XYZ2 Software Design

Draft 1

Started March 7, 2007

Revised May 5, 2007

For:

ASL USGS

Written by:

Instrumental Software Technologies, Inc.

Offices:

PO Box 4554 Saratoga Springs, NY 12866
Office: (518) 602-0001 FAX: (518) 602-0002

PO Box 963 New Paltz, NY 12561
Office: (845) 256-9290 FAX: (845) 256-9299

http://www.isti.com

info@isti.com

TABLE OF CONTENTS

1. Introduction

The US Geological Survey (USGS), Central Geologic Hazards Team (CGHT) has a requirement to replace the existing graphical user interface (GUI) and associated software supporting seismic waveform data quality control (QC) analysis. Instrumental Software Technologies, Inc. (ISTI) has been awarded a contract to implement the migration from the aging XYZ to modern modular expandable software. The working name of the future software is XYZ2. This name might change in the future.

In this section we discuss specifications of XYZ2. Specifications are derived from the original Statement of Work (version 3) and also based on discussions at the planning meeting in Albuquerque in the winter of 2007 and further email correspondence.

2. Specifications

This section of the document provides a summary of testable specifications for the programming tasks which have to be completed during implementation phase of XYZ2.

2.1. General Specifications

- ◊ Core Functionality of legacy XYZ must be preserved. The most important concepts of XYZ (panels and sequencing) also must be preserved.
- ◊ GUI must be designed to maximize the efficiency of the analyst by minimizing the number of physical actions that are required to perform tasks.
- ◊ The main components of the software shall be open source.
- ◊ The GUI shall run on the following platforms: (a) Solaris; (b) Linux; (c) OS X.
- ◊ GUI shall work on multiple screen types.

2.2. Technical constrains

- ◊ Primary display operation should be no slower than the current implementation;
- ◊ The GUI must be able to display at least 1200 time series segments;
- ◊ The GUI must be able to display at least 30 days of timeseries;
- ◊ The GUI must be able to display at least 16 channels concurrently in each panel;

◊ The GUI must be able to display at least 12-16 panels of data.

2.3. Input Specifications

◊ GUI shall accept the following inputs: (a) Equally sampled timeseries in miniSEED, (b) uncompressed binary format, (c) Unequally sampled timeseries of parameters in XML (autoQC, SOH, picks, arrivals); (d) Response information in RESP or/and SEED files.

2.4. Output Specifications

◊ GUI shall be able to output (a) Selected timeseries in miniSEED (for evenly sampled data); (b) Selected timeseries in XML (for unevenly sampled data); (c) Panel or pop-window image with captions; (d) Updated flow control information via an API**.**

2.5. Startup Specifications

◊ At a startup XYZ2 will read a configuration file (optional) and process command-line options (also optional);
◊ Command-line specification shall override the configuration file options;
◊ Minimum startup configuration will require a location with data files;
◊ The application shall support the use of wildcards in the specification of station/channels/location code/time span;
◊ Default startup parameters are defined in the Statement of Work; Sample configuration file is defined in section 3.2 of this document;
◊ GUI configuration should include references to autoQC XML files and allow the ability to load autoQC reports at startup.

2.6. GUI Layout Specifications

◊ The GUI will generate a root window that typically fills the entire screen;
◊ The root window consists of left or right subwindow (user configurable) for the display of optional autoQC and earthquake information and a top/bottom subwindow (user configurable) for the display of control buttons. The remainder of the root window consists of subwindows (a frame) for the display of the traces;
◊ The output of certain activities or requests for information may be displayed as pop-up windows (spectra, response, correlation function, QC report, MSEED headers);
◊ Pop-up windows for panels may be generated on request.

2.7. Waveform Panel Specifications

- ◊ Each channel (except for overlaid traces) will be displayed in its own panel;
- ◊ The GUI will support the ability to display multiple traces in the same panel in order to overlay traces;
- ◊ The GUI will allow the interrogation of the display to return time and amplitude values based on the selection of one or two points;
- ◊ Trace labels will include SNCL (station-network-channel-location codes), start time, duration, sampling rate;
- ◊ Default behavior of waveform panel will be: (a) independent (absolute) amplitude scaling of each channel and (b) absolute time alignment of the traces;
- ◊ Supported options
    - o Relative amplitude (normalized) scaling between channels (i.e., maximum displayed amplitudes of all channels map to all channel window limits);
    - o Ability to apply an offset to overlaid traces for separation;
    - o Ability to remove the mean of individual timeseries;
    - o Relative time alignment of the traces (alignment starting from the first sample of all channels without regard to absolute time);
    - o Ability to flip polarity of a timeseries (i.e., multiply amplitudes by –1);
    - o Ability to expand/contract the y-scale on an individual panel.
- ◊ Once displayed, the GUI should allow re-selecting subsets of data for redisplay: may be by highlighting existing channel names;
- ◊ The GUI must support the ability to select traces to be overlaid;
- ◊ Using a pointing device, the user will be able to select a section of timeseries to be “zoomed” in a fast and convenient manner;
- ◊ The GUI must support multiple levels of zooming;
- ◊ The GUI must be able to step back through previous zoom levels as well as support returning to the original or full display;

2.8. Waveform Panel Ordering Specifications

- ◊ Default view: All traces for a given station, sorted by sample rate and sensor type. Channels with the same channel name but different location codes shall be displayed within the same panel. For example, at BHZ at 20 sps and a BHZ at 40 sps shall appear in the same panel.
- ◊ Supported Alternatives
    - o All traces for a given station, sorted by sample rate and sensor type: each trace in its own panel;
    - o All traces for a given station displayed in a single panel; multiple stations fill multiple panels;
- ◊ Once displayed, the GUI will allow panels to be ordered (moved up or down in the list);
- ◊ The GUI must support the ability to select traces to be overlaid.

2.9.AutoQC Panel Specifications

- ◊ XYZ2 will read autoQC information from the XML file and present it in a special panel on a frame (either on the left or right side of the waveform frame);
- ◊ The QC panel will display QC "issues", with individual "instances";
- ◊ The GUI will display issues in priority order determined by "criticality";
- ◊ Clicking on an "issue" will cause the GUI will display the affected channels and highlight all instances. Clicking on an "instance" will cause the GUI to zoom in to display the time period of interest;
- ◊ The GUI will allow trace control so that the display of the issues and instances will either take place on all traces specified in the autoQC report or only on the currently displayed traces;

2.10. Event Panel Specifications

- ◊ XYZ2 will read Event information from the XML file and present it in a special panel on a frame (either on the left or right side of the waveform frame);
- ◊ The event pane will contain a list of "known" events.
- ◊ Events for which synthetics exist will be identified through some means such as color, bold type, etc.;
- ◊ Channel-specific event information (as provided in the XML file), such as analyst picks and predicted arrival times shall be displayed on the specified traces.
- ◊ Clicking on "known events" and the GUI will display the affected channels and highlight all events; Clicking on an event and the GUI will zoom in to display the time period of interest (in this case, perhaps starting at the origin time and extending some fixed interval of time); The GUI will allow trace control so that the display of the events will either take place on all traces specified in the event information or only on the loaded traces

2.11. Transformation Specifications

- ◊ XYZ2 shall support the following transformations: (a) Fourier transform; (b) Filtering; (c) Rotation; (d) Particle Motion; (e) Cross- and Auto-correlation

2.12. Closeout Specifications

- ◊ XYZ2 will update flow control at closeout;
- ◊ XYZ2 will provide a user an option to save all unsaved data in pop-up windows.

3. Design Documentation

3.1. Overview

In this section of the document we present the design for the XYZ2 program. ISTI proposes to use Java programming language to develop the XYZ2 tool. The discussion about testing Java language for this project is provided in Section 4.4. Minimum version of Java supported by this project will be 1.5. Java provides a platform independent execution mechanism using the Java Virtual Machine, which runs on all of the specified operating systems. ISTI has a large toolkit of modules that we can provide to speed the development of the tool. The ISTI Util toolkit includes configuration and command line utilities, logging support, and other often used widgets and libraries that ISTI has found useful over the last 8 years.

Java is a modern object oriented and multithreaded language that allows for modular development. All tasks within the modules will be written as discrete Java classes. The modules can be tightly or loosely coupled depending on the use, but will involve separate archive files ("jars" in Java jargon) and packages within the jars that perform specific tasks and functions on the data.

ISTI proposes to develop this software as a series of modules; for both processor engines and for the graphical user interface (GUI) front-end. The GUI front-end will be coupled to the processor engines, such that processing results can be achieved in a multithreaded manner when necessary.

Data will be manipulated based upon the standard NSCL (Network/Station/Channel/Location) organization model, allows for different data streams to be grouped arbitrarily and processed according to these user definable groupings.

With the data manipulation structure defined, the data objects may be populated from any data source. So processing will be independent of access method and data format.

