



Chapter 4

4. xSonify – The Application

This chapter describes the functionality of xSonify followed by the detailed structure and technical inner life.

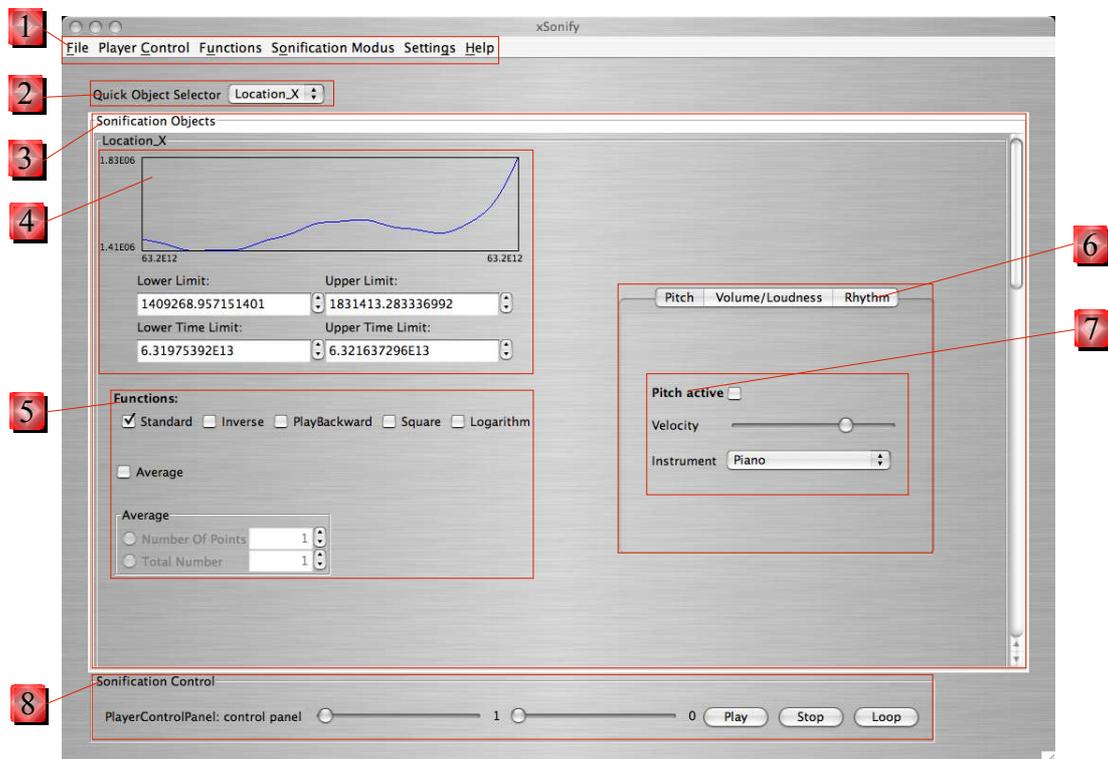


Figure 4.1: xSonify Main Window

4.1.1 Functional Survey – How xSonify Works

Before I explain the handling of xSonify I would like to give a general overview how the application works. As mentioned in the beginning of this thesis the program provides basically the opportunity to display numerical data as sound with the help of three different kind of sound attributes. Attributes like the pitch, the volume and the rhythm of sound.

In order to start the Sonification process the numerical values have to be converted into values of the internal data structure. The data values of this structure are floating point variables in the range from 0.0f to 1.0f.

0.0f represents therefore the smallest and 1.0f the largest value of the original data.

To realize the idea of Sonification, xSonify takes advantage of the MIDI support from JAVA. To display the information of numerical data for instance by dint of the pitch of a played music instrument the smallest value (0.0f) represents the lowest frequency and the largest value (1.0f) the highest frequency according to the settings. Each tone represents one value and the whole sequence of different tones accordingly the whole dataset.

The user can also assign different Sonification modi or different instruments to each dataset. This option is necessary if the user wants to distinguish the different datasets while listening to them at the same time.

Sonification provides naturally also a chance for blind scientists to work with data and needs speech support. xSonify provides the user optionally with its own speech support software – independent from commercial screen reader software.

4.1.2 How To Work With xSonify

Figure 4.1 shows the main window of xSonify. Basically the application is based on different software modules and the GUI Module is one of them. It would be even possible to operate xSonify from the console without the GUI support or to replace it with another GUI by paying attention to the interfaces. The main window is separated in different sections. Beginning with the menu bar **(1)**, the “Sonification Object” **(3)** section and the “Player Control” **(5)** section.

To work with the application the user has to import the data he wants to sonify. In order to do that he has two options:

- He can use the function “File => Import Data” which is based on a resource toolkit from the application ViSBARD. With this function he can access a remote database or a local file in order to retrieve the data. Both ways handle files with the formats like *.cdf and *.vba.
-
- The second option to retrieve data is with the function “File => Import Data Textfile”. This option simply reads a text file with the data according to the file structure explained in Chapter 4.2.3.5 Package: *textfile*.

After the data are successfully imported the Sonification procedure can begin. The new imported data create for each Sonification object one panel (3). The “Quick Object Selector” (2) provides an overview of the existing data objects and the user can select the requested data object directly without scrolling to it.

Each Sonification object panel exists of a data plot (4) which plots the data as a simple diagram. Later on during the play-back a cursor displays the current position in the sequence. It is also possible to define bounds of the displayed object.

xSonify provides also a pool of functions (5) which can be applied to the corresponding object.

On the right side of each Sonification object (3) a tabbed field (6) with the choice of three Sonification modi including their appropriate settings (7) enables the user to add the specific Sonification object to the sequence.

It is only possible to apply one modus for one Sonification object. If none of the three modi is selected (7) the whole Sonification object will not be considered for the Sonification procedure.

The sequence will be created and played after a click on the “Play” button in the “Sonification Control” panel (8). The play-back can be interrupted or changed into an endless loop. It is also possible to change the playback-speed and to move the current position directly by moving the sliders.

