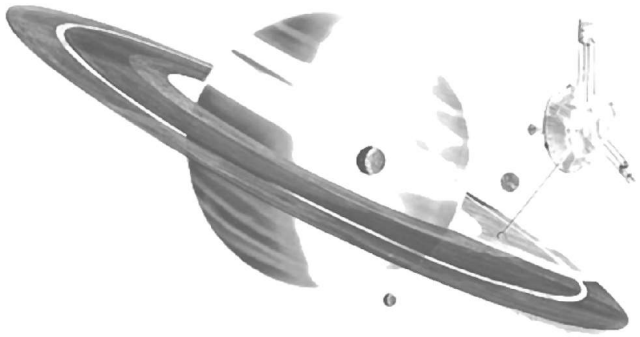
Table of Contents



Chapter 1

1. Introduction Of Sonification

1.1 Definition Of Sonification

Sonification in one sentence described is the use of non-speech audio to convey information. The expression Sonification comes from the two latin syllables “sonus” which means sound and the ending ”fication” which forms nouns out of verbs which are ending with '-fy'. To “sonify” means therefore to convey the information via sound.

A more detailed definition of Sonification can be split into two important parts: A technique and an intent.

- The technique is the process of mapping numerical data, presumable embodying some relationships in the physical world (in our case space physics) or a model world, to sound.
- The intent is to understand or communicate something about that world.

Both parts of the definition are necessary in order to distinguish the field of Sonification from other fields that involve sound computation.

As a very basic example for Sonification can be seen a Geiger detector which "conveys" information about the level of radiation or even more basically a church bell which "conveys" the current time.

The best known example in space physics was the use of sound for detecting micrometeoroids impacting Voyager 2 when traversing Saturn's rings; these impacts were obscured in the plotted data but were clearly evident as hailstorm sounds[1].

Especially in this example there can be seen the high significance of using audio to display data.

Alone or in combination with visual imaging techniques, Sonification offers a powerful tool of transmitting information. It can improve and increase the bandwidth of the interface "human-computer" and can find a lot of applications in the wide range of information technology.

J. Keller[2] defined in 2003, that Sonification can be categorized in three ways:

- Iconic Sonification: This type of Sonification is when someone maps data to sounds that are associated with certain phenomena. For example, if we gathered weather data, such as cloud cover, temperature, and humidity, to calculate the probability of rain tomorrow, then using the sound of rain to indicate when there is a high probability of rain would be an iconic Sonification.

- Direct Conversion Sonification: This type of Sonification is when someone maps data to sound to listen for patterns that are represented in the data. For example, space scientists map data of waves made up of magnetic and electric fields called electromagnetic waves to sound waves. This direct conversion Sonification can be as simple as taking the frequencies of the waves and making sound waves with the same frequencies, which is most useful as long as the frequencies are at pitches that our ears can hear. Earth's whistler wave is such an electromagnetic wave that scientists have been sonifying for over 30 years.

- Musical Sonification: This type of Sonification is when someone maps data to sound in a musical way. For example, we have created a computer software program that will convert data of very fast particles that have come from the Sun and are captured by an instrument on one of 2 satellites in space, called Helios 1 and Helios 2, to bell-like sounds. Several musicians have used musical Sonification of space data to create quartet or orchestra music pieces.

1 NASA: http://voyager.jpl.nasa.gov
2 J. Keller's Definitions, 2003, Private Communication http://cse.ssl.berkeley.edu/impact/vos/beginners.html

According to Gregory Kramer[2] Sonification is also very often associated with Audification which is not absolutely correct. In comparison to Sonification, Audification is in detail the direct translation of a data waveform into the audible domain for reasons of monitoring and comprehension. Examples for an appliance of Audification can be found on the website of NASA's Voyager project[1].

1.2 The History Of Sonification

The whole field of auditory display and Sonification seems on the first view very young and new but the roots can be found earlier.