At the moment of writing of this design document many details of the formats and protocols are not known to us and will be provided by USGS staff at a later time. Therefore, we see our minimal task as to design and implement the XYZ2 as an expandable, modular and well-documented clone of the old XYZ. Well-defined new features will be also included in the first version of XYZ2. Our second task is to develop XYZ2 in a way that further additions or modifications of the program will be done with minimal modification of the source code. Fortunately, the concepts of object-oriented programming (class inheritance, objects, and interfaces) provide means for fast and straightforward modifications of the scope and functionality of the program whenever the protocols and formats for new objects will be available.

For example, In Section 3.4.3 we introduce IEvent interface which will be a common interface for various types of objects (SOH, Picks, Arrivals, etc) and in Section 3.4.4 we introduce AbstractEvent class which is a superclass for all common types of XYZ2 extension classes. These classes will provide common behavior to more concrete classes that can be added at a later time.

Since implementation and design are not fully decoupled, we anticipate that the finer points of this design will be changed and completed as the implementation proceeds. We also anticipate that development of XYZ2 will be done in close cooperation with representatives from ASL USGS and our regular partial releases will be evaluated and commented by USGS staff so we could detect and fix problems at early

stages.

XYZ2 program will consist of four independent Java packages:

- ◊ com.isti.xyz.common (described in Section 3.4) – is a library containing common objects which are used by other packages of XYZ2;
- ◊ com.isti.xyz.data (described in Section 3.5) – contains objects related to data storage;
- ◊ com.isti.xyz.gui (described in Section 3.6) contains classes related to XYZ2 graphical user interface;
- ◊ com.isti.xyz.processing (described in Section 3.7) defines classes dealing with various data transformations (such as, rotation and particle motion) and data products (such as spectra and PSD).

Further extensions of the program are designed via Plugin Module which is described in Section 3.8

Design of XYZ2 configuration mechanism is discussed in Section 3.2.

High-level interaction between packages is presented in Section 3.3.

3.1.1. Pattern-Oriented Software Architecture and Design

Most of the design of XYZ2 is based on the conceptual approach of Pattern-Oriented Software Architecture. This approach is discussed in detail elsewhere[1]. Use of Pattern-Oriented Software terminology simplifies and shortens design documentation. Here we briefly describe the approach and few patterns that are used in the design of XYZ2.

Design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Algorithms are not thought of as design patterns, since they solve computational problems rather than design problems.

Not all software patterns are design patterns. Design patterns deal specifically with problems at the level of software design. Other kinds of patterns, such as architectural patterns, describe problems and solutions that have alternative scopes.

Architectural patterns are software patterns that offer well-established solutions to architectural problems in software engineering. An architectural pattern expresses a fundamental structural organization schema for a software system. The schema consists of predefined subsystems and specifies their responsibilities and relations.

- [1] Buchman F., Meunier R., Rohnert H. & Sommerlad P. & Stall M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*.

A pattern system provides, on one level, a pool of proven solutions to many recurring design problems. On another it shows how to combine individual patterns into heterogeneous structures and as such it can be used to facilitate a constructive development of software systems.

In comparison to design patterns, architectural patterns are larger in scope.

Throughout this document we use several well-known design patterns, which we adopt for the XYZ2 design and implementation:

- Observer design pattern (sometimes known as publish/subscribe) is a design pattern to observe the state of an object. The essence of this pattern is that one or more objects (called observers or listeners) are registered (or register themselves) to observe an event that may be raised by the observed object (the subject).
- Wrapper (decorator) design pattern works by wrapping the new "decorator" object around the original object, which is typically achieved by passing the original object as a parameter to the constructor of the decorator, with the decorator implementing the new functionality. The interface of the original object needs to be maintained by the decorator.
- Model–View–Controller (MVC) design pattern - In complex computer applications that present lots of data to the user, one often wishes to separate data (model) and user interface (view) concerns, so that changes to the user interface do not affect the data handling, and that the data can be reorganized without changing the user interface. The model-view-controller solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller.
- Strategy design Pattern is a particular software design pattern, whereby algorithms can be selected on-the-fly at runtime. The Strategy pattern is useful for situations where it is necessary to dynamically swap the algorithms used in an application. The strategy pattern is intended to provide a means to define a family of algorithms, encapsulate each one as an object, and make them interchangeable.
- State design pattern – is a fully encapsulated, self-modifying Strategy Design Pattern.
- Singleton design pattern – restricts instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.

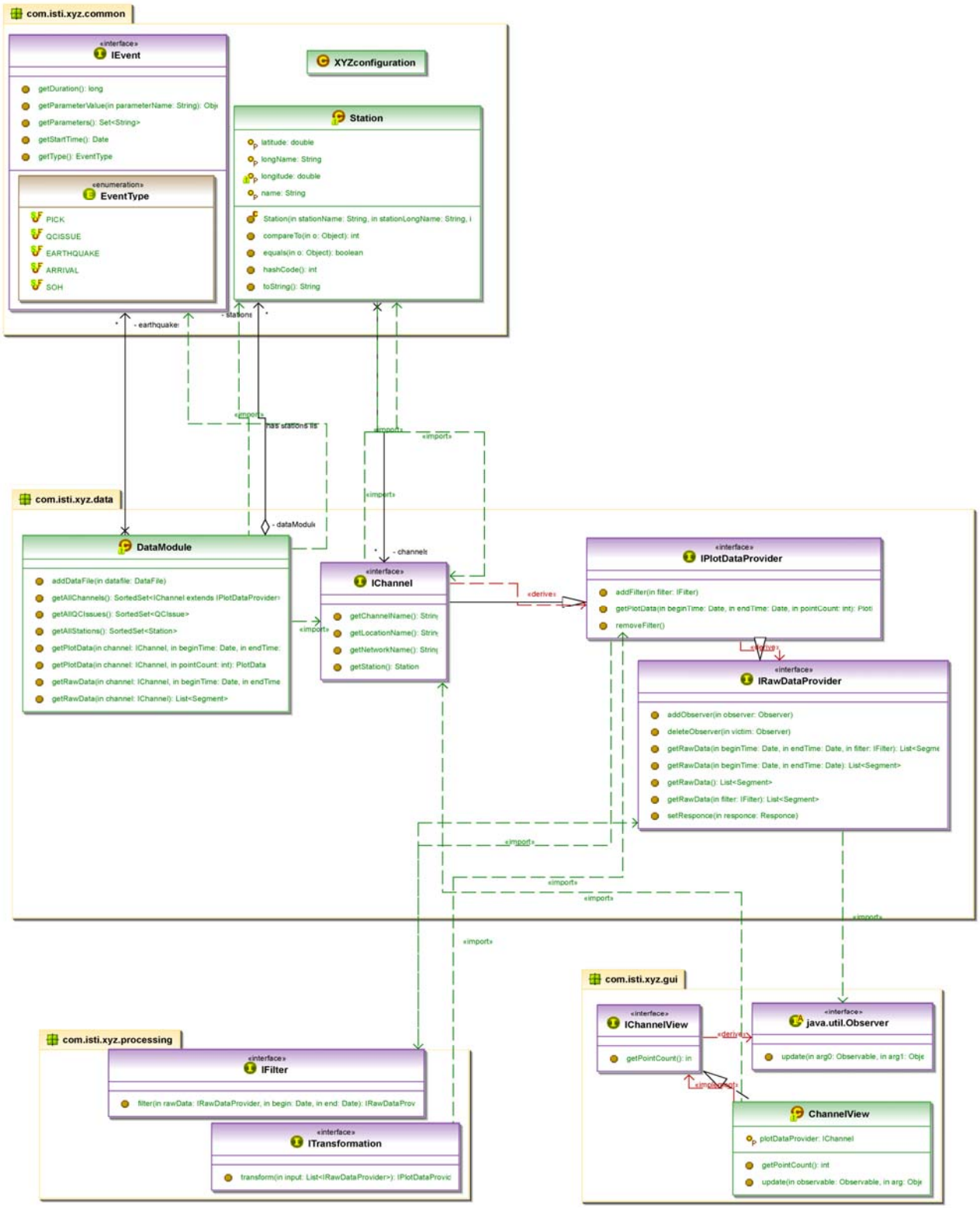
Some additional information about design patterns is presented in the corresponding sections of our document.