The GUI components and their actions can be optionally read by xSonify. Independent from screen reader software this feature opens up visually impaired people the usage of this application.

The following Chapter 4.2 Technical Architecture will deal with the detailed technical background of xSonify.

4.2 Technical Architecture

During the design phase of the application I focused on modularization which has the following advantages:

- easy to understand
- easy and quick replacement of existing modules
- structured and clearly arranged

Additionally to this chapter I would like to refer to the Java documentation of xSonify which is available as HTML files and the *Appendix A* and *B* which comprises the UML diagrams.

4.2.1 Module Overview

For the abstraction of the modules I chose a very simple meta view to display the different modules and classes while displaying the direct relationships between the individual modules as the overlapping areas.

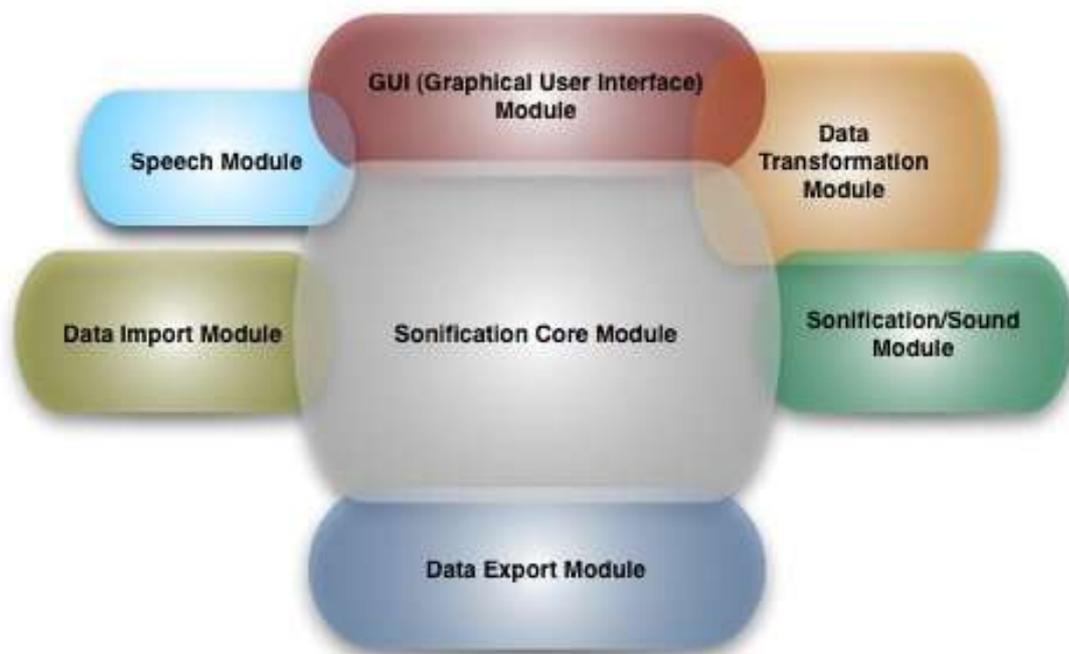


Figure 4.2: Module Overview

Every single module can be considered as its own package in the program hierarchy and contains at least one class or interface. The single modules will be introduced in the following chapters and the detailed class information can be found in the HTML-Documentation.

4.2.2 Sonification Core Module

As the name already describes, this module is the core piece of xSonify which includes also the main function in the *Sonification_Core* class. Beside this main class the module contains also other classes which are representing the internal data structure of xSonify. This data structure keeps internally the data after the data import.

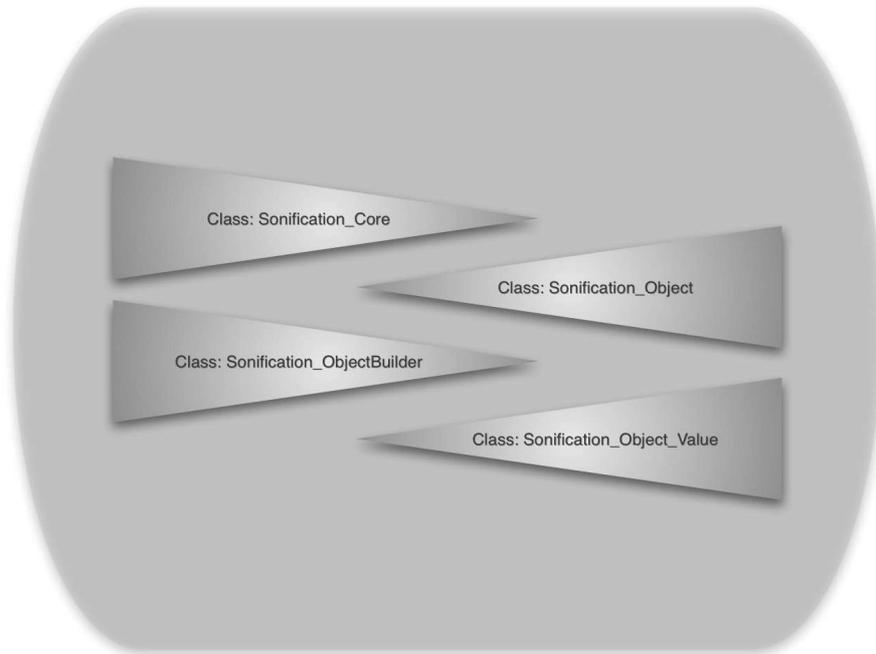


Figure 4.3: Sonification Core Module

4.2.2.1 Class: *Sonification_Core*

The *Sonification_Core* class is the entry point into xSonify. Beside the main function it also has several functions to control the data import, create the GUI(Graphical User Interface) and an instance of the Sonification/Sound Module. It also provides an user interface consisting of a specific list of functions which can be invoked by typing in letters into the I/O-console in case the GUI module is not included.

The start procedure with all the creation and initializations activities are described in detail in the appropriate UML diagrams.