The history of auditory representations of data could be said to include actually the research by Pythagoras, Ptolemy, Kepler, Mozart, and Dufay – Sonification in music.

For years data structures have been perceived in sound and these structures have become a basis for musical systems. Predating Pythagoras, who analyzed the structure of harmonics and applied them to musical scales, we see the application of natural law to human-generated sound-producing systems. Pythagoras, referred to his results as “sounding members”. Refinements and extensions of this law defined the development of the musical scales in use all over the world, from the Shakuhachi (bamboo flute) of Japan to the diatonically tuned music synthesizers of global popular music. Manipulating sound for musical ends based upon data or mathematically derived structures arises from a distinguished tradition. Early in the Christian era, the astronomer Ptolemy remarked on the elements of musical modulation and wrote widely studied book on harmonics, as did Kepler and Newton.

1914 the development of the first reading machine with audible output from Fornier D'Albe was designed as an interfaces for blind users and was a next big step in the history of Sonification. The “Optophone” had a six-tone code and was significantly improved in 1922.

If i talk about the history of Sonification I have also to include the exploration of the wide fields of sonar which will not be explained here furthermore.

The first pioneers of Sonification however who started with Sonification the way we understand it nowadays were Pollack and Ficks[3] in 1954. They published a paper detailing research into the use of abstract auditory variables to convey quantitative information. Using tone and noise bursts, they designed a display that presented eight binary variables encoded as the pitch area of the noise, the loudness of the noise, the pitch of the tone, the loudness of the tone, the pitch/noise alternation rate, the temporal ratio of tone to noise, the total duration of the display, and the stereo location of the display. They also created a display without the noise bursts which yielded six binary variables.

Not later than 1961, Speeth[4] reported the results of experiments that used Audification of seismic data to determine if subjects could differentiate earthquakes from underground bomb blasts. Because of their complexity, seismograms that resulted from these events were difficult to understand and categorize. By speeding up the recordings of the seismic data, the complex wave was shifted into the audible range. For over 90% of the trials, subjects were able to correctly classify seismic records as either bomb blasts or earthquakes. Additionally, by speeding up the playback of the data, analysts could review 24 hours of data in about five minutes.

In 1974 three scientists, Chambers, Mathews, and Moore[5] designed a three-dimensional auditory display at AT&T Bell Laboratories. In an auditory enhancement of a scatter plot, they encoded three data variables as pitch, timbre and amplitude modulation. While no formal testing was conducted, they found that the auditory representation did assist in the classification of the data.

1982, chemist Edward Yeung[6] developed a Sonification technique for displaying experimental data from analytical chemistry. Like most researchers in the field of Sonification he looked for auditory variables that had some degree of independence. He selected two pitch ranges, loudness, decay time, stereo location, duration, and silences between events. Using these variables and no more than two training sessions per subject, Yeung asked the subject to classify detected levels of metals in a given sample. A given data point could belong to none of four categories and Yeung's subjects attained a 98% correct classification rate.

Also in the car industries Sonification became an issue. 1986, a team of engineers, working at Fiat Auto[7], developed and patented a Sonification system for continuous monitoring of various automobile parameters. A plurality of sensor devices were used as control signals for a group of tone generators. Once again the task was real-time, not analysis, and psychometric tests were not conducted to determine the efficacy of the display system.

With the enhancement of the computer hardware at the end of the 80's the slow pace of development in Sonification began to accelerate.

Stuart Smith began to work with a team on Exvis[8] at the University of Massachusetts/Lowell. Exvis is a auditory/visual display tool for representing multidimensional (up to seven dimensions) data. The data variables were encoded simultaneously as the geometric attributes of graphic elements and as the attributes of a synthesized sound. The graphic elements produced data-driven visual textures and the auditory display was triggered by moving the mouse cursor through the graphical representation.