3.2.XYZ2 configuration and start-up options

At startup XYZ2 will read a configuration file (optional) and process command-line options (also optional). Command-line specification shall override the configuration file options. Finally, default parameters are used for those variables which are not defined elsewhere.
XYZ configuration file will be in XML format. Here is a template:

```
<XYZ2SessionConfiguration>
<XYZ2Version>V1.0</XYZ2Version>
<Configuration>
<StartTime>2007-01-03T07:00:00</StartTime>
<EndTime>2007-01-03T08:00:00</EndTime>
<Path>/home/data</Path>
<Station selectedFlag="true">
<Name>B004</Name>
<Channel selectedFlag="true">
<Name>LS1</Name>
<DataStartTime>2007-01-03T07:00:00</DataStartTime>
<Duration>4</Duration>
</Channel>
<Channel selectedFlag="true">
<Name>LS2</Name>
<DataStartTime>2007-01-03T07:00:00</DataStartTime>
<Duration>4</Duration>
</Channel>
</Station>
<Station selectedFlag="true">
<Name>B005</Name>
<Channel selectedFlag="true">
<Name>LS1</Name>
<DataStartTime>2007-01-03T07:00:00</DataStartTime>
<Duration>10</Duration>
</Channel>
</Station>
<Station selectedFlag="true">
<Name>B005</Name>
<Channel selectedFlag="true">
<Name>LS1</Name>
<DataStartTime>2007-01-03T07:00:00</DataStartTime>
<Duration>60</Duration>
</Channel>
</Station>
<QCdataPath>/home/QC</QCdataPath>
<EventdataPath>/home/Events</EventdataPath>
<StationInfoPath>/home/Events</StationInfoPath>
< PanelOrder>1</PanelOrder>
<TracesInFrame>16</TracesInFrame>
</Configuration>
<SessionData>
</SessionData>
</XYZ2SessionConfiguration>
```

3.3. High-level diagram: interaction between packages



Classes are building blocks of a Java program. Classes are grouped into the physical archives, which are named packages. As it is presented in the diagram, there are four packages in XYZ2: COMMON, GUI, DATA, and PROCESSING.

Not all classes are equally important to the architecture of the software system: some of them are used only inside a single package; others represent the key components which are used by almost every module of the system. The diagram in this section shows only the most important classes of XYZ2 design.

Let us use the diagram above and discuss how XYZ2 will work.

At startup the program searches for the configuration file and initializes XYZConfiguration class. Next, command line parameters (if any) are used to overwrite configuration file parameters and provide additional configuration values. XYZConfiguration is updated. Finally, default parameters are used for those variables that are not defined elsewhere.

Next, XYZ2 uses a list of stations/channels from the configuration class and scans data files in order to initialize Station and IChannel classes. An order of panel rendering is being determined. Next, the program loads additional metadata (picks, QC, earthquake information), which initialize IEvent interface. Metadata are linked with the station channels.

At this moment XYZ2 starts Graphical User Interface (GUI). GUI contains a set of instances of ChannelView class which are responsible for plotting in XYZ2 frame. ChannelView class registers with the corresponding channel via the Observer interface and informs the recipient that it is interested in receiving events related to data modifications. The GUI uses methods from DataModule class in order to receive lists of seismic events, QC issuances and so on for further plotting in XYZ2 frame.

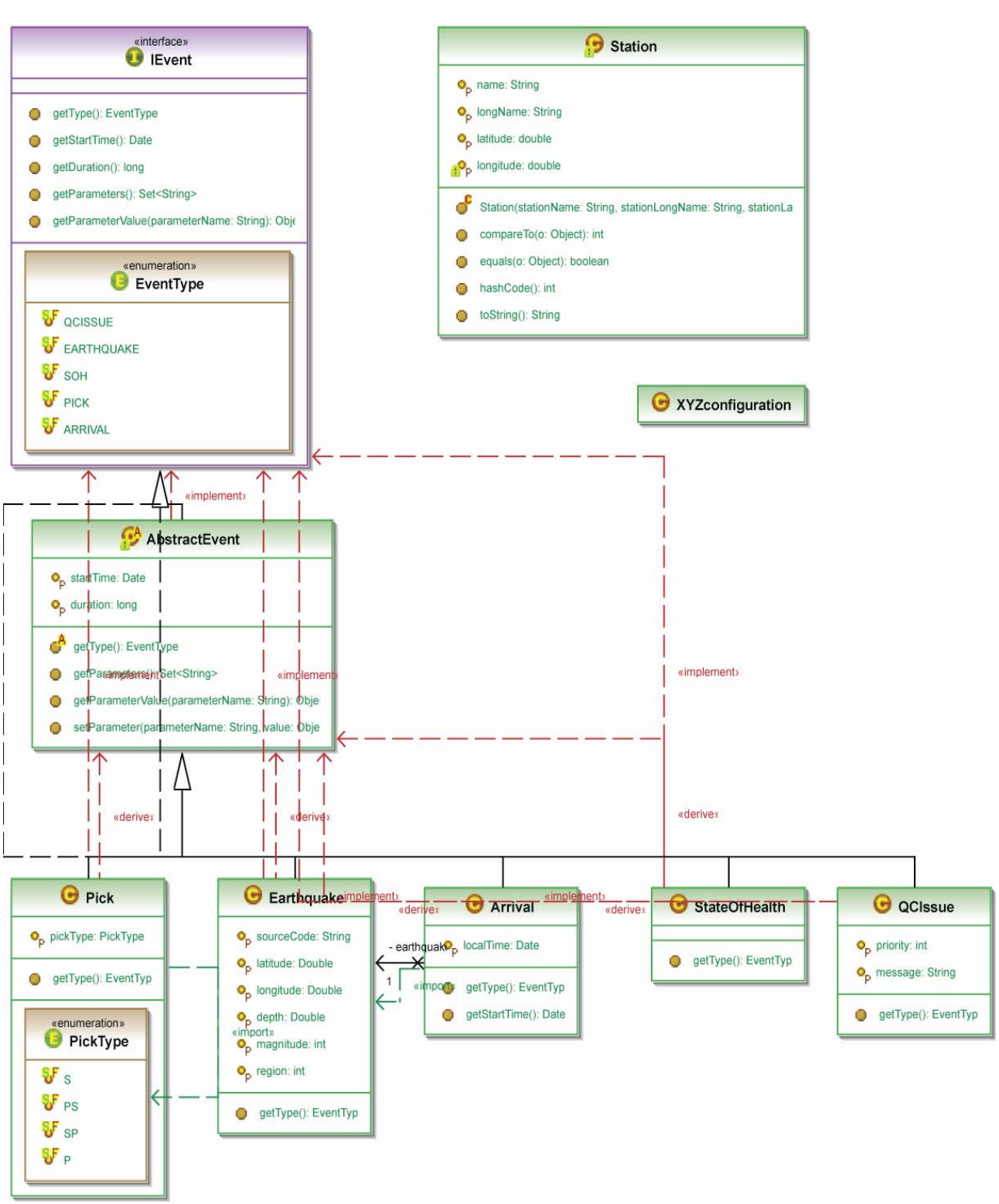
Simultaneously, XYZ2 starts a separate thread that is responsible for waveform loading and internal storage. An interface IRawDataProvider is responsible for data loading. We load that data first which will be displayed in the first frame. As soon as we load enough data to fill the first panel of XYZ2 screen, IRawDataProvider informs ChannelView about this and ChannelView begins rendering data on the screen.

In a meantime, IRawDataProvider thread continues loading additional seismic data in the background. Such an approach should speedup the overall performance of XYZ2.

In order to plot in the panel ChannelView calls method getPlotData() from the interface IPlotDataProvider, which is inherited from IRawDataProvider. The functionality of getPlotData() is to convert original data to the format suitable for fast rendering on the screen.

Various transformations of the data which can be requested by a user are designed as a set of plug-in modules inherited from one of two interfaces: IFilter or ITransformation. Filter gets raw data segments from IrawDataProvider and outputs modified seismic trace segments. Transformation creates a new data product (PSD, Spectra) and passes the output for the display or storage.

3.4. Basic classes and interfaces



3.4.1. Configuration class

This class holds configuration data. It reads configuration file and uses default values for missed parameters. It overwrites configuration parameters with command-line parameters.

Properties:

- String dataPath – path to directory where datafiles live;
- Date startTime – global start time;
- Date endTime – global end time;
- List<StationConf> stations – stations list, see StationConf definition below;
- String QCdataFileName – full pathname for quality control data file;
- String EarthquakeFileName` – full pathname for earthquakes definition file;
- String StationInfoFileName – full pathname for stations definition file;
- int tracesInFrame – quantity of traces shown on the screen;
- int panelOrder – Order to sort traces to show, see Appendix 4.2 for options list.

Methods:

- List<String> getStationNames() – returns a list of configured stations names;
- List<String> getChannelNames(String stationName) – returns a list of configured channels for a particular station;
- List<String> getSelectedChannelNames(String stationName)) – returns a list of selected configured channels for a particular station
- Date getChannelStart(String channelName) – return begin time for give channel;
- Date getChannelEnd(String channelName) – return end time for given channel.

Class is designed as a Singleton pattern.

Configuration class has internal classes StationConf and ChannelConf, which hold information about particular station and channel configuration, correspondingly.

ChannelConf properties:

- String name – channel name;
- Boolean isSelected – selection mark;
- Date startTime – channel time section start;
- Date endTime – channel time section end;

StationConf properties:

- String name – station name;
- List<ChannelConf> channels – channels list;

3.4.2. Station class

This class holds stations information. Station list and channels are initialized during startup: the program scans data files and builds a list of station/channel pairs.
Such information as longitude, latitude, elevation and depth is loaded from a station configuration file.
Class implements an interface "Comparable"[2] to define default sort order in the station sets.

