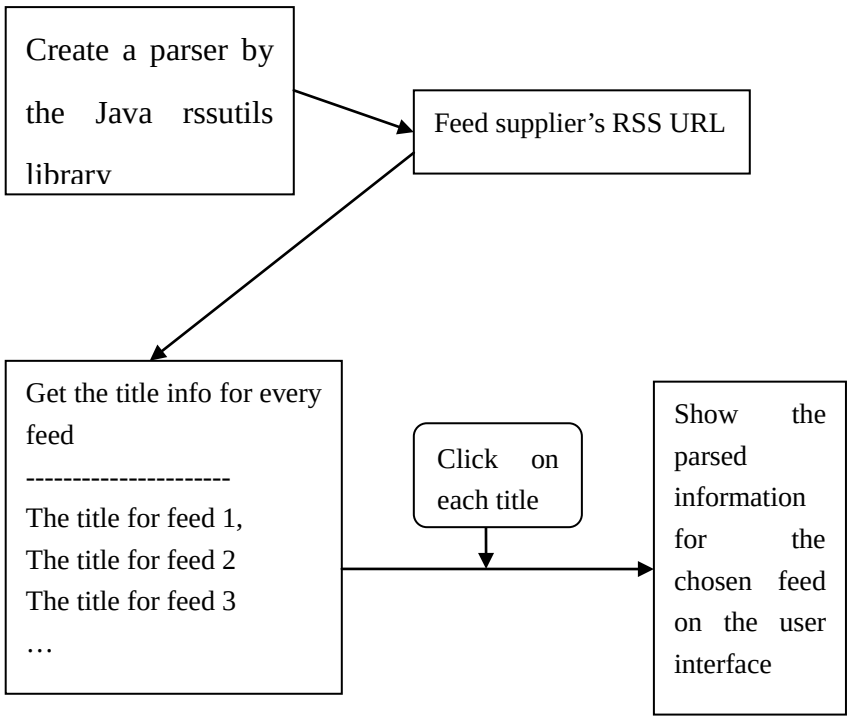

Fig 3.32 the working procedure for the RSS reader model

In Fig 3.33, it shows the main coding part for retrieving the Internet feeds over the feed suppliers' website.

```
public void readRSSDocument() throws Exception{
    String strTitle= new String();
    String str= new String();
    RssParser parser = RssParserFactory.createDefault();
    //Parse a RSS/XML site
    Rss rss = parser.parse(new URL(jTFSource.getText()));
    //give current RSS web's catologe
    jName.setText((rss.getChannel().getTitle().toString()));
    //Get all XML elements in the feed
    Collection items = rss.getChannel().getItems();
    if(items != null && !items.isEmpty()) {
        for(Iterator i = items.iterator();
        i.hasNext();
        System.out.println()) {
            Item item = (Item)i.next();
            str="\nTitle:   " + item.getTitle()+"\n\nDate:"+ item.getPubDate()
            +"\n\nLink:   " + item.getLink()+"\n\n\n";
            strTitle=""+item.getTitle();
            vecStr.addElement(str);
            str=""+item.getLink();
            vecLink.addElement(str);
            ((javax.swing.DefaultListModel)jListl.getModel()).addElement(strTitle);

        }
```

Fig 3.33 the "readRSSDocument()" class

4. Project Deployment

This Java-based Internet feed listener application is developed by NetBeans IDE (version 5.0). The Java version is JDK1.5.0-13. The application can be executed either in the NetBeans platform or in the Java Web Start environment.

4.1 Enterprise application deployment

An enterprise application, named as “VOEventReceiver” was created. It contains eight independent Java files and one add-on Java library. The physical file location and structure is shown in Fig 4.1.

The requirements and notices for running the application in NetBeans IDE:

1. The “rssutils.jar” file includes the parsing functions for the RSS model of the application, and therefore, it must be added before running the application.
2. The application would generate three text files, named as “readResults.txt”, “setting.txt” and “vo.xml”. All of them would be in the root folder of the application.
3. The application is tested under Microsoft Windows system.
4. All Java classes can be compiled by JDK1.5 or above version.
5. NetBeans IDE 6.0 is able to run this application as well (The “rssutils.jar” file has to be re allocated).
6. In the first deployment of the application, the eSTAR server would be set on the default server (Host name: estar3.astro.ex.ac.uk; Port number: 8099). A reset

operation may be required.