The to most important variables in this class are the:

- *llSonification_Object_original*
- *hSonification_Object_original*.

They contain the original data objects thru the whole program duration in the structure which will be described in the next chapters. There are two structures where the references to the data objects are kept. The first is a *LinkedList* which is preferably for appliances concerning the whole data. The second is a *HashMap* which is for the appliance of functions on selective data objects.

4.2.2.2 Class: *Sonification_ObjectBuilder*

Sonification_Object_Builder is the foundation or base class for xSonify's internal data structure. It organizes and keeps the original data right after the import for the whole time the application is running. It builds initially the internal data structure of xSonify. It creates instances of the class *Sonification_Object*. Every instance of the *Sonification_Object* represents a combination of an original variable from the source datasets and the corresponding time. The detailed body of the *Sonification_Object* class however will be explained more in detail in the following two subchapters.

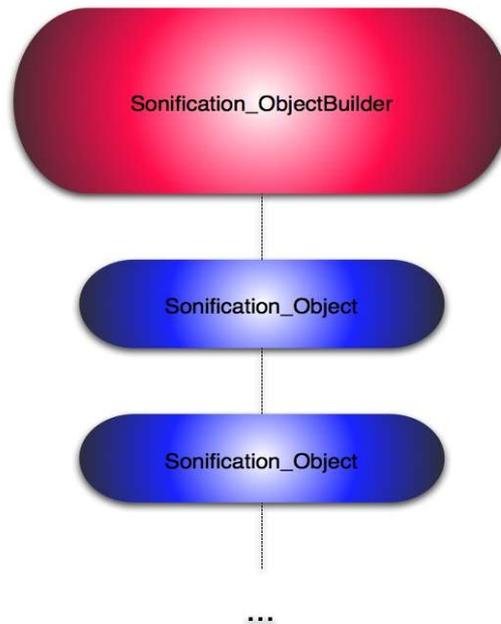


Figure 4.4: Class - *Sonification_ObjectBuilder*

The instances of the *Sonification_Object* classes are stored in two different kind of data structures. The first is a *LinkedList* and the second is a *HashMap*. The reason for this is easy. For functions (e.g. *BuildStandardTransformedList()* in class *Sonification_Object_Transform*) that want to access all the *Sonification_Objects* sequentially, the fastest way to do this is to run thru all the elements of a Linked List. But if a function needs to access a certain *Sonification_Object* directly by delivering the name of the object a data collection like a *HashMap* could be very useful (e.g. *doStandardTransformation(String subjectname)* in class *Sonification_Object_Transform*).

Before the user quits the application the two data objects *llSonification_ObjectList* and *hSonification_ObjectList* are stored automatically in a persistent external file called "*lastsession.obj*". During the program start it checks if such a file exists and if so it will be used and the data from the last session will be recovered as default values. If not, the data object panel is empty and can be filled by importing data.

4.2.2.3 Class: *Sonification_Object*

As mentioned before this class is an important part of the internal data structure of xSonify. Every *Sonification_Object* represents a certain variable, attribute or measurement parameter of an space science file. Each single *Sonification_Object* is added into the *LinkedList* and *HashMap* data structures as a reference.

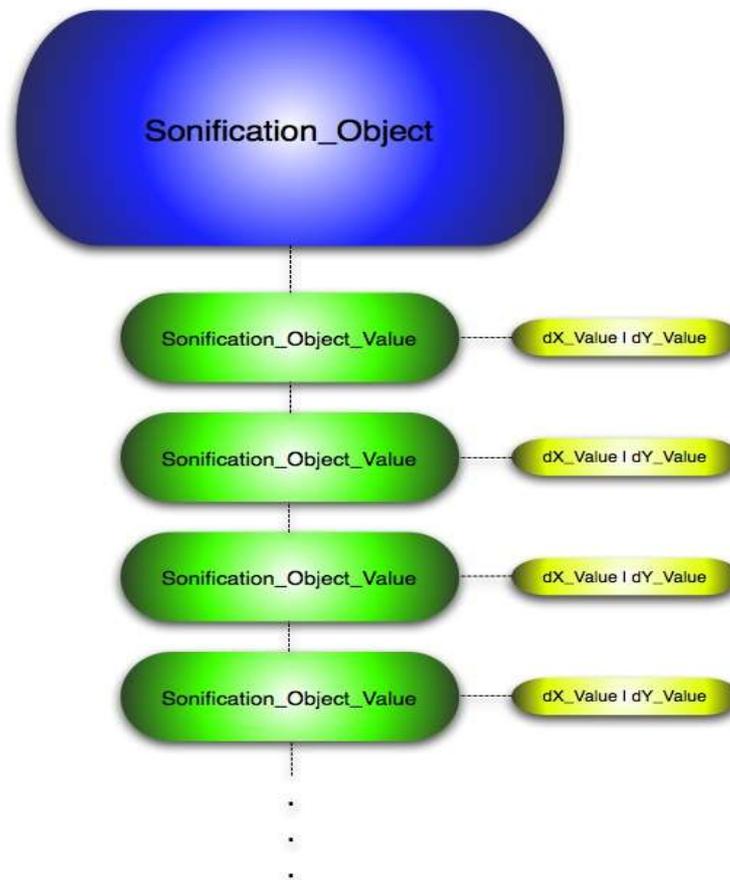


Figure 4.5: Class - *Sonification_Object*

All the single *Sonification_Object_Value* instances are stored in an array list which is represented as the green symbols.

Every *Sonification_Object* has beside the object name also very important parameters like the minimum and maximum of all the x and y variables and additionally information about bounds which are initially set to the minimum and maximum respectively. The Sonification takes place only within the valid bounds.

4.2.2.4 Class: *Sonification_Object_Value*

The object which contains the actual data values of a certain Sonification object is an instance of the class *Sonification_Object_Value* (Figure 4.5: Class - *Sonification_Object*). It holds two value of the data type Double. The first value represents the corresponding time (*Double: dX_Value*) of a measurement and the second contains the acquired value itself (*Double: dY_Value*).