Properties:

- String Name – station name;
- String LongName – long station name;
- double Latitude – latitude;
- double Longitude – longitude;
- double Elevation – elevation;
- double depth – depth;
- Set<IChannel> Channels – set of station channels.

Methods:

- static Set<Station> initStations(File file, Set<Station> stations) – initializes station set from file. If stations parameter is null the class loads all stations, otherwise it only fills provided stations set with information from file.
- void addChannel(IChannel channel) – adds channel to station

3.4.3. IEvent interface

This is a generic interface to represent an event inside the program. This interface extends "Comparable" in order to define panels sorting order. Event here is as an abstract object that is defined by a start time, duration, and series of other parameters and needs to be displayed in the channel panels. Note that we use the word Event in a software sense, not seismological.

IEvent interface has enumeration EventType to hold types of event, currently PICK, EARTHQUAKE, ARRIVAL, SOH, QCISSUE.

Methods:

- EventType getType() – returns type of an event;
- Date getStartTime() – returns event start time;
- long getDuration() – returns event duration, in ms.

Since different in nature events implement IEvent interface, we need a common way to retrieve specific parameters for each type of event:

[2] Standard Java interface defines means to compare two objects

- Set<String> getParameters() – returns a set of attached parameters names for each type of event;
- Object getParameterValue(String parameterName) – returns a value of a parameter.

Also this gives us a way to seamlessly add additional types of events in the future.

3.4.4. AbstractEvent class

AbstractEvent class implements IEvent interface and contains all common behaviors of events. All concrete events should extend this class.

Properties:
- Date startTime – time of event;
- Long duration – event duration, in ms;
- Map<String, Object> parameters map, consists of pairs “parameter name – parameter value”.

Methods:
- Inherited from IEvent;
- Inherited from Comparable.

3.4.5. Pick class

This class deals with times of picks attached to a channel, manually entered by an operator or read from XML file. Enumeration representing pick type are : P, S, PS, SP (needs to be continued).

Properties:
- pickType – holds pick type;
- IChannel channel – name of the channel associated with a pick.

Methods:
- static Set<Pick> getPicks(File file) – loads picks set from file;
- void save(File file) – add pick description to a file.

3.4.6. State-of-Health class

The class deals with the State of Health (SOH) information for a channel. Functionality will be provided after SOH XML file structure is defined.

3.4.7. Earthquake class

The class holds data about a particular earthquake, loaded from XML file.
Properties:
- String sourceCode Earthquake codename;
- double latitude Earthquake latitude;
- double longitude Earthquake longitude;

- double depth Depth of earthquake;
- int magnitude Magnitude;
- int region Region code.

Methods:

- static Set<Earthquake> getEarthquakes(File file) – loads earthquakes set from the file.

3.4.8. Arrival class

This is a wrapper for Earthquake class, it represents an earthquake arrival time to a given instrument. Collection of Arrival(s) is a property of a channel and can be found in the events list.

Properties:

- Earthquake earthquake – reference to Earthquake class;
- Date Time – time of arrival at a channel.

Methods:

- Date getTime() – overrides Earthquake's getTime(), returns arrival's local time instead of Earthquake's time.

3.4.9. QCissue class

This class holds information about Automated Quality Control issue, loaded from XML files.

Properties:

- String message – QC text message describing the essence of issue
- int priority – Priority of this issue, calculated on the base of criticality algorithm supplied by USGS;
- Set<IChannel> channels – Collection of affected channels.

Methods

- static Set<QCissue> getQCissues(File file) – loads QC issues set from file

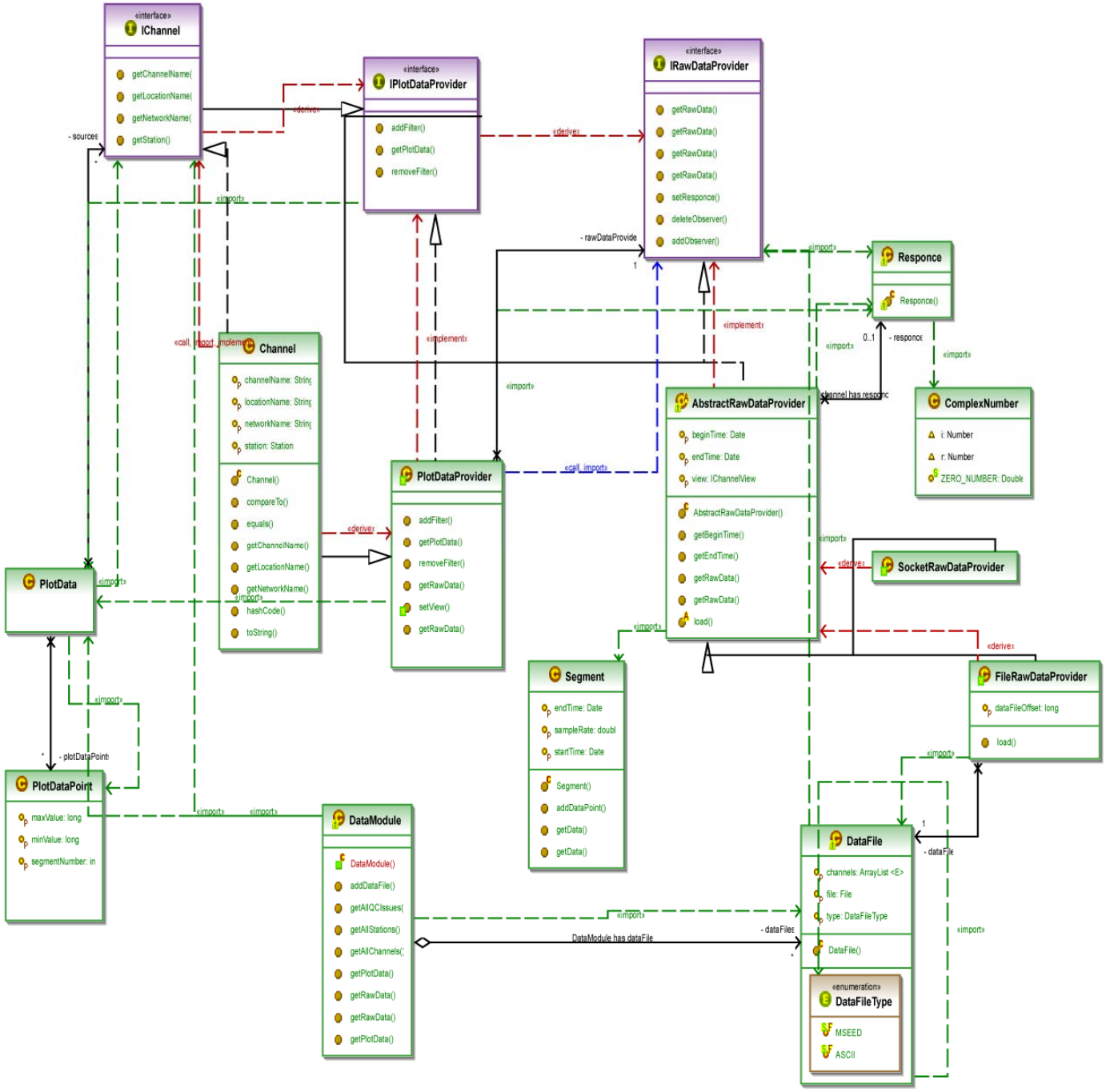
Full description of QCissue will be provided when QC XML file is defined.

3.5. Data storage/handling

Data module consists of three main sections: DataModule class (see 3.5.1), which holds a collections of objects from loaded dataset, descendents of IRawDataProvider interface (see 3.5.3) to hold and manipulate raw trace data, and descendents of IPlotDataProvider interface (see 3.5.6) to create and manipulate pixelized view of data prepared for rendering.

Data module design allows seamless integration of additional heterogeneous data sources in the future.

A diagram of the Data Module is presented below:



3.5.1. DataModule class

This class holds collections of events, stations, channels, datafiles loaded from configured data sources. It provides a simplified interface to access data module functions. The Class is designed as a Singleton.

Methods:

- SortedSet<Station> getStations() – Returns a collection of stations;
- SortedSet<IChannel> getChannels() – Returns a collection of channels;
- SortedSet<QCissue> getQCIssues(Set<IChannel> channels) – Returns a collection of QC issues loaded for a given channels to fill QC issues panel. If channel list is empty, it returns all known QC issues;
- List <DataFile> getDataFiles() – Returns currently loaded set of datafiles;
- SortedSet<Earthquake>getEarthquakes() – Returns all loaded earthquakes;

3.5.2. Segment class

This class holds raw data for a trace segment. A segment is a waveform trace without gaps. A Seismic trace therefore is defined as a sorted list of segments for a given channel.

Properties:

- List<Long> data – holds an array of sensor info;
- double sampleRate – sample rate of a digitizer;
- startTime – start time of a segment.

Methods:

- Date getEndTime() – returns a segment end time;
- List<Long> getData(Date begin, Date end, IFilter filter) – returns a segment subset of segment data with applied filter;
- List<Long> getData(IFilter filter) – returns all segment data with applied filter;
- void addDataPoint(Long value) – adds data to the end of list.