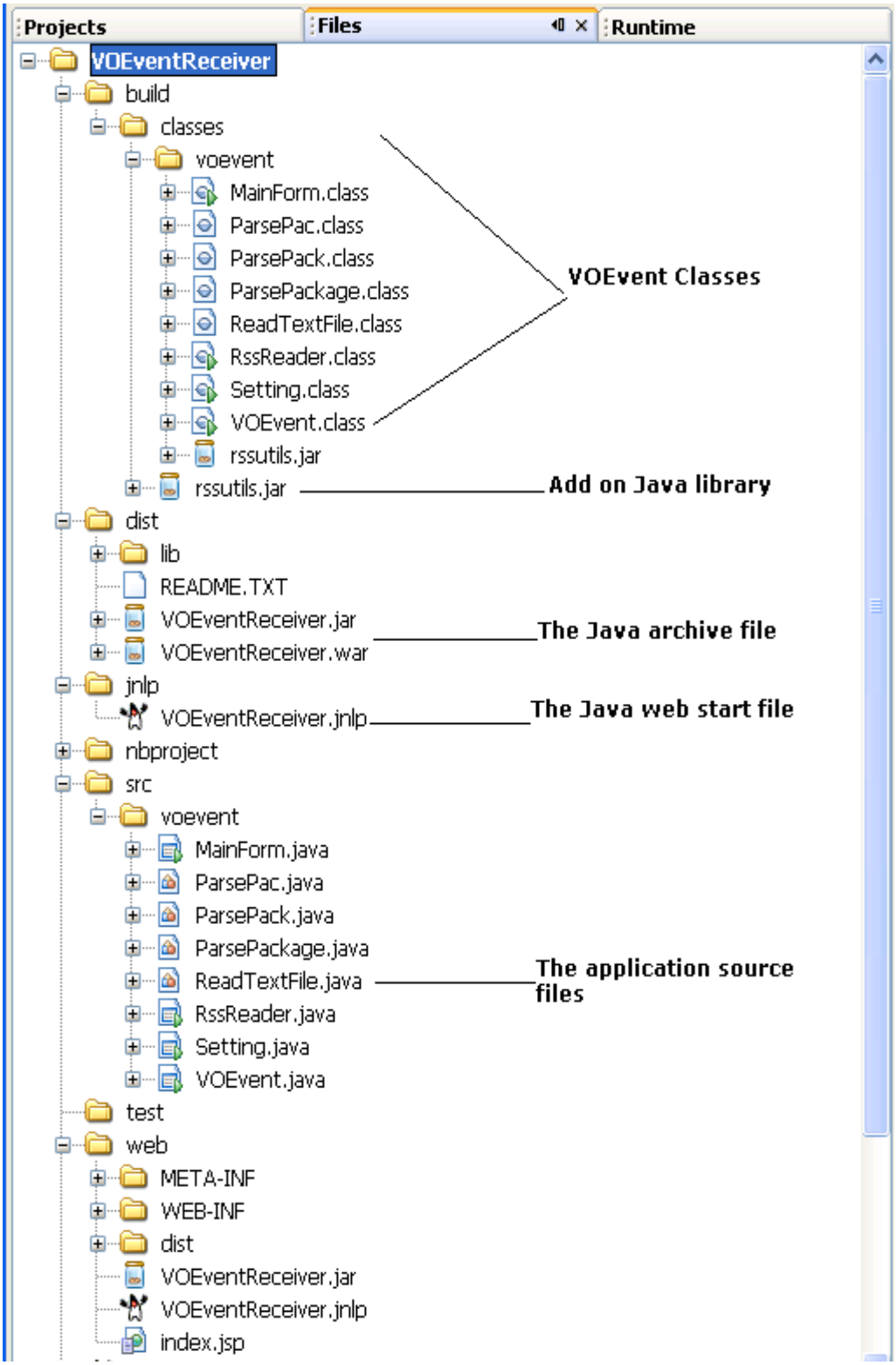

Fig 4.1 the physical files structure

4.2 Java Web Start Deployment

This application is also can be deployed by using Java Web Start technology. The NetBeans IDE has its own module for Java Web Start. To enable it, firstly, install the plug-in Java Web Start module from the NetBeans update center. Secondly, activate the Java Web Start function at the VOEventReceiver project. Thirdly, edit the "VOEventReciever.jnlp" file to define the JAR resource. Finally, the application is able to deploy and run in the Java Web Start environment (as shown in Fig 4.2). In the right side of the Fig 4.2, it is the configuration for the "VOEventReceiver.jnlp" Java Web Start file.