At about the same time, Gregory Kramer[9][10] began work at the Santa Fe Institute on Sonification of complex systems and Clarity's Sonification Toolkit. In searching for ways to enable our perceptual systems to more fully contribute to comprehending complexity, Kramer's work, meanwhile, was pushing the limits of dimensionality. Using data supplied by the mathematician Mayer-Kress, Kramer attempted to represent nine-dimensional chaotic systems (ten-dimensions including time) in an auditory display[11]. He also worked with Apple Computer's ACOT group to produce Sonifications of predator/prey models for education purposes, using both realistic and abstract sounds to represent the dynamic system.

Kramer is also one of the founder of the International Community for Auditory Display (ICAD)[12] which coordinates since the foundation in 1992 the research in the field of Sonification. In October 1992, the first International Conference on Auditory Display (ICAD92) convened in Santa Fe, New Mexico under the sponsorship of the Santa Fe Institute. The ICAD92 brought together 36 researchers, nearly all working with issues of how non-speech audio can be used to convey information. Since 1992 the ICAD has been taken place once a year at several places on this globe and many scientists have been contributed to a steadily growing knowledge base for Sonification.

1990, Scaletti and Craig[13], working at the National Center for Supercomputing Applications, produced a series of Sonifications to accompany scientific visualizations developed there. Their work added Sonification to create a sophisticated sonified data visualization. The data represented both aurally and visually included ozone levels, swinging pendula, and forestry data. By displaying these video tapes to the robust computer graphics community, a new and broader audience became aware of Sonification.

At the same time, Rabenhorst[14] was working with some colleagues at IBM's Watson Labs on an auditory and visual representation of three scalar fields associated with electron density, hole density, and potential throughout the volume of a semiconductor. Like the application Exvis, the user could use a mouse to select the region to be displayed. In the IBM work, two volumetric variables were visualized in high resolution while one was sonified.

As current project i would like to introduce the “Sonification Sandbox“[15]. It is a project of the Psychology Department's Sonification laboratory at Georgia Institute of Technology. Bruce Walker, PhD. is the chairman of this project which is motivated by a need for a simple, multi-platform, multipurpose toolkit for sonifying data. This toolkit can map data to multiple auditory parameters and add context using a graphical interface. It supports visual and auditory renderings of the data and the auditory results can also be exported as MIDI files for archiving the results.

1.3 MOTIVATION

General Aspects For Sonification

The human ears provide a very good alternative and supplementation to the visual way of reception information for visualization and understanding complex scientific data. With Sonification it is possible to display effectively large and multidimensional datasets which could help in finding otherwise hidden correlations and patterns. This will provide users with alternative and additional ways of identifying and extracting physical signatures represented in the data, including selection and inter comparison between datasets.

In addition to improving data exploration and analysis for most researchers, the use of sound is beneficial as an supporting technology for visually impaired people and non-visually-oriented people. And as a third big advantage it can also make science and math more exciting especially for young students because people learn in many different ways.

Further examples of the usefulness of Sonification additionally to visualization are:

- uncovering patterns masked in visual displays
- identifying new phenomena current display techniques miss
- improving data exploration of large multi-dimensional and multi-dataset
- exploring in frequency rather than spatial dimensions
- analyzing complex, rapidly, or temporally changing data
- complementing existing visual displays

- monitoring data while looking at something else (background event-finding)
- improving visual perception when accompanied by audio cues

Space Science Specific Aspects For Sonification

Complex datasets (*e.g.,* particle measurements varying in energy, look direction, time, and particle species (such as electrons and ions)) have more parameters to be displayed than there are visual ways to distinguish them and are usually examined in a subset of dimensions at a time, thus forcing the researcher to build up a picture in her mind of the whole dataset. The capability of looking at some dimensions while listening to other dimensions allows one to process more information at once and make better correlations. Alternatively, looking and listening to the same data at one time provides two different views, perhaps exposing patterns hidden in only visual displays. The use of sound will allow identification of otherwise "difficult-to-see" patterns, including dynamic outliers such as leading and lagging indicators. Dynamic outliers differ from the rest of the dataset in their dynamics rather than their actual values and are difficult to discover. Also, auditory accompaniment (e.g. movie music) clearly leads to improved visual information reception.