3.5.3. IRawDataProvider interface

This interface represents a seismic trace and introduces an abstract way to get loaded raw data.

Methods:

- List getRawData(Date startTime, Date endTime, IFilter filter) – gets filtered trace data ordered by time interval from startTime to endTime;
- List getRawData(Date startTime, Date endTime) – get trace data ordered by time interval from startTime to endTime without filtering filter;
- List getRawData(IFilter filter) – get all loaded trace data, with applied filter;
- List getRawData() – get all loaded original trace data;
- void setView(IChannelView view) – sets view for this trace according to MVC pattern (see GUI section) to notify about data changes;
- void setResponse(Response response) – sets response for this data used during data loading;
- void addObserver(Observer observer) – adds observer, i.e., object which tracks changes of the other objects. More details are provided in AbstractRawDataProvider description;

- public void deleteObserver(Observer victim) – delete observer from observers list.

3.5.4. AbstractRawDataProvider class

This class implements IRawDataProvider interface. This is an abstract class which holds all Raw Data providers shared behavior. Concrete RawData providers shall extend this class.

This class contains an internal private class SegmentCache which holds both original Segments and filtered Segments. The class also manages cache policy. It extends Observable and realizes the Observer design pattern. The Observer pattern (sometimes known as publish/subscribe) is a design pattern used to observe the state of an object in a program. It is related to the principle of implicit invocation. The essence of this pattern is that one or more objects (called observers or listeners) are registered (or register themselves) to observe an event which may be raised by the observed object (the subject). The object, which may raise an event generally, maintains a collection of the observers.

In XYZ2 we use an internal Java realization of a pattern; all objects interested to be notified will implement Observer interface and register themselves via addObserver() method from this class. This feature is intended for use in GUI which will render this data.

Properties:

- Date startTime – trace start time;
- Date endTime – trace end time;
- List<SegmentCache> RawData (list of segments);
- Response response – response of this trace used during data loading;
- IChannelView view – view for this trace to notify it about data changes.

Methods:

- Inherited from IrawDataProvider;
- abstract void load() – Method specific for every subclass of AbstractRawDataProvider to load data from a data source.

3.5.5. FileRawDataProvider class

This class extends AbstractRawDataProvider. This is a concrete class which will be used to load data from the file.

Properties:

- DataFile dataFile – reference to a DataFile object which represents data file on the disk;
- private long dataFileOffset – offset in a disk file to random access to this trace.

Methods:

- public void load() – loads data from the disk file. Concrete realization load() from AbstractRawDataProvider.

Others, more specific realizations of AbstractRawDataProvider, for example: MSeedRawDataProvider, SeedRawDataProvider, SocketRawDataProvider etc. will extend this class.

3.5.6. IPlotDataProvider interface

This class extends IRawDataProvider, so all instances implementing IPlotDataProvider still can provide raw data if needed. The class adds a method to get a pixelized view of the trace data. The concept of a pixelized view is a realization of a simple idea that the computer screen can fit up to a couple of thousand horizontal data points (pixels) at most. So implementing a class of pixelized data (in other word, store pixelized cache and manage pixalization policy) we can significantly speed-up plotting capability of the program.

Methods:

- public PlotData getPlotData(Date startTime, Date endTime, int pointCount) – returns PlotData in time window from startTime to endTime, with filtering out redundant points;
- public void addFilter(IFilter filter) – add a filter to the trace;
- public void removeFilter() – remove a filter from the trace.

3.5.7. PlotData class

This class represents pixelized data which are prepared (filtered) for rendering.

Properties:

- List<PlotDataPoint> plotDataPoints – a collection of PlotDataPoints. Visualizer requests not just raw waveform data, but actual pixels by calling IPlotDataProvider.getPlotData; the output of the method contains requested pixels.
- List<IChannel> sources – a collection of traces, used as sources to get PlotData. In most common case it is a single trace, but it can be modified to operate on several traces and return a single PlotData object.

3.5.8. PlotDataPoint class

Data needed by Visualizer to plot a pixel of a graph. Data point in this case is a vertical line from minimal to maximal value in a time window.

Properties:

- long minValue – minimal value on pixel time window;
- long maxValue – maximal value on pixel time window;
- IEvent event – if some event takes place in the pixel time window, a Visualizer can find a reference to it here, otherwise the value is null;
- int segmentNumber – segment number.

3.5.9. PlotDataProvider class

This class implements IPlotDataProvider. It holds trace pixelized data, calls Pixelizer for a requested trace and manages pixelized data cache. It is implemented as a Wrapper design pattern, i.e., it does not extend RawDataProvider, but rather contains IRawDataProvider member and transfers IRawDataProvider methods calls to it. That is why we can use the same PlotDataProvider interface with different RawDataProviders.

Properties:

- IRawDataProvider rawDataProvider – wrapped member;
- SortedSet<IEvent> – list of events known for a given seismic trace, sorted in event time order. Used to prepare PlotData.

Methods:

- private void pixelize(Date begin, Date end, int pointCount) – populates internal data structures with pixelized data;
- Inherited from IplotDataProvider;
- Inherited from IrawDataProvider.

3.5.10. IChannel interface

This interface extends IPlotDataProvider. It represents a channel, i.e., channel identification information and trace data.

Properties:

- String getChannelName() – channel name;
- String getLocationName() – location name;
- String getNetworkName() – network name;
- Station getStation() – reference to a station;

3.5.11. Channel class

This class extends PlotDataProvider and implements IChannel and Comparable interfaces.

Properties:

- String channelName;
- Stationstation;
- String locationName;
- String networkName;

Methods:

- Inherited from Comparable;
- int compareTo(Object o) – Compares this object with another object. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object;

- Boolean equals(Object o) – Indicates whether some channel is equal to this one.

3.6.GUI module

GUI module is designed as a Model–View–Controller (MVC) design pattern. It separates data from its visual representation so that changes to the user interface do not affect data handling, and data portion of the pattern can be reorganized without changing of the user interface.
As it follows from the name, this pattern consists of three parts:

- Model: the domain-specific representation of the information on which the application operates. In our case DataModule provides all required methods to access data.
- View: renders the model into a form suitable for interaction, in this system SWING GUI.
- Controller: processes and responds to events, typically user actions, may invoke changes on the model.

3.6.1. XYZframe class

This class extends JFrame (a standard Java class).
Main application window: holds Swing GUI widgets[3] and information about current application state.

We plan to use the State design pattern to hold the current state of GUI. This is a clean way for an object to partially change its behavior at runtime. For example, we have several different scale modes in our GUI representation. We define a property scaleMode of IScaleMode type. IScaleMode is an interface defining methods to get boundaries for our graph. We develop different classes implementing IScaleMode and representing concrete states: ScaleModeAuto, ScaleModCom and ScaleModXhair and assign one of them to a scaleMode property. When application needs to know the start and end of X-axis section, it calls the corresponding methods of a scaleMode member.

Such an approach allows us to add other states without changing existing ones and avoid huge if-else sections, spread functionality to avoid too complex and long classes and decrease program complexity – each concrete state keeps all conditions and code required to transfer to another state.

Properties:

- IScaleModeState scaleMode – holds current scale mode of main application window graphs. Available choices are ScaleModeAuto, ScaleModCom and ScaleModXhair;
- IPolarityState polarity state – define polarity state;
- IColorModeState colorMode – toggles color on/off. Available choices are ColorModeEnabled and ColorModeDisabled;
- IOverlayState overlayMode – toggles overlay mode on/off. In overlay mode we draw several traces on a single graph simultaneously. Available choices are OverlayEnabled and OverlayDisabled;
- IShiftState shiftState – toggles shift mode. Available choices are MeanMode - shifts traces on its mean value, OffsetMode – in overlay state shifts traces to clear visibility, and RegularMode, without

[3] Standard Java GUI library.

any shift;

- ITimeAlignmentState – toggles traces time alignment state, TimeAlignmentAbsolute or TimeAlignmentRelative;
- IPhaseState phaseState toggles phase drawing on a graphs. Available choices are PhaseModeEnabled and PhaseModeDisabled;
- IPickState pickState – toggles pick mode: add picks, delete picks, disabled;
- int xMax – Max value on X axis of graphs in the main frame;
- long xMin – Min value on X axis of graphs in the main frame;
- Date timeStart – Time window start in the main frame;
- Date timeEnd – Time window end in the main frame;
- int channelsShowCount – number of channels in the main frame;

3.6.2. GraphPanel class

This class extends JPanel. This is Graphics container; it contains a list of ChannelView(s) (panels) and renders them as a 1-column table; responsible for ChannelView selecting and ordering.

3.6.3. ChannelView class

It also extends JPanel and implements IChannelView.

Panel contains plotting area; the class can render one or several channels on a single panel.