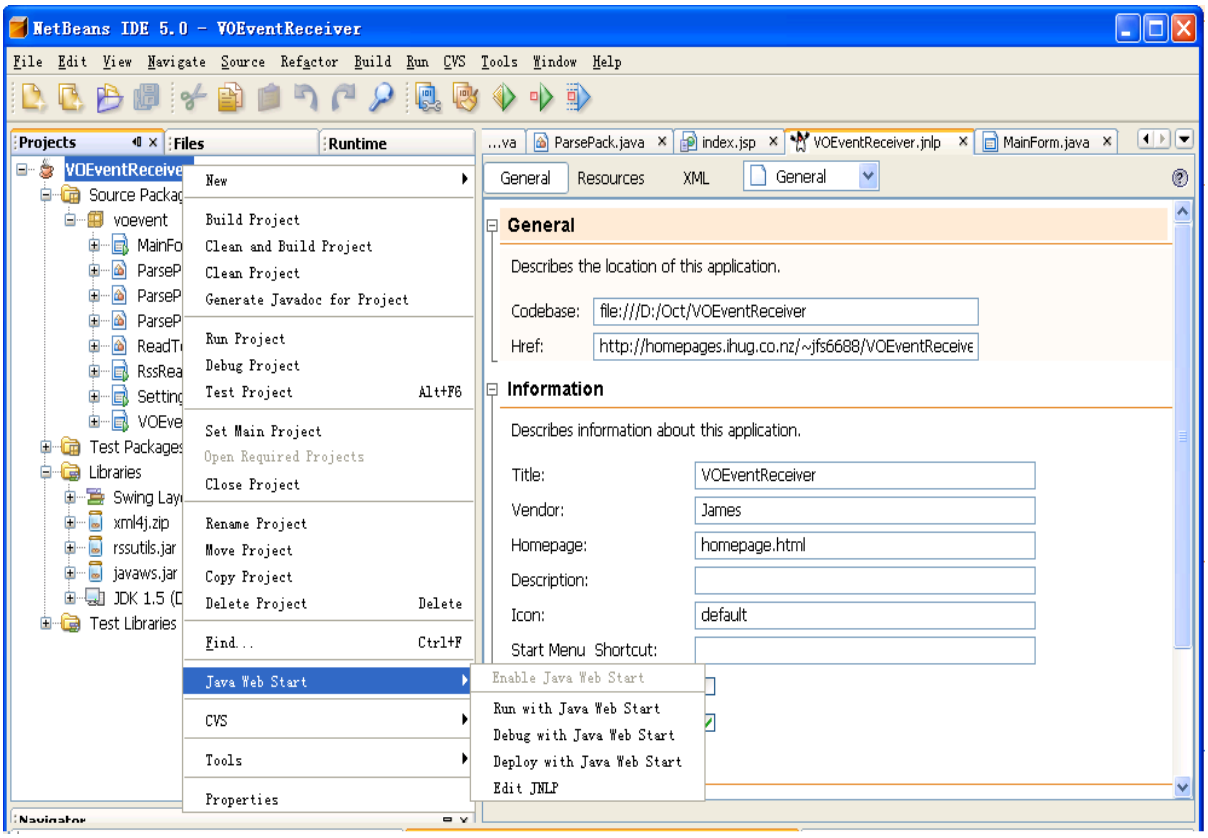

Fig 4.2 the application deployment with the Java Web Start technology

The main advantage of the Java Web Start is that after the application is uploaded to

the server, the client side user only needs a web browser to access the application site and download the application into the local PC. After it, the application is able to be executed. Also at the same time, the application can be updated automatically from the Java Web Start server. In Fig 4.3, it shows the Java-based Internet feed listener application running in Java Web Start.