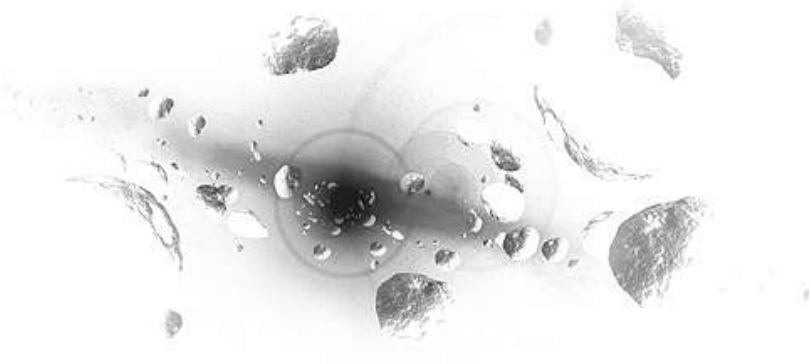
Sonification is in a similar situation to scientific visualization a decade ago but now the progressed computer audio technology makes auditory data representation viable for large number of users. Although there is a rich visualization literature, only a few researchers have published on the use of sound for data exploration and many were limited by the technologies at the time.

Accessibility

Sonification will provide greatly-improved accessibility to space science data for visually-impaired scientists, perhaps even making possible insights not available through visual displays. Current methods for examining data non-visually are inherently inferior and intrinsically more costly. Reading data values is a tough way to analyze data. Raised plots (whether simple time-series "lines" or "maps" to mimic spectrograms) inevitably require very specialized and expensive hardware and are not readily tied to web services. Since most current workstations have sound generation capabilities, Sonification allows effective data browsing for the visually impaired. With Sonification a large fraction of the CDAWeb data collection will be opened to a completely new and now excluded audience. Both, professional and public. In addition to visually-impaired users many people are aurally- rather than visually-oriented and Sonification provides a powerful new tool for all. Sonification also appeals to the educational community, making science more exciting to students and the general public.

Using sound in exploratory data visualization adds to the scientific research capabilities in NASA, especially in complex multi-dimension and multi-dataset research such as for Earth and space sciences. It is cost effective to add Sonification

tools to extract more knowledge from existing and future data sets and missions and new missions will have even larger and more complex datasets to analyze. Supporting and encouraging the visually-impaired and education communities are important NASA goals.



Chapter 2

2. Existing Space Science Applications

The Sonification application xSonify can be used as mentioned in the introduction as an independent stand alone program as well as an additional software module for one of the following described space science applications. How to implement exactly xSonify as an additional module into one of them will be explained in one of the technical chapters later. In this chapter I would like to give an overview of the considered applications.

Before I start with the introduction of the existing space science applications I think it is necessary to familiarize the reader of this thesis a little bit with the field of research

of the Earth's magnetosphere. All the tools are working with the data of this field of space science.

2.1 Magnetosphere Introduction

The magnetosphere is a region around an astronomical object, for instance the Earth's magnetic field. It is confined by the solar wind plasma blowing outward from the Sun. The magnetosphere can extend to distances in excess of 60,000 kilometers from the Earth.