4.2.3 Data Import Module

As mentioned before xSonify should be used as a standalone program as well as a additional software module for already existing space science applications. In the second case it is necessary to have an interface to retrieve the data from the applications internal data structure.

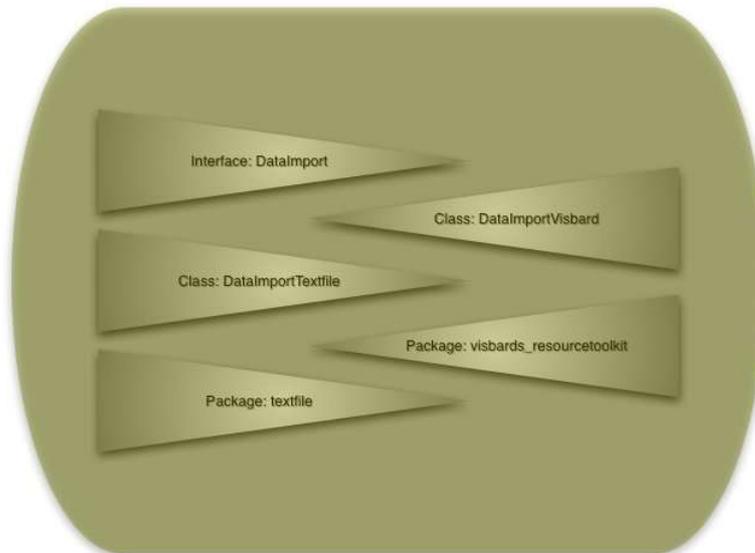


Figure 4.6: Data Import Module

4.2.3.1 Interface: *DataImport*

As a result of the variety of data import opportunities(e.g. import from a textfile) it is necessary to build a kind of standard of allowed functions which can be called by the class *Sonification_ObjectBuilder* access the data from the appropriate data import class like *DataImportVisbard* and *DataImportTextfile*. Classes like the mentioned have to implement this interface *DataImport* which defines the necessary functions. This interface makes sure that the necessary function will be implemented.

4.2.3.2 Class: *DataImportVisbard*

The purpose of this class and all of the *DataImport*-classes is to request data from outside the application xSonify and prepare them for an easy and standardized access from inside the application by objects of the classes like *Sonification_ObjectBuilder*. The class creates an object of the class *visbards_resourcetoolkit_main* and has thus access to the internal data structure of the *visbards_resourcetoolkit*.

4.2.3.3 Class: *DataImportTextfile*

This class looks from inside the application xSonify the same like the class *DataImportVisbard* does. It offers the same selection of functions as the interface *DataImport* which both classes are implementing. Inside the functions of course the implementation looks different since they have to communicate with an instance of the class *TextfileParser* created in the constructor of *DataImportTextfile*.

4.2.3.4 Package: *visbards_resourcetoolkit*

This package is a very complex data retrieval module from another application (friendly supported by the ViSBARD team) which allows the application to retrieve space science data stored for example in CDF files locally or from remote databases via the Internet.

4.2.3.5 Package: *textfile*

The class *TextfileParser* from this package provides an opportunity to access text files for the data retrieval. It includes the selection of the file by a *FileChooser* and is basically a text parser which takes advantage of Java's *StreamTokenizer* class. The data are stored in a data structure similar to the xSonify's internal data structure consisting of an *ArrayList* which has as elements *LinkedList*'s for the amount of single values. Each *LinkedList* represents one column of values of the text file. The first element (index 0) is the name of the column if available or an automatically created name like “*Time, Value_1, Value_2*”.

The text file is basically organized in columns. Each column represents one data object whereas the first column needs to represent the time axis. Optionally the first line of every column can also be a text describing/naming the object whose values follow the lines underneath.

In order to read the data from a file it needs to be structured like the following Backus Naur Form:

```
<file> ::= [<header_line>] {<data_line>} <EOF>

<header_line> ::= {<header_text><TABULATOR>} <EOL>
<header_text> ::= {<letter> | <digit> | <special>}

<data_line> ::= {<data_value><TABULATOR>} <EOL>
<data_value> ::= {<digit> | <special>}

<letter> ::= a | b | . . . | z | A | B | . . . | Z
<digit> ::= 0 | 1 | 2 | . . . | 9
<special> ::= .
```

4.2.4 GUI (Graphical User Interface) Module

To ease the user interaction with xSonify the application provides of course a graphical user interface. The main class of this module is the class *Sonification_MainWindow* and comprised of the following classes.