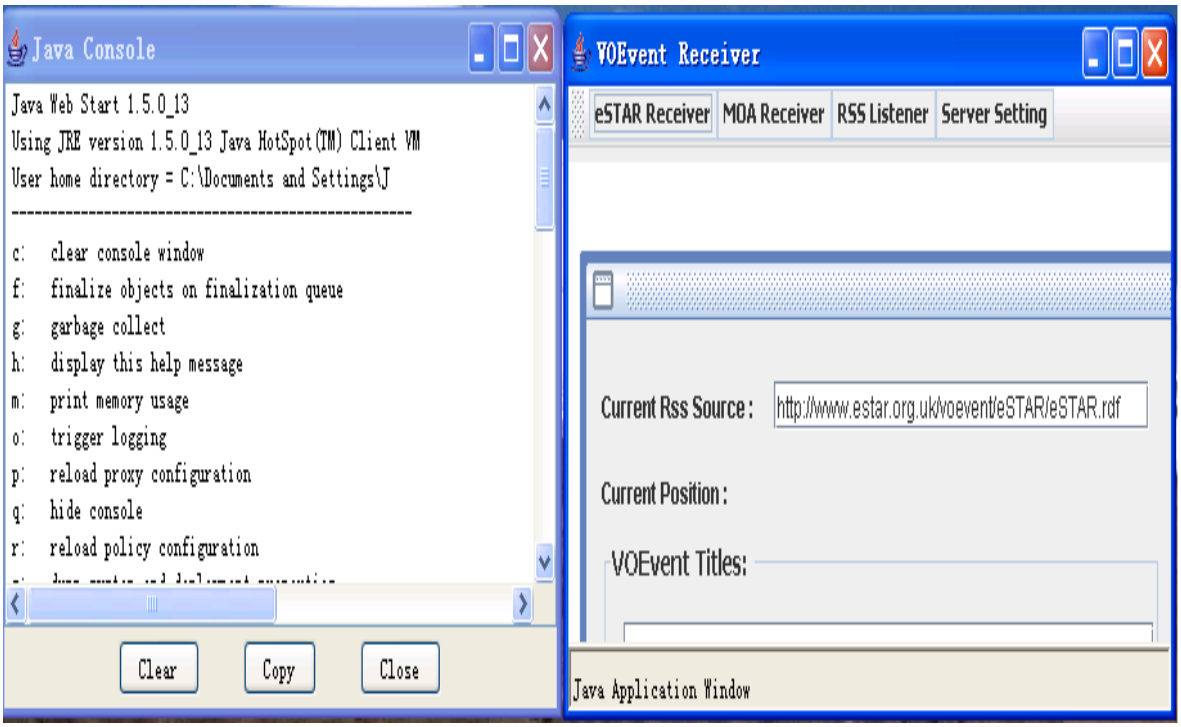

Fig 4.3 the application running in Java Web Start

5. Project Performance

This chapter focuses on discussing the performance of the Java-based Internet feed listener application, analyzing the possible methods and giving the one with the better performance.

The project performance was tested through by running the following particular coding in the application, as shown in Fig 5.1.

```
long startTime, endTime;
startTime = System.currentTimeMillis();
//uRLReader();
//...
//...
endTime = System.currentTimeMillis();
System.out.println("parsing response time: "
        +(endTime-startTime)+" milliseconds");}
```

Fig 5.1 the method to test the performance

5.1 Multiple resource listening

The application is able to receive data from multiple resources in both eSTAR servers and MOA servers. It means that the application may receive data by opening two or more windows at the same time. In the real-time listening model, because each of the windows for receiving data is packed in a thread, the system is able to run the threads as many as possible (See Fig 5.2).