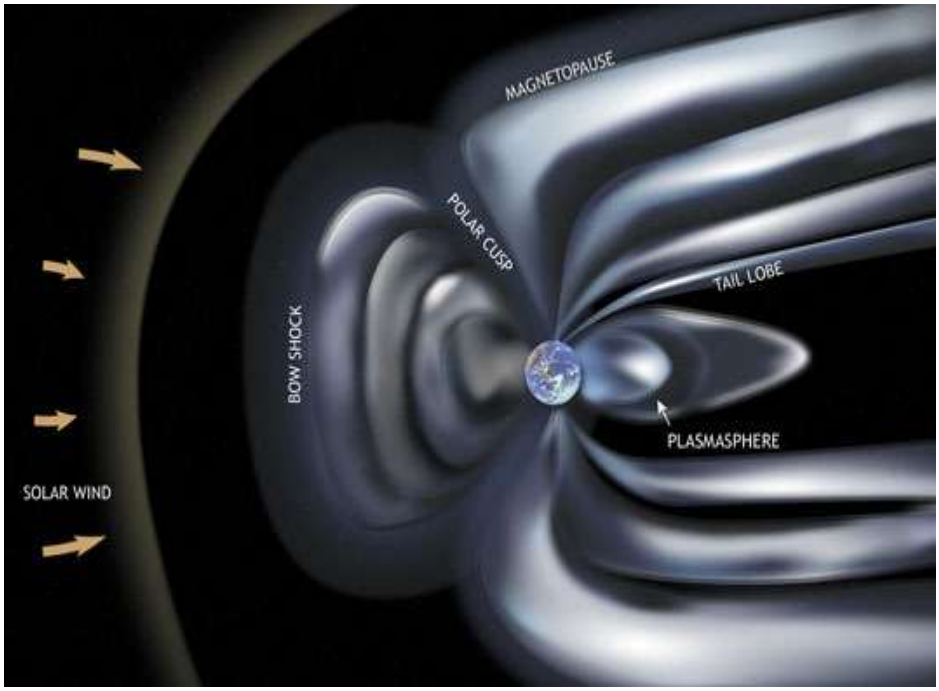


Figure 2.1: Magnetosphere

It is formed from two essential components. One of them is the Earth's magnetic field which is basically generated by currents flowing in the Earth's core. The form of this field outside the Earth has the same form as that of a bar magnet – a dipole field aligned approximately with the Earth's spin axis.

The other component is the solar wind which is a fully ionized hydrogen/helium plasma that streams continuously outward from the Sun into the solar system. This wind is therefore composed of protons and alpha particles, together with electrons.

The Earth's ionosphere is the third component and plays also an important role. The upper atmosphere is partially ionized by far-ultraviolet and X-rays from the Sun above altitudes of about 100 kilometer. The resulting ionosphere forms a second source of plasma for the magnetosphere.

To enhance the perceptions in research of this and other fields of space science, NASA has send out several spacecrafts in Sun's and Earth's orbit collecting data which are processed and archived in the NSSDC[3].