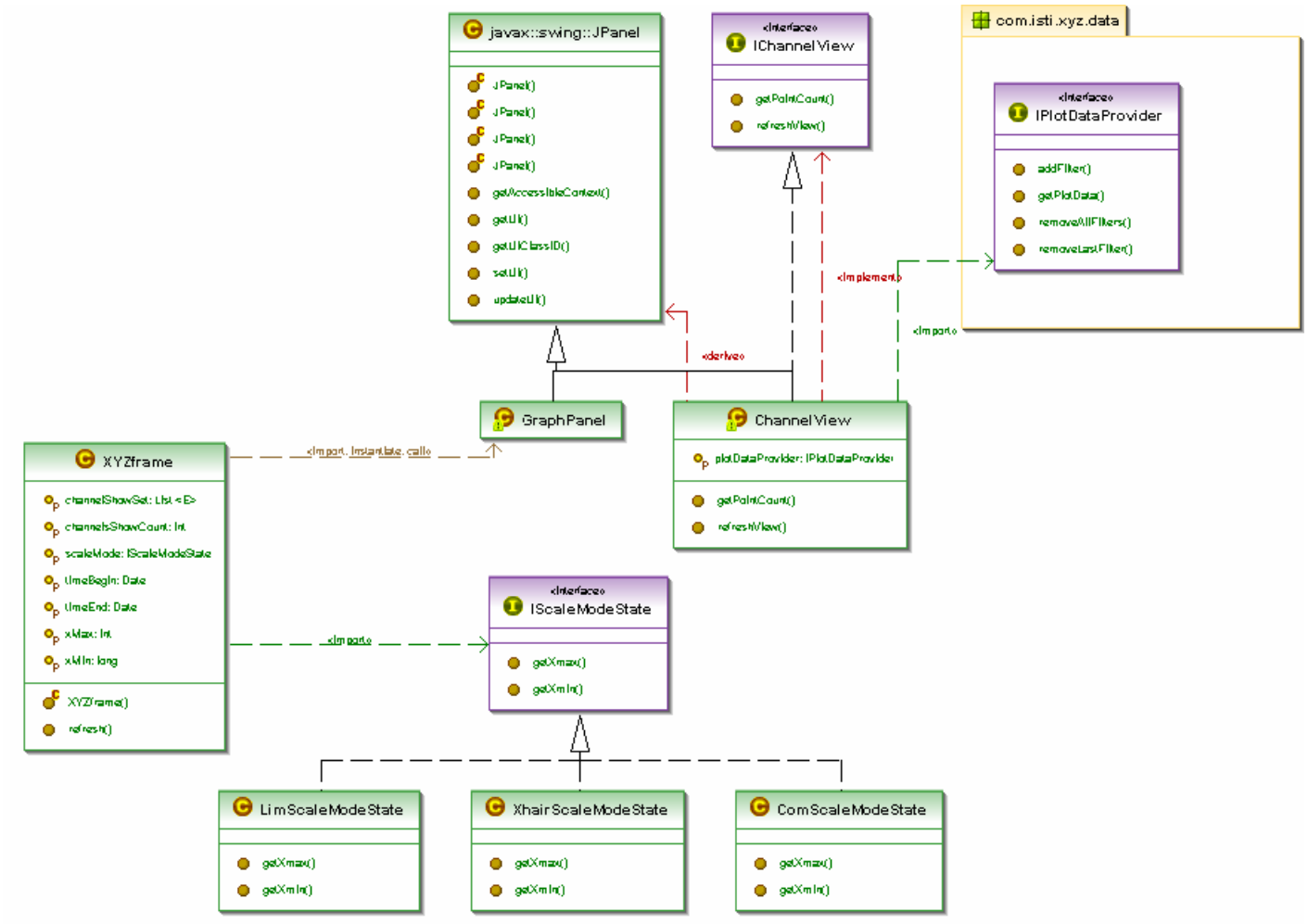
Properties:

- List<IChannel> plotDataProviders – list of channels to render in this panel;

Methods:

- int getPointCount() – returns drawing area width to request data to plot;
- void update(Observable observable, Object arg) – inherited from Observable, redraws an area with current data.

Current UML view of GUI module:



3.7. MVC Pattern's controller

User actions from GUI generate special command objects. A command object encapsulates an action and its parameters. The idea of encapsulation of a command as an object is based on the following reasons:

* A command object can be easily stored and retrieved.

* Treating commands as objects enables data structures containing multiple commands. A complex process could be treated as a set of command objects.

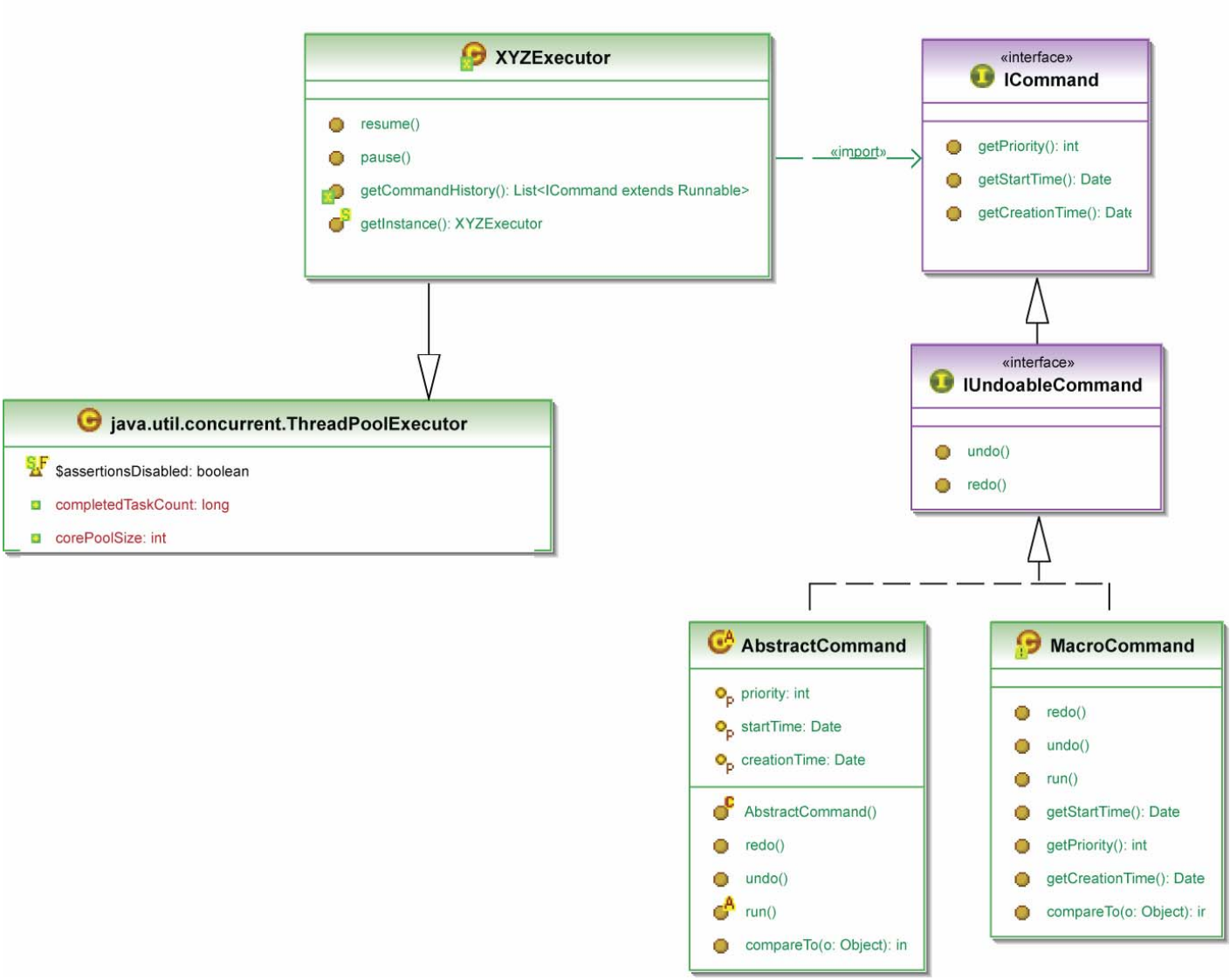
* We can easily maintain command history log and undo-redo operations.

* The command is a useful abstraction for building generic components, such as a thread pool, that can handle command objects of any type. If a new type of a command object is created later, it can work with these generic components automatically.

*A thread pool could maintain a priority queue of command objects consumed by worker threads.

The Controller itself is a queue of commands. At the initialization stage the program creates a pool of empty threads. When a user requests a particular command, GUI creates a corresponding command object and adds it to a queue. A command processor keeps a pool of threads and assigns current commands to it for

execution. A pre-assigned thread pool allows us to execute commands efficiently and permits a quick return to calling GUI thread.



3.7.1. ICommand interface

This interface introduces a generic command.

It extends Runnable, so it can be launched in a separate thread.

Methods:

- void run() – executes a command. Contains command code and calls external objects;
- int getPriority() – returns command priority. It is used to define command processing order;
- Date getCreationTime() – returns command creation time;

- Date getStartTime() – returns command starting execution time.