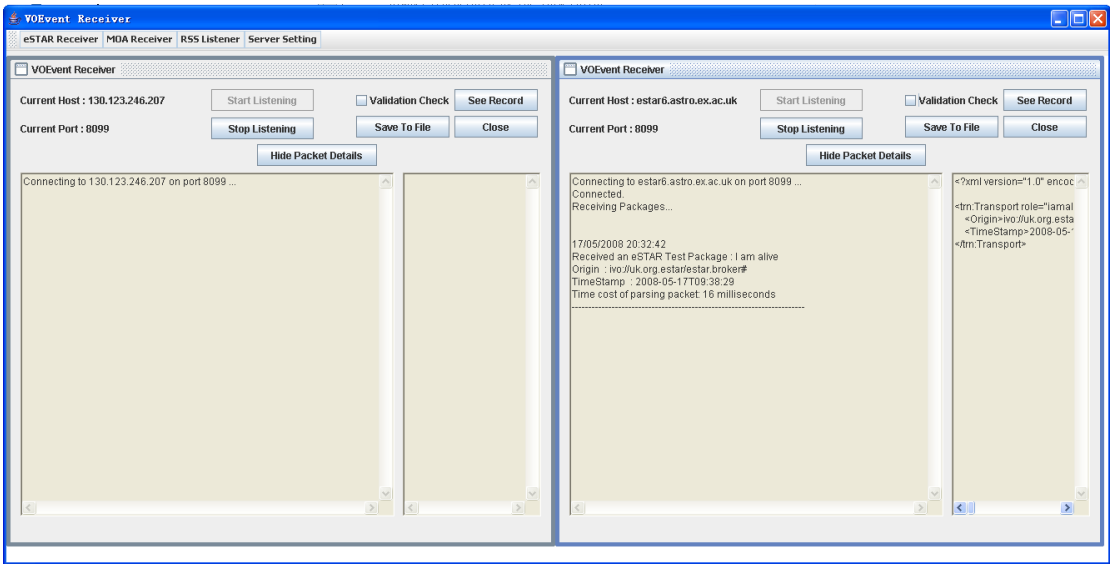

Fig 5.2 listening data from two servers at the same time

5.2 Real-time parsing time cost

To examine the performance of the application, the parsing time cost is the most important part. The following table (Table 5.1) is giving the time costs for three VOEvent packets. The first one is the “iamalive” test packet, and the second one is the example VOEvent packet, named as “example1-v1.0” (VOEvent example, 2008), and the third one is another example VOEvent packet with a bigger file size and more information, “exmaple2-v1.0” (VOEvent example, 2008).

Packet	**Size**	1st	2nd	3rd	4th	Ave
I am alive	462 Byte	16ms	15ms	16ms	17ms	16ms
Example1	787 Byte	31ms	32ms	31ms	32ms	31.5ms
Example2	3170 Byte	47ms	46ms	47ms	48ms	47ms

Table 5.1 time costs for receiving three types of VOEvent packets

The “example1-v1.0” packet contains only three VOEvent XML tags, which are

<Who>, <What> and <Why>. The “exmple2-v1.0” includes six VOEvent XML tags, which comes with more information in it.

From the data in Table 5.1, the real-time parsing time costs depend on the sizes and the content of the VOEvent packets, and also the Internet connecting speed is the countable factor.