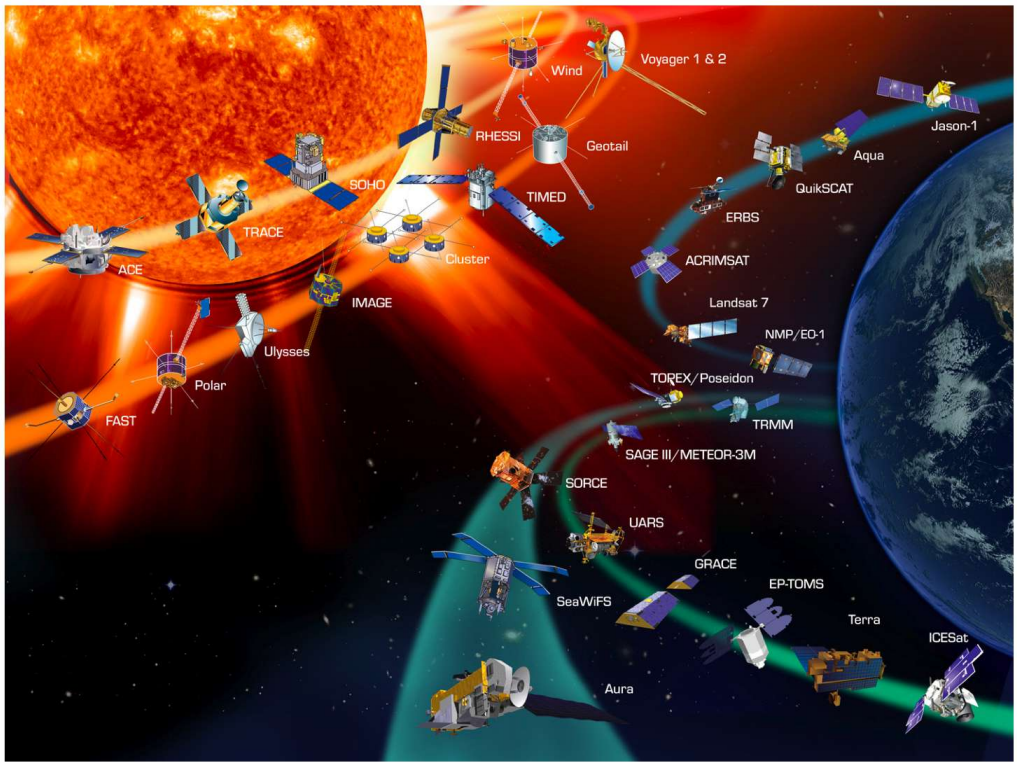


Figure 2.2: Spacecrafts In Orbit

These data are the base of modern research and can be accessed for example via the CDAWeb[4] which offers the scientists Java and web interfaces to access the data in a database.

3 NSSDC: National Space Science Data Center, NASA GSFC
http://nssdc.gsfc.nasa.gov/

4 CDAWeb: Coordinated Data Analysis Web, NASA GSFC
http://cdaweb.gsfc.nasa.gov/

2.2 TIPSOD

As the name TIPSOD[5] (Tool for Interactive Plotting, Sonification, and 3D Orbit Display) already describes, this software application is designed for interactive, animated, 4D (3D + time) visualization of satellite orbits.