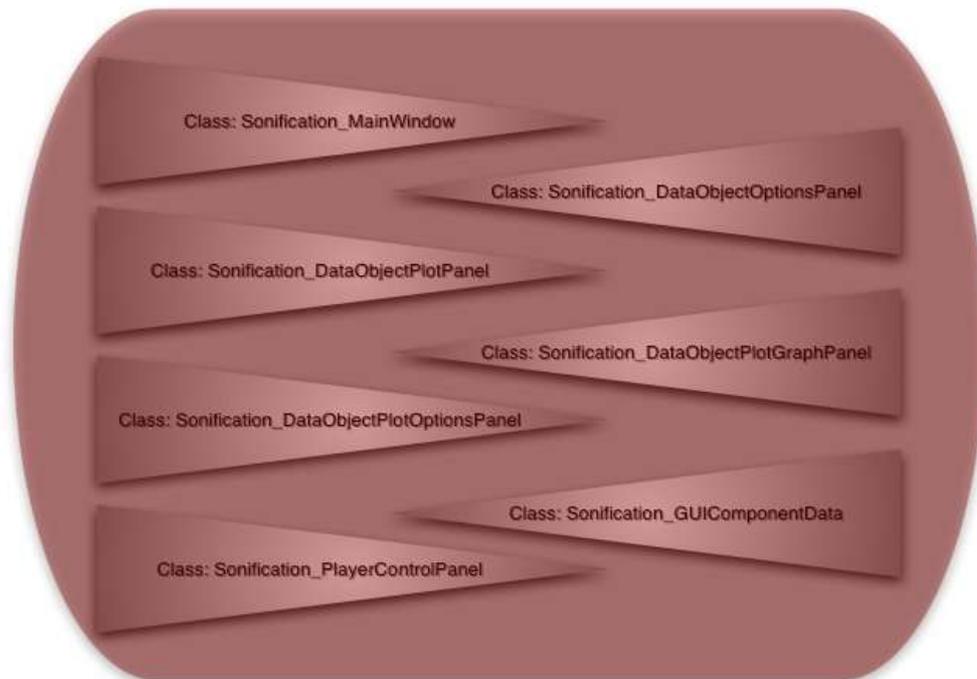


Figure 4.7: GUI Module

4.2.4.1 Class: *Sonification_MainWindow*

An instance of the class *Sonification_MainWindow* is created by the main class or rather core class *Sonification_Core*. It receives a reference to an object of the *Sonification_Object_Transform* class and another reference to an object of the *Sonification_Sound* class. The *Sonification_MainWindow* builds the GUI together with the following classes in this chapter. The GUI can be split up in several sections. Sections like the menu, Sonification object area which contains a list of all loaded data objects and the third section which represents the Sonification player control. The Sonification object area itself contains again some subareas. Each area is realized by a subclass of *JPanel*.

4.2.4.2 Class: *Sonification_DataObjectPlotPanel*

This panel unites the two panels of the following two classes in this chapter. It was created to separate the objects of the two classes since the class *Sonification_DataObjectPlotGraphPanel* contains the graphical plot based on Java 2D technology. A repaint or rather a refresh of this panel can be made independently to the other panel. Another reason is also to keep it more structured and easier to understand.

4.2.4.3 Class: *Sonification_DataObjectPlotGraphPanel*

Objects from this class display the data graphically in a 2D plot. As mentioned before the applied technology is Java 2D. To accomplish the graphical display it was necessary to prepare the data by mapping all the values in a range from 0 to 1 in a parallel data structure which is explained in detail in *Chapter 4.2.6 Data Transformation Module*.

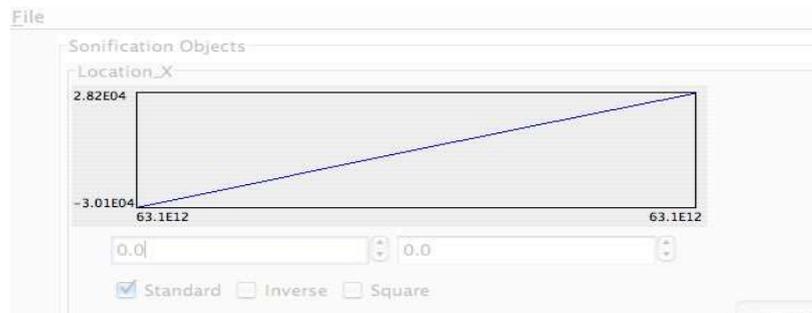


Figure 4.8: Class - *DataObjectPlotGraphPanel*

Beside the graphical display of data the graph gives also information about the current position in the played Sonification sequence in form of a vertical, red line and bounds of the area which is supposed to be sonified. The bounds appear as green lines and can be set in the panel which is described in the following chapter.

4.2.4.4 Class: *Sonification_DataObjectPlotOptionsPanel*

To modify the data in the plot and also later on for the Sonification this panel provides some functions to limit the area which has to be sonified. Limits like an upper and lower bound of the x- and y-values. It also offers to apply an inversion and square of the y-values and of course the standard function which brings the values back into the original state.

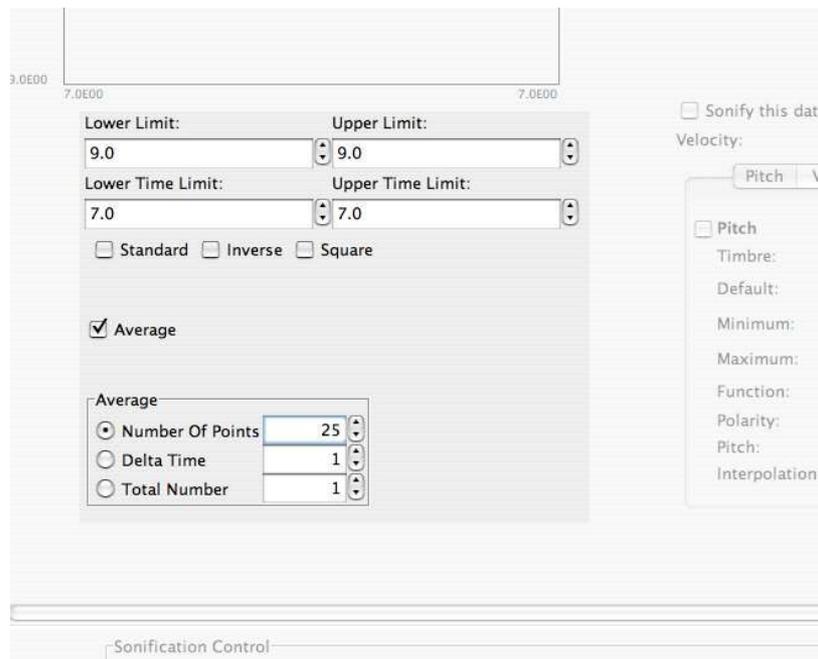


Figure 4.9: Class - *Sonification_DataObjectPlotOptionsPanel*

Another function is to build the average over y-values. The criteria how to build the average can be defined in the by choosing between the three radio buttons. The result can be seen in the plot immediately after the selection.

4.2.4.5 Class: Sonification_DataObjectOptionsPanel

To select and to configure a data object for the Sonification procedure this class offers functions for the choice of the Sonification modus, instrument and strength of the played instrument. The different Sonification modi are represented in the *JTabbedPane* and can be selected and configured.