5.3 RSS reader parsing time cost

In the RSS reader model, the RSS reader responding time is the time cost from hitting the “read feeds” button to the full information of the feed titles generated. The parsing time for each feed is from hitting the title of the feed to the full parsed information showed (As shown in Fig 5.3)

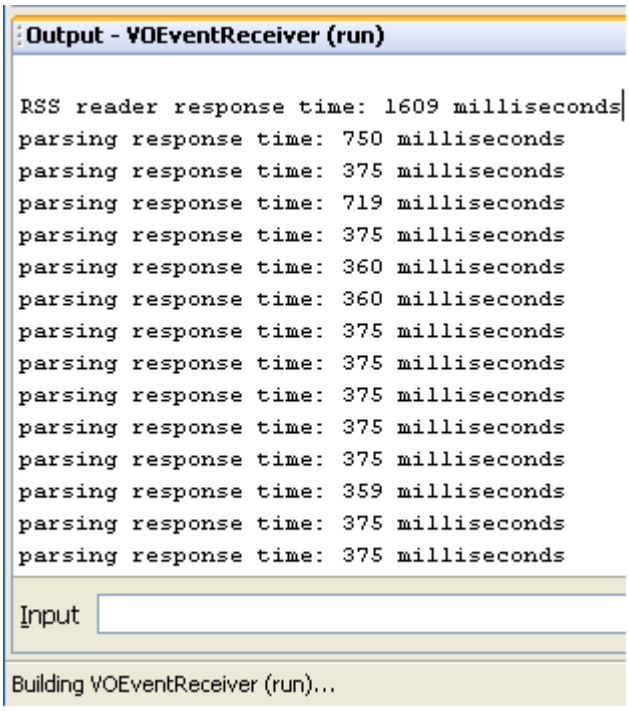

Fig 5.3 the time cost for the RSS model

5.4 Validation check time cost

The Java-based Internet feed listener application applies the validation checks for the received VOEvent packet data. During the check, the application has to retrieve the corresponding schemas from the VOEvent packet (As shown in Fig 5.4).

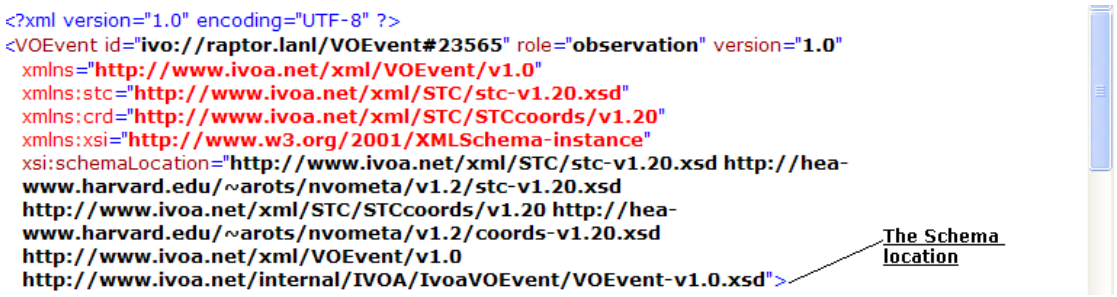

Fig 5.4 the schema location in a VOEvent packet

When the schema in the VOEvent packet is allocated, the application has two ways to process it. The first way can be fulfilled by the "java.net.URL" or "javax.xml.transform.source". The steps are as below.

- Link to the schema location
- Download the schema into the local memory and be ready for the validation check.

Since the application keeps listening the VOEvent information at all time and the VOEvent schema does not updated frequently, the application may store all the VOEvent schemas as the file formats in the root folder of the application. The second way to process the schema is via the "java.io.File" class. The steps for the second way are as following.

- Check the current schema name
- Find the correct schema file in the local folder,
- Load the schema into memory and be ready for the validation check.

In Table 5.2, it shows the time costs for the both ways.

	1st packet	2nd packet	3rd packet	4th packet	Average
URL	10204ms	8516ms	6844ms	7203ms	8191ms
File	3578ms	2719ms	3824ms	3015ms	3284ms

Table 5.2 the time cost for the validation check

To compare these two ways of processing VOEvent schemas, the Java-based Internet listener uses the second way with the file procedure, which is able to give the better performance than the URL way (As shown in Table 5.3).

	Advantages	Disadvantages
URL	Able to check the updated schema in real-time.	Cost more time than the "file" method. Take up more Internet bandwidth due to checking the each packet
File	Cost less time than the "URL" method, more than half processing time saved.	Cannot automatically update the schema

Table 5.3 the comparison about both ways to process VOEvent schemas

5.5 Parsing method designs

During the development of the parsing method, two methods were designed. One is the "ParsePack.java" with the existing "javax.xml" parser. And the other one is the "ParsePackage.java", which is using the own Java "Vector" to process the parsing.