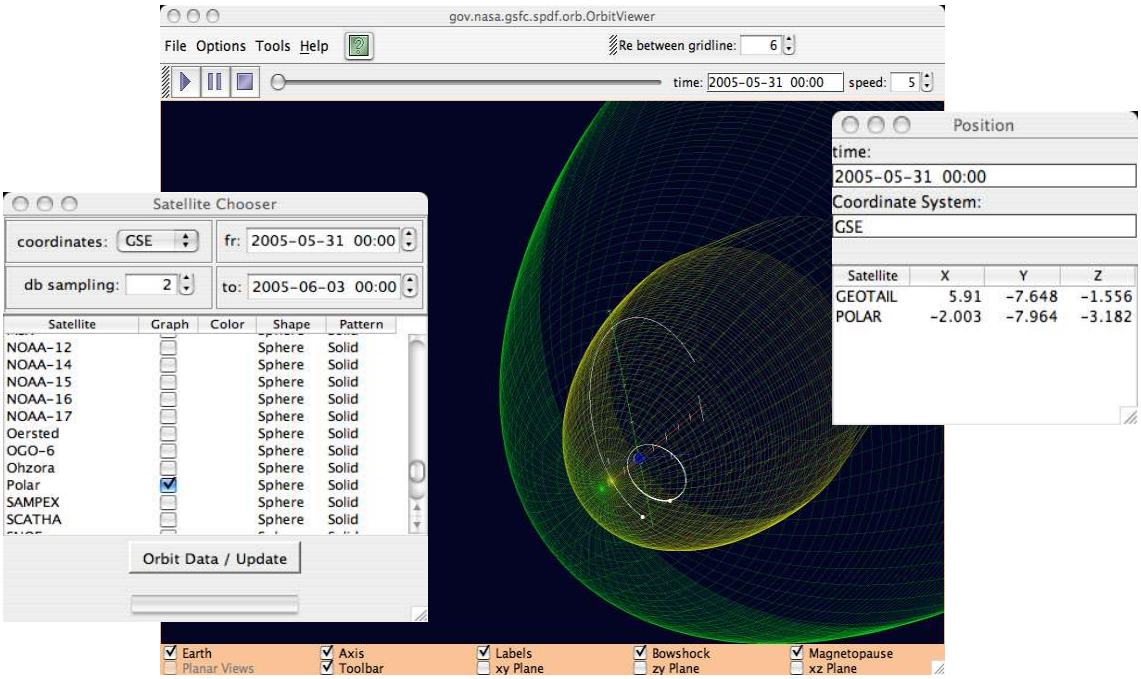


Figure 2.3: TIPSOD

It utilizes the SSCWeb[6] services programming interface to communicate with SSC logic and database over the open protocols of the Internet. TIPSOD is implemented in Java 3D and makes it possible to display satellite orbits. In addition to satellite orbits,

5 http://sscweb.gsfc.nasa.gov/tipsod/

6 SSC: Satellite Situation Center
http://sscweb.gsfc.nasa.gov

the software computes and displays the Sibeck's magnetopause and Fairfield's Bow Shock surfaces. The displays are time-dependent through user activity. The program is used as a projection or interpretation tool by the scientific community.

The requested spacecraft(s) can be chosen in the “Satellite Chooser” window. Additionally to the selection of spacecraft(s) this window can also be used to specify the search by changing parameters like the time range and to change the display attributes of the satellites like the shape.

Another window called “Position” display the current position of the spacecraft dependent on the type of the selected coordinate system.

2.3 VISBARD

ViSBARD[7] displays data in three dimensions along the orbits which may be displayed either as connected lines or as points. The application provides a way of visualizing multiple vector and scalar quantities as measured by many spacecraft at once.