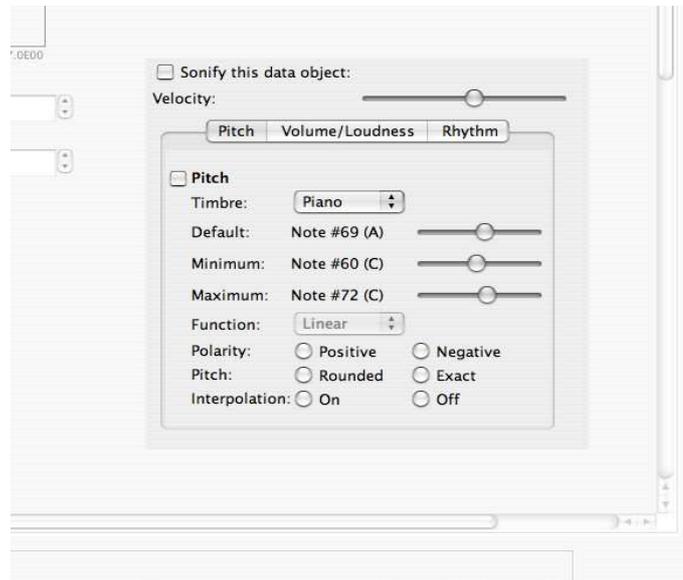


Figure 4.10: Class - Sonification_DataObjectOptionsPanel

4.2.4.6 Class: Sonification_PlayerControlPanel

The main goal of this software solution is to explore the data as mentioned in the summary. The scientist should be able to “play” with the data by using the user interface. Therefore it is necessary to provide the user with extended control functions additionally to a simple “Play” and “Stop” button.

Functions like setting the current sequence position or the speed of the sonified data sequence.

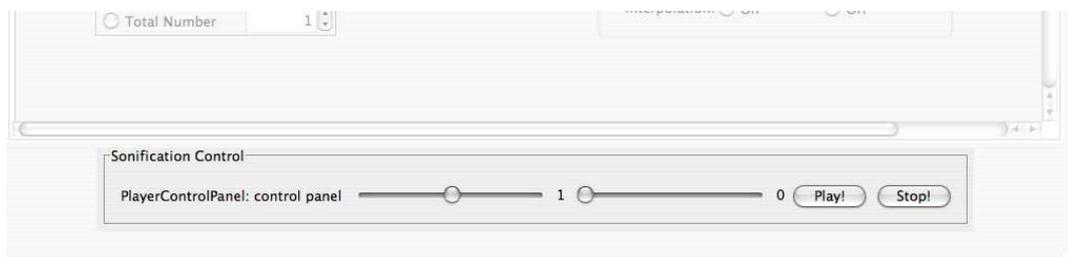


Figure 4.11: Class - Sonification_PlayerControlPanel

4.2.5 Sonification/Sound Module

This module of the application represents the technical conversion of the data into sound. For the general technical background of the Java Sound API I would like to refer to *Chapter 3.5 MIDI*. For the better understanding of the activity there is also an UML diagram in *Appendix B*.

4.2.5.1 Class: *SonificationSound*

The class *SonificationSound* identifies the selected Sonification objects, chosen Sonification modi and instruments. It creates with classes of the Java Sound API, which is described in *Chapter 3.5 MIDI* a MIDI sequence which can be played and navigated by the functions of the *PlayerControlPanel* class in the GUI.

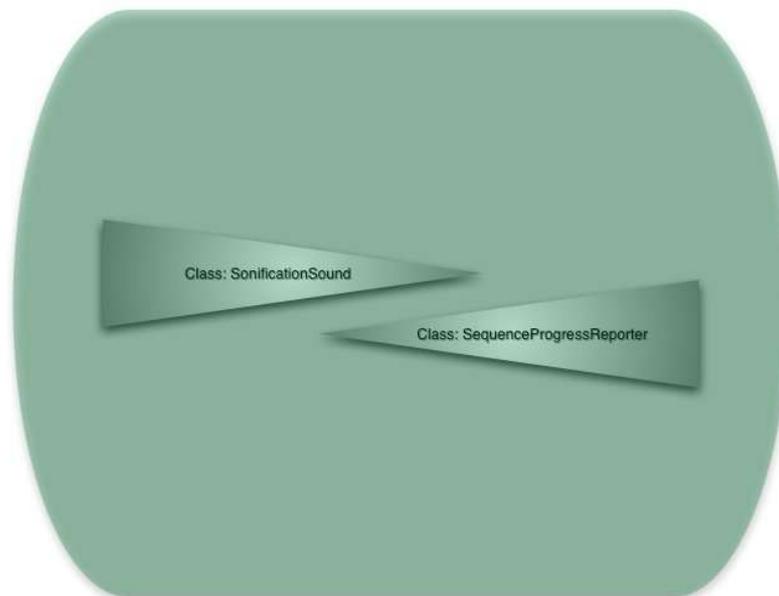


Figure 4.12: *Sonification/Sound Module*

The core of this class builds the function *createSequence()* which puts the single MIDI events according their mapped values of the data structure together to tracks and finally to a sequence.

The different variations of Sonifications like

- pitch
- volume/loudness
- rhythm

are generated in the *SonificationSound* class as well.

4.2.5.2 Class: *SonificationSound*

Design Pattern in Sonification/Sound Module: Observer Pattern

Unfortunately the development of the Java Sound API from Sun seems in comparison to other Java technology packages a little bit neglected. Especially the limited range of functions of the sequencer class implied to create solutions like the graphical update of the current position of the sequence which was solved by a popular design pattern: The Observer Pattern.