The first method is the one discussed in 3.3.3.4, and the working procedure of the second "Vector" method is shown in Fig 5.5.

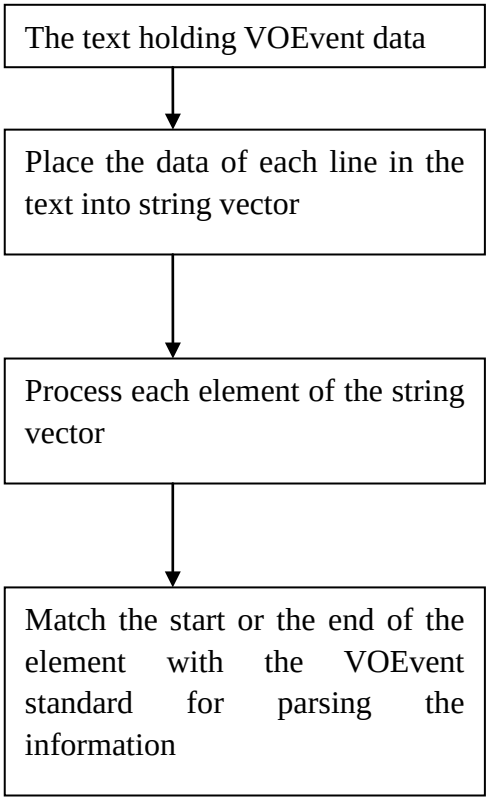

Fig 5.5 an alternative parsing method

Because this “Vector” method cannot be used to parse the VOEvent packet with parameters in the particular tags, for example, sometimes in the <What> tag of the VOEvent packet (As shown in Fig 3.21). There are the information like “<Param name=“RA”…>” and “<Param name=“Dec”…>”, because they are shared with the same starting and ending string, this method is not able to process them. Also because the validation check is difficult to be applied in this method.

As a result, the application chooses the “javax.xml” parser, the developed Java library, which is also easy to be integrated with the validation check functions.

6. Conclusions

This Java-based Internet feed listener is developed with user friendly interface, easy to backup received data, and multiple resources available. It can do real-time detection and gather event information from RSS sources on the Internet. The program could be very helpful for VOEvent users and astronomical fans. Also because it is implemented in Java, it would be more secure and more portable between platforms than the other computer language. Users are even able to trace the astronomical events from the Java supported PDAs and mobile phones.

The application was tested for continuously receiving the VOEvent data from the eSTAR server for over three days. Through the tests, we can see the application is capable to retrieve the data as a VOEvent event broker and take advantage of the benefits of Java.

During the development of the application, a large number of technologies have been studied and practiced. The main technical components of the application are Java application programming, Java Internet programming, XML programming and astronomical VOEvent packet studies.

7. Future work

For now, this application works on listening to the VOEvent packets, parsing them into human readable information, and then sending the information to users. In the future work, because astronomical events may happen in the same time, the application may help a user to make intelligent decisions on what type of events to follow up. For finding out potential rules or connections between events, the application may use Java to analyze the contents of VOEvent packets.

Also because the application was written by Java, it should be compatible with any platform, like in UNIX, the operating system of a mobile phone, etc... For the future work, the application needs to be tested in other different platforms

Recently Google Sky was produced by Google. It is added into the Google Earth software for viewing stars and astronomical images (Google Sky, 2007). And it becomes a popular tool to explore the universe. For the further work of the application on this side, when receiving VOEvent information with the details of the physical location, the application is able to point the location in Google Sky and give a link to the image on Google Sky for users' observation

Finally, we may connect this application to a robotic telescope. Because we've already found out the event location in the parsed information, the application may automatically send a command to make the telescope point to that position for a real-time observation.