3.7.2. IUndoableCommand interface

It extends ICommand. It allows to undo/redo a command.

Methods:
- void undo() – undo command
- void redo() – redo command

3.7.3. AbstractCommand class

It implements IUndoableCommand and contains shared behavior of command objects. All concrete objects should extend this class. It implements 'Comparable' to define default sorting order of commands in the queue.

Properties:
- int priority – command priority;
- Date startTime – command start execution time;
- Date creationTime – command create time;

Methods:
- abstract void run() – this method is defined in AbstractCommand subclasses and contains command code;
- Inherited from Comparable, ICommand, IUndoableCommand.

3.7.4. Concrete command classes

The classes extend AbstractCommand and define its abstract run() method. Concrete classes also contain properties for keeping parameters and appropriate constructors to initialize commands.

See Appendix 4.3 for actions list required to implement old XYZ functionality.

3.7.5. MacroCommand class

It also implements IUndoableCommand and implements a command, but it does not contain its own code. MacroCommand contains list of commands which are executed sequentially one after another.
It implements "Comparable" to define a default sorting order of commands in the queue.

It does not have properties since they are computed from included command list.

3.7.6. XYZExecutor class

It performs initialization and tuning of internal Java ThreadPoolExecutor class. GUI classes interact XYZExecutor: they create command objects and pass the to execute() method ThreadPoolExecutor. Commands will be queued and executed according to its priority in thread pool environment.

The class will be implemented as a Singleton pattern.

Properties:

- List<ICommand> history – list to hold the executed commands for undo-redo purposes;
- PriorityBlockingQueue<ICommand> tasks – queue subclass instances used by ThreadPoolExecutor to store issued but not yet executed commands.

Methods:

- void beforeExecute(Thread t, Runnable r) – override ThreadPoolExecutor's method. This is hook to keep command history, logging and statistics maintenance.
- void afterExecute(Thread t, Runnable r) – override ThreadPoolExecutor's method. Purpose the same as beforeExecute()

3.8. Processing & reporting

3.8.1. Overview

Processing and reporting functionality is designed as plugins using Java Plugin Framework (JPF). JPF provides a runtime engine that dynamically discovers and loads "plugins". A plugin is a structured component that describes itself to JPF using a "manifest". JPF maintains a registry of available plugins and the functions they provide (via extension points and extensions).

- ❖ Extension point – a place where the functionality of plug-in can be extended.
- ❖ Extension – particular functionality, the plug-in contribute to the system.

Using JPF brings the following benefits to project:

- Makes the application easily extendable. With extension points you can allow other developers to extend your application simply.
- Improves resource reuse. The JPF extension mechanism allows a programmer to easily share the code and other resources among different applications. Plugins themselves can have extension points, so plugins can reference each other.
- Plugin is a component model. A JPF Plugin is a component that has: a name (ID), a version identifier, code and/or other resources, a well-defined import interface, a well-defined export interface, and well-defined places where it can be extended (extension points). You can think of plugins as a module for your application.

3.8.2. Core

In the core application we define extension points for:

Filter (receives IRawDataProvider):

```
<extension-point id="Filter">
     <parameter-def id="class"/>
     <parameter-def id="name"/>
     <parameter-def id="description" multiplicity="none-or-one"/>
</extension-point>
```

Transformation (receives array of IRawDataProviders)

```
<extension-point id="Transformation">
     <parameter-def id="class"/>
     <parameter-def id="name"/>
     <parameter-def id="description" multiplicity="none-or-one"/>
</extension-point>
```

We define the following types of plugins for XYZ:

3.8.3. Filters

This plugin is connected to Filter extension point, returns filtered data from a IrawDataProvider. It will be implemented as a Singleton pattern. Butterworth filter and rotation are examples of Filter plugin.

3.8.4. Transformations

This plugin is connected to Transformations extension point; it receives a list of IRawDataProviders and returns one IPlotDataProvider object for transformed data. It has PlotDataConsumers extension point to allow next-in-chain plugin connection.

It will be implemented as a Singleton pattern. Examples of transformation are cross-cross and auto-correlation.

3.8.5. Cartesian transformations.

This plugin is similar to Transformation plugin, but it returns a list of CartesianDataPoint for non-timeseries data. It uses CartesianDataConsumers extension point.

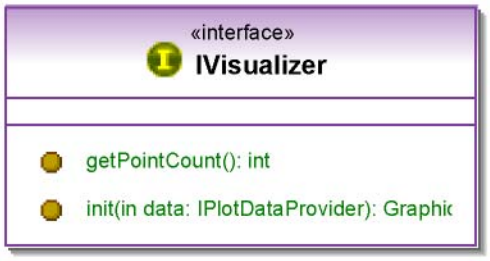
```
<extension-point id="CartesianDataConsumers ">
     <parameter-def id="class"/>
     <parameter-def id="name"/>
     <parameter-def id="description" multiplicity="none-or-one"/>
</extension-point>
```

Fourier transform and particle motion are examples of Cartesian transformation plugin.

3.8.6. Visualizer

This plugin is connected to PlotDataConsumer extension point. It receives PlotData object. Visualizer opens its pop-up window and renders supplied data. It has extension point GraphicsConsumer and exports its standard Java Graphics object.

```
<extension-point id="GraphicsConsumer">
     <parameter-def id="class"/>
</extension-point>
```



3.8.7. Cartesian Visualizer

The same as Visualizer, but connects to CartesianPlotData and draws non-timeseries objects.

3.8.8. Exporter

It connects to GraphicsConsumer extension point, uses a corresponding Graphics object and exports it as a a file in various graphics formats.

```
<extension-point id="ImageConsumer">
     <parameter-def id="class"/>
</extension-point>
```

Proposed image formats: output of XYZ are:

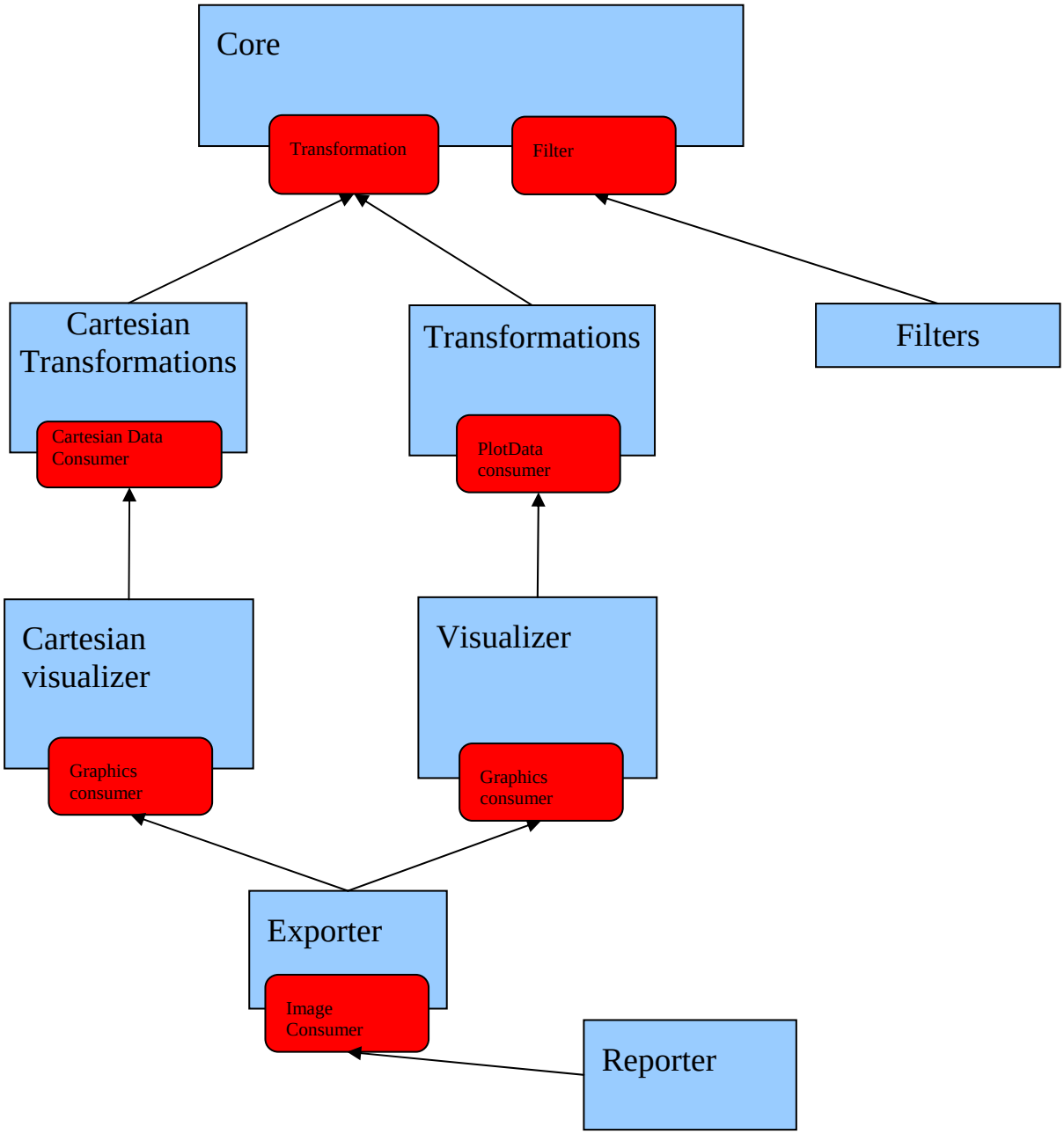
- PNG – standard javax.imageio package
- SVG – scalable vector graphics. http://xmlgraphics.apache.org/batik/. Firebox has built-in support of svg; IE and safari needs a plugin (for example, from Adobe)
- PS, PDF – iText library

3.8.9. Reporter

This plugin connects to ImageConsumer extension point, and will be used to create reports (HTML files with descriptions and references to images)

3.8.10. Plugin Diagram

Blue blocks represent modules, red rounded rectangular are extension points.