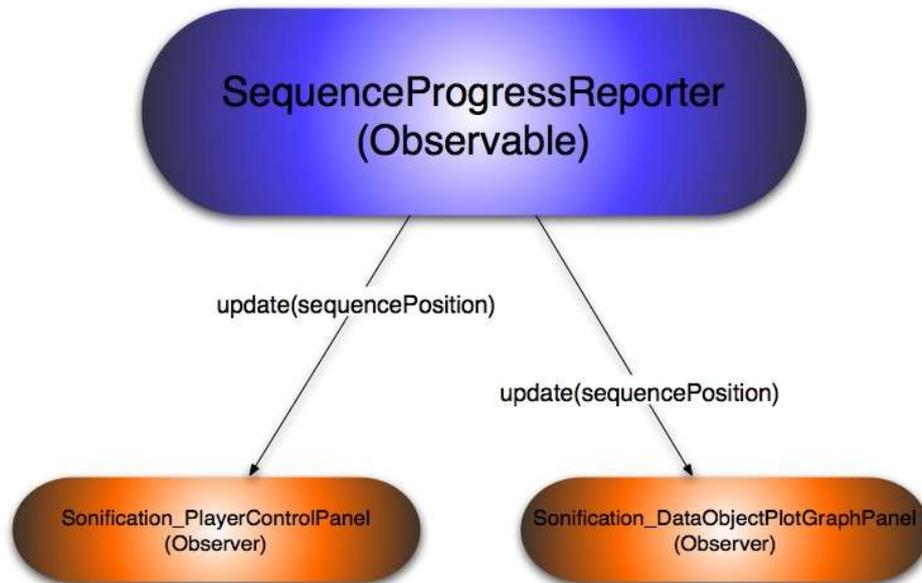


Figure 4.13: Class - *SonificationSound*

This observer pattern shows that there are two observer objects which are waiting for the invocation of their *update(sequencePosition)* function as the current sequence position as parameter. As soon as the sequence is started in the sequencer, a thread in the class *SequenceProgressReporter* will be set into the state “running”. The thread checks every 400 milliseconds the current position of the played sequence and calls the *notifyObservers(sequencePosition)* function in the class *SequenceProgressReporter* which is a subclass of the class *Observable*. This invocation forces the observable object to call the function *update(sequencePosition)* to set the current sequence position in the two *Observer* objects.

4.2.6 Data Transformation Module

The raw data as they are stored in xSonify's internal data structure (Objects: *llSonification_Object_original*, *hSonification_Object_original*) can be considered as the initial point of the data operatorability. This data structure is used to build a transformed data structure similar to the original data structure but with transformed values. Transformed means the values are mapped into a value range between 0 and 1. The advantage of this procedure is to make the data available in an independent form regarding their ranges and scales.

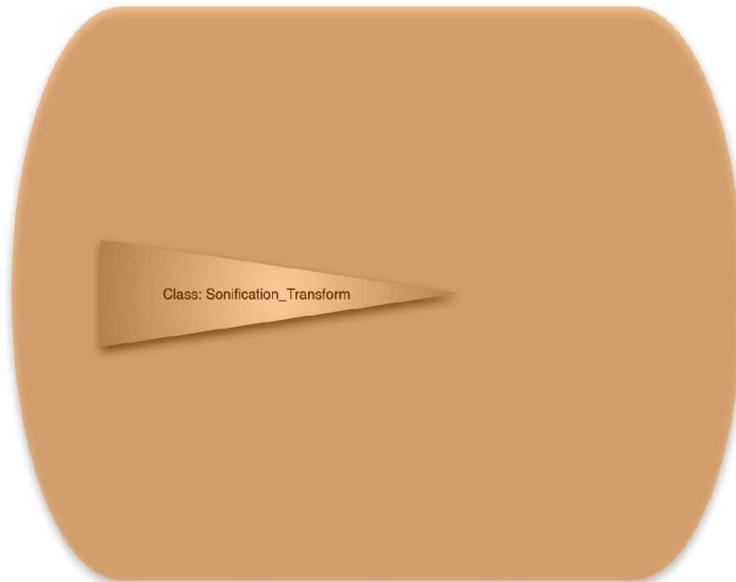


Figure 4.14: Data Transformation Module

In order to support the researchers in gaining better results from the scientific aspect the application needs to have the ability to transform or rather to change the data for their purposes. For the execution of transformations xSonify provides the user with a selection of different functions introduced in the following sub chapters.

Every result of the appliance of such a function can be seen immediately in the 2D plot and later on heard during the play of a sequence. Technically there is only the public function *objectTransformation()* which invokes the appropriate private functions need for the requested transformation.

This function can be considered as a relay function which receives certain parameters and decides which functions inside the class *Sonification_Transform* need to be called.

4.2.6.1 Functionality “Standard”

The function “Standard” is the initial function which is invoked right after the start of the application the first time. After every appliance of a function the initial state can

be reached by calling this function again.

4.2.6.2 Functionality “Inverse”

Sometimes it is just useful to see thing inverse. The function “Inverse” displays every y-value upside down. It just inverts every standard value which is basically in the range of $0 < y < 1$.

4.2.6.3 Functionality “Square”

The function “Square” builds the square of every single y-value in the transformed data structure. To square every value is for certain variables important to compute power from energy.

4.2.6.4 Functionality “Logarithm”

The function “Logarithm” builds the logarithm of every single y-value in the transformed data structure. This function makes sense if some of the values are tight together. After the function the values are stretched which improves the identification.

4.2.6.5 Functionality “Average”

The function “Average” combines the single values to groups and builds the average of all y-values inside a group. The size of the groups depends on the values in the appropriate *JSpinner* box of the *Sonification_DataObjectPlotOptionsPanel*. The visual result of this function can be seen in the plot as a kind of histogram and be heard as sound with a longer duration.

4.2.7 Speech Module

To enhance the user interface especially for visual impaired people the Speech Module will give xSonify the ability to talk. The class *Sonification_Speech* is based on the Java Speech API.