References

C.T. Arrington, Syed H. Rayhan (2003), *Enterprise Java with UML*. Wiley Publishing 2-3

eSTAR Event broker, 2006, Retrieved May 14, 2008, from http://www.estar.org.uk/wiki/index.php/ESTAREventBroker

eSTAR server status, 2008, Retrieved May 14, 2008, from http://estar9.astro.ex.ac.uk/estar/cgi-bin/status.cgi

Google Sky, (2007), Retrieved May 14, 2008, from http://www.google.com/sky/about.html

P.K. Yuen &V.Lau (2003), *Practical Web Technologies*, Addison-Wesley Publishing, p80

IVOA, International Vitual Observatory Alliance (2002), Retrieved April 11, 2008, from http://www.ivoa.net/pub/info/

Java RSS Parser, (2004), Retrieved Oct 14, 2007 from http://java.sun.com/developer/technicalArticles/javaserverpages/rss_utilities/

Java Web Start (2006), Retrieved May 10, 2008 from http://java.sun.com/products/javawebstart/

Microsoft Encarta (2007), Retrieved May 11, 2008, from http://encarta.msn.com/encyclopedia_1741502444_1____56/astronomy.html#s56

MOA, Microlensing Observations in Astrophysics, Retrieved April 12, 2008, from http://www.phys.canterbury.ac.nz/moa/index.html

MOA transient alert page (2000), Retrieved May 16, 2008, from https://it019909.massey.ac.nz/moa/alert2000/alert.html

Mereghetti, S., Jennings, D, Pedersen, H., et al. Prospects for rapid gamma-ray burst localization with INTEGRAL. In: Proc. 3rd INTEGRAL Workshop - Astro. Lett. and Communications, 39, 301–304, 1999.

Monique Signore, Denis Puy, (2001). *Supernovae and cosmology,* New Astronomy Reviews.

NASA's HEASARC: Education & Public Information, Supernova (2003). Retrieved Oct 12, 2007, from http://heasarc.gsfc.nasa.gov/docs/snr.html

NASA's Image Galley, Supernova Remant Turns 400 (2004). Retrieved Oct 12, 2007, from http://www.nasa.gov/multimedia/imagegallery/image_feature_219.html

NASA's Imagine the Universe, Gamma-Ray Bursts: Introduction to a Mystery (2007). Retrieved Oct 14, 2007, from http://imagine.gsfc.nasa.gov/docs/science/know_l1/bursts.html

NASA Swift catching Gamma-Ray Bursts on the fly (2006). Retrieved April 11, 2008, from http://swift.gsfc.nasa.gov/docs/swift/news/2006/06-18.html

Nikos Drakos, Computer Based Learning Unit (1993), *Gamma-Ray Burst Physics*, University of Leeds. Retrieved May 10, 2007, from http://www.astro.psu.edu/users/nnp/grbphys.html

O'Reilly, (1996), *Java Network Programming, Elliotte Rusty Harold, 2nd Edition, p301-302*

Rogers Cadenhead, Laura Lemay (2004), *Teach Yourself Java 2*, SAMS, 10-11

Steven Kim, IBM (2001), *Developing and distributing Java applications for the client side*, from http://www.ibm.com/developerworks/java/library/j-webstart/

UML Official Website (2004), Received from http://www.uml.org

VOEvent Citations (2006), Retrieved May 10, 2007, from http://www.ivoa.net/internal/IVOA/IvoaVOEvent/VOEvent-v1.0.html

VOEvent example (2008), Retrieved May 5, 2008, from http://www.ivoa.net/internal/IVOA/IvoaVOEvent/

VOEvent Packet structure (2006), Retrieved May 10, 2007, from http://www.ivoa.net/internal/IVOA/IvoaVOEvent/VOEvent

W3C School (2008), Retrieved May 10, 2007 from http://www.w3schools.com/xml/xml_dtd.asp

Wikipedia, VOEvent, 2006, Retrieved May 10, 2008, from http://en.wikipedia.org/wiki/VOEvent#cite_note-2

Winkler C. INTEGRAL: the current status. In: Proc. 3rd INTEGRAL Workshop - Astro. Lett. and Communications 39, 309–316,1999.