4. APPENDIX

4.1. List of specs in table format

Inputs

Configuration	XML	See isti-utils
Equally sampled timeseries	MiniSEED	See JQL
Equally sampled timeseries	In the clear	?
equally sampled timeseries	Socket	Protocol will be specified later. See DHI
Auto QC	XML	Protocol will be specified later
State of health voltage	XML	Protocol will be specified later
state of health latency	XML	Protocol will be specified later
Earthquake information	XML	Protocol will be specified later
Picks	XML	Protocol will be specified later
Unequally sampled timeseries	Socket	Protocol will be specified later
Metadata	Dataless SEED	See SHAPE, DMC's JavaSeed
Metadata	RESP	See jPlotRESP, jEvalRESP

Outputs

Equally sampled timeseries	MiniSEED	?

Unequally sampled timeseries	XML
Non-timeseries data (spectra etc)	In the clear ?
Plots	PNG
Reports	HTML
Flow control	XML – and provided API

Editing

Manual HTML reports	
Auto QC report	?
MiniSEED header for point	View only

Transformations

Demean
Spectral smoothing
Spectra
Baseline spectra from external XML
Filtering -filters preset
Convolving response from preset of generic instrumental responses
Deconvolution

Polarization

CrossCorrelations

Autocorrelations

3-D rotation

Rotation into the ray frame

Rotation into great-circle path

Rotation into UVW

Trace energy minimizing by arrival azimuth

4.2. Old XYZ startup

At a startup the following user menu appears:

```
macSol-ilya 65> xyz test.gfs

Choose browsing mode:

1 ... step through file 1 trace at a time

2 ... panels: |??Z|??N|??E|??R|??T|

3 ... panels: |XXZ|XXN|XXE|XXR|XXT|

4 ... panels: |XXZ|XXN|XXE|XXR|XXT|, gfstype 3

5 ... all traces, by station, by sample rate

6 ... two stations/screen, by channel
```

7 ... 16 stations/screen

8 ... same as mode 5, but one chan/panel

9 ... all traces, by station, sample rate, & event

10 ... 16 stations/screen, by event (type 3)

In the table below we describe startup options of the existing XYZ.

#	Option	Description	New XYZ2
1	step through file 1 trace at a time	Does exactly this: one trace/panel per screen	Need to be reproduced
2	panels: |??Z|??N|??E|??R|??T|		
3	panels: |XXZ|XXN|XXE|XXR|XXT|		
4	panels: |XXZ|XXN|XXE|XXR|XXT|, gfstype 3	Not supported	Not supported
5	all traces, by station, by sample rate		
6	two stations/screen, by channel		
7	16 stations/screen		
8	same as mode 5, but one chan/panel		
9	all traces, by station, sample rate, & event	Not supported	Not supported

10	16 stations/screen, by event (type 3) enter choice:	Not supported	Not supported

4.3. Old XYZ buttons description

#	Button	Action	Description	Modifications for new XYZ
1	PLT	A) plot	Redraw screen using currently available in memory (what happens if we update disk data?) panels	Keep this functionality
2	PLT	B) sel	Select panels to plot/return to main screen. A user selects traces on the command line	Keep this functionality
3	PLT	C) ovr	Overlay panels/return. Trace selection is done via command line; second invocation of overlay is a return to a non-overlaid position	Keep this functionality
4	N/A	A) next+plot	Next trace set + screen redraw + printing the list of SNCLs in the command line space	Keep this functionality
5	N/A	B) next	Next trace set + printing of SNCLs without redrawing. You need to click PLT: A) option to draw data. This is obviously done for speed-up of stations listing....	Keep this functionality
6	N/A	C) back	Back to previous time range: same as next but in opposite direction: no trace are being drawn	Keep this functionality
7	TOG	A) phases	Draw phases marks if available. Requres allorder.bin file with EQ info and gsn_sta_list file with station info. Also requires HRVDBS env variable to be set and point to the dir where above files live.	Keep this functionality
8		B) color	Removes all colors from the current screen. Second invocation puts colors back.	Keep this functionality
9		C) mean	Demean/return. Useful in overlay mode.	Keep this functionality
10	DMP	A) SAC	Dump to SAC file format	Upon request

11	DMP	B) GFS	Dump to GFS file format	Upon request
12	DMP	C) ASCII	Dump to ASCII file format	Upon request
13	quit		Quit from program	
14	PHS	A) +	Not known: Sent info request to ASL	
15		B) -	Not known: Sent info request to ASL	
16		C) EQ ID	Not known: Sent info request to ASL	
17	EXPORT	A) hardcopy	Prints the current frame to printer	Keep this functionality
18		B) Pretty plot	Prints current frame to file	Replace this button with an option to create PNG and HTML
19	pick/offset	A) offset	An option offsets segments in Y dimensions so gaps could be seen clearer	Keep this functionality
20	pick/offset	B) ttpick	Not known: Sent info request to ASL	
21	pick/offset	C) delpick	Not known: Sent info request to ASL	
22	FLTR	A) lp	Low freq filter	Keep this functionality
23	FLTR	B) bp	Band-pass filter (low and high cutof frequencies are defined)	Keep this functionality
24	FLTR	C) dyo	Design you own filter (seems to be broken)	Keep this functionality
25	SCL	A) auto	Y-scale panel independently for each panel	Keep this functionality
26		B) com	Y-scale panel based on maximum value for ALL panels	Keep this functionality
27		C) xhair	Not known: Sent info request to ASL	Keep this functionality
28		A) PPM	Particle motion pop-up window	Keep and enhance this functionality
29		B) PSD	Power Specrtal Density pop-up window	Keep and enhance this functionality

30		C) RESP	Instrumenal resonse pop-up window	Keep and enhance this functionality
31	LIM	A) xlim	Manual cutting of X-axis: interval is set in a command line	Keep and enhance this functionality
32		B) ylim	Manual set of Y-axis; Y-interval is defined on a command line	Keep and enhance this functionality

4.4. Java Speed tests

ISTI has performed several tests in order to convince ourselves that XYZ2 written in Java can be comparable in speed to the original XYZ implementation. First and foremost we have been interested in the speed of the canvas drawing of the waveforms. To answer the question we have developed a simple application[4] for rendering. It can render three tab controlled pages consuming all application window area. Three tabs in the application are:

1. A simple graph of a function which is computed a run-time (in our case sinus). Number of samples to plot is a configuration parameter; default number of samples is the canvas width in pixels. This tab represents a panel with a single waveform of XYZ.
2. Same as above but with 10 waveform panels in a canvas.
3. Large raster image (covering the whole canvas). Here we tested a speed of bitmap caching of the plot data.

We ran our test program on modern PC processors (Intel Dual Core and Intel Centrino; 2 GB of RAM) using Mac OS X (MacBook Pro) and Linux FC6 (Toshiba notebook 1600 Mhz) and via X Window between these machines. X Windows connection was done via high-speed LAN.

We found that tabs 1 and 2 (1 data panel and 10 data panels) were rendered quick and smooth in all three environments. Time for rendering is almost linear proportional to the number of samples (parameter in our program), but negligible for number of samples less than about 100,.000's.[5]
Tab 3 (bitmap image) renders fast in a native environments, but we observed significant annoying delays (up to 5 seconds) running via X Window. This delay is not coherent, it varies between 0.5 and 5 seconds, mostly because of X Window caching.

Thus, we conclude that Java graphical performance should be sufficient for a speedy XYZ2 application. We will avoid bitmap caching and not use this technique in the design and implementation of XYZ2.

[4] Available upon request

[5] Natural consequence of this "discovery" is that we need to be accurate and not plot all data points in the canvas, but use algorithms of smart averaging, in particular, those which can be found in the Util library of ISTI.

AUTHORS

ISTI Team: Max Kokoulin, Ilya Dricker with additional support from Paul Friberg and Sid Hellman.

BUG REPORTS

Send bug reports to:

info@isti.com