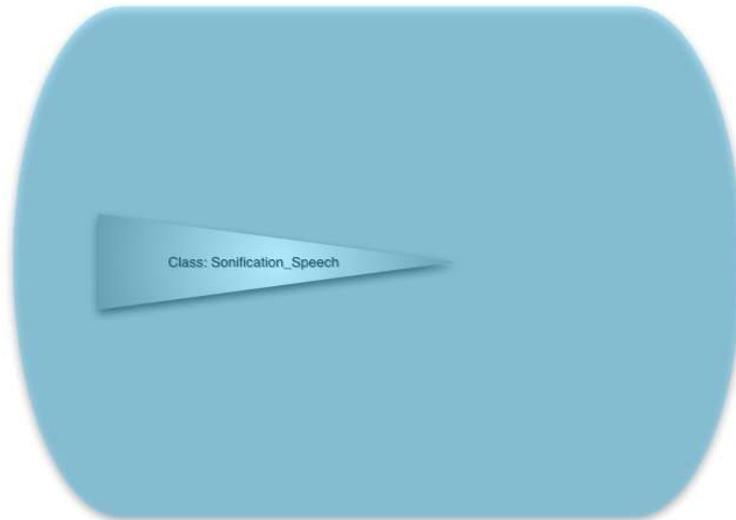


Figure 4.15: Speech Module

The creation of an object of this class takes place in the *Sonification_Core* class and the reference to this instance will be passed to instances of the GUI module. Whenever an event occurs (e.g. *FocusEvent* for a SWING component) which requires an verbal output the function *speak(Text)* can be called with the spoken text as the function parameter.

4.2.8 Data Export Module

After the Sonification sequence was generated successfully and the result seems promising to the scientist it is very useful to archive this sequence as a sound file. It could be also very useful to exchange this result with other colleagues or as material for a presentation for example. Hence it is necessary to provide this sequence in a common sound format.

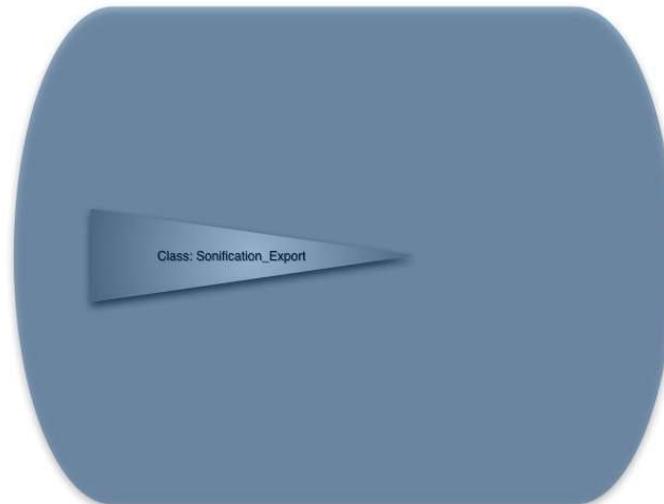


Figure 4.16: Data Export Module

The class *Sonification_Export* should provide methods to deploy the Sonification results as common sound formats. Beginning with the *.mid* format in future the class should be extensible for more sound file formats like *.wav* and *.mp3*.

4.3 Implementation of xSonify as a module into existing applications

As mentioned in Chapter 2 – Existing Space Science Applications, xSonify can also be added into applications as a module.

4.3.1 Implementation in TIPSOD and CDAWeb+

The appearance of xSonify in the applications TIPSOD and CDAWeb+ will be limited to a simple button or menu item. As soon as this component is activated a new instance of the application xSonify will be created and a new independent window containing the application pops up at the screen. Unfortunately the two applications TIPSOD and CDAWeb+ don't have an internal data structure of the focused data so that the data retrieval and access needs to be managed autonomously by xSonify.

4.3.2 Implementation in ViSBARD

In comparison to the first two applications, xSonify can be fully implemented in ViSBARD. One way of xSonify's data import is based on ViSBARD's "Resource Toolkit" and xSonify accesses consequently the internal data structure of ViSBARD.

The following sequence diagram should illustrate how the invocation of the Sonification module takes place.

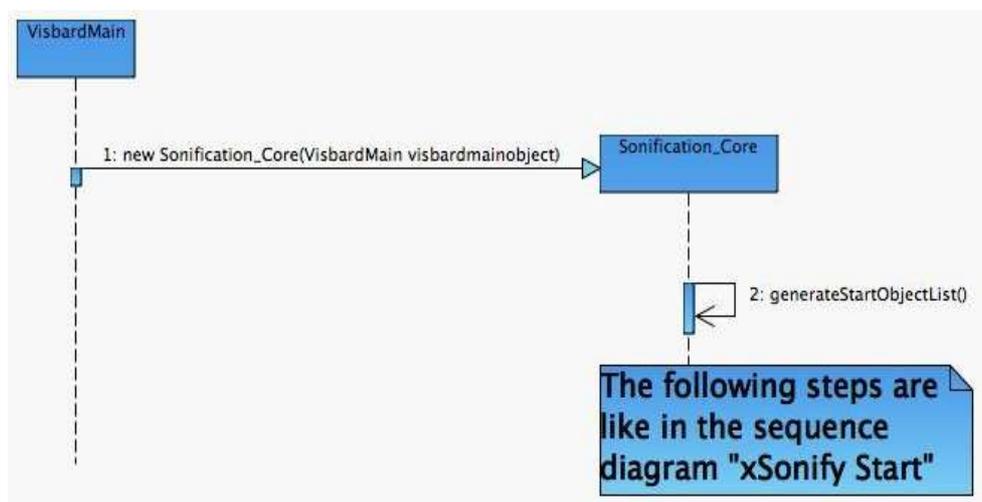


Figure 4.17: Implementation Of xSonify

The class *DataImportVisbard* in the *Data Import Module* needs to be adapted. In order to do this it is necessary to add further constructors into the classes *Sonification_Core* and *DataImportVisbard* with *VisbardMain* as parameter. The first instruction within the new constructor of the class *Sonification_Core* should be the initialization of the

reference *dDataImport* like:

```
try{
    dDataImport = new DataImportVisbard(visbardmainobject);
}
catch(Exception e){}
```

It is also necessary to remove the original *DataImport* functionality from xSonify.