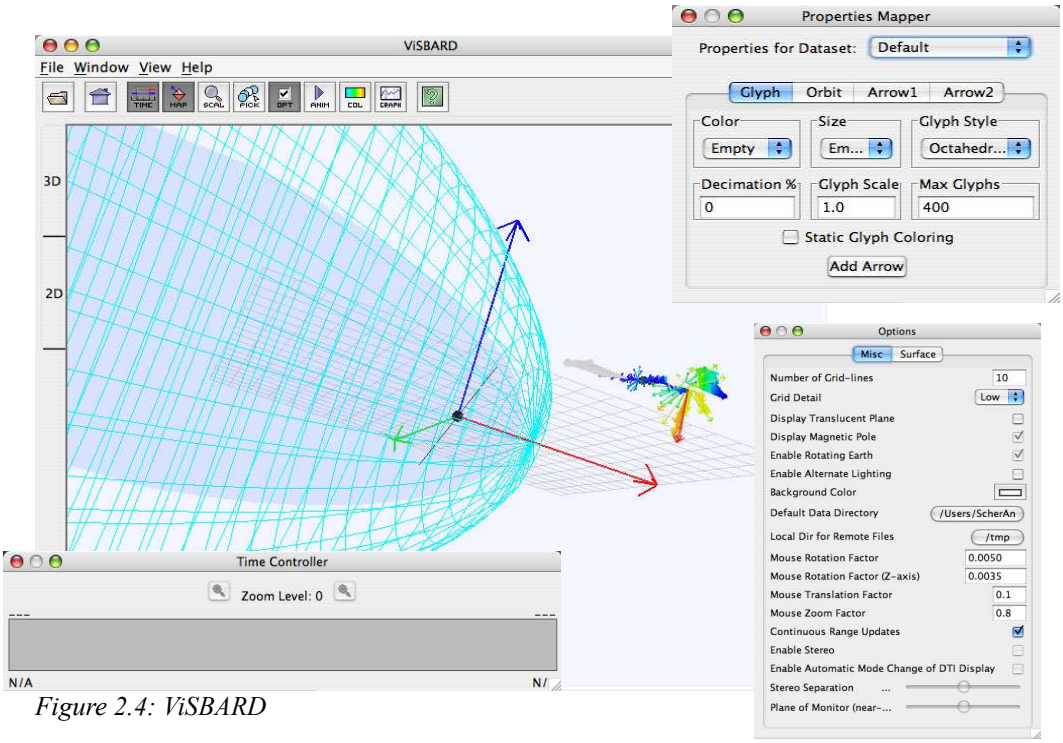


Figure 2.4: ViSBARD

The data display allows the rapid determination of vector configurations, correlations between many measurements at multiple points, and global relationships. Things such as vector field rotations and dozens of simultaneous variables are very difficult to see in panel plot representations. The data are displayed 3D along the orbits which may be displayed either as connected lines or as points. ViSBARD is linked to the NSSDC's CDAWeb repository via a SOAP interface for direct access to space physics data.

In future it will be linked to a Virtual Space Physics Observatory to allow direct access to a wide variety of datasets. The data may be read into ViSBARD as file formats like ASCII or CDF.

The application is platform independent since it is written in Java. It also supports stereoscopic hardware for 3D viewing.

---

7 ViSBARD: Visual System for Browsing, Analysis, and Retrieval of Data
http://windsor.gsfc.nasa.gov/selected_software/visbard/

The dialog “Resource Toolkit” is the data import module of ViSBARD to retrieve the data from a local file or a SOAP connection via the Internet. I would like to emphasize this part of the ViSBARD application because it found its reincarnation in xSonify.

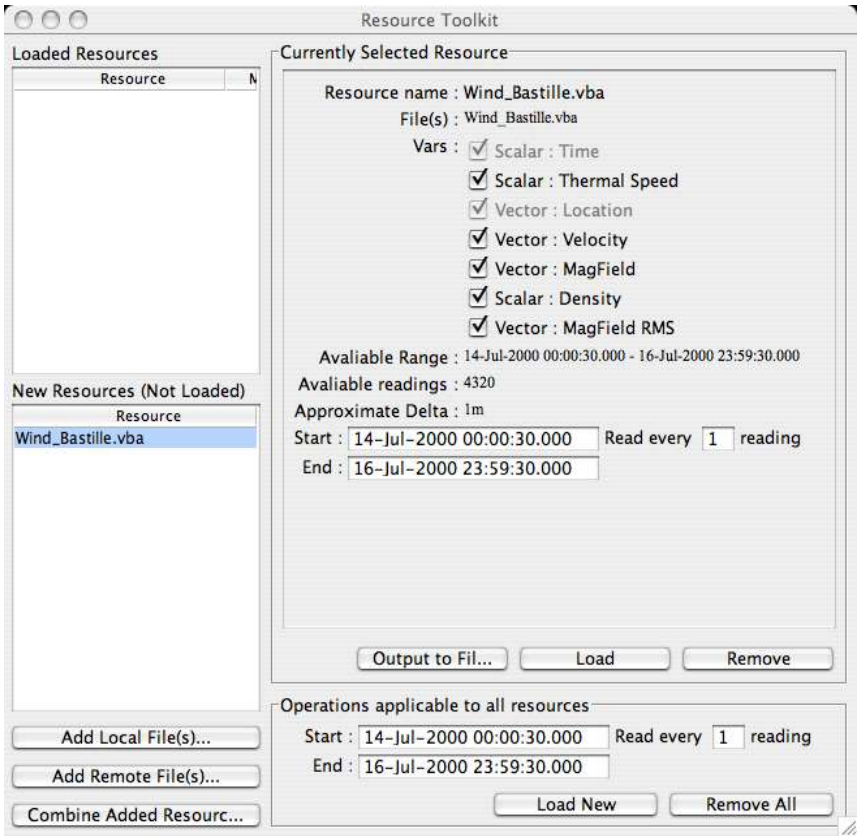


Figure 2.5: ViSBARD's Resource Toolkit

As soon as the file(s) are loaded either via the Internet or from a local file the header information of the file(s) is displayed in the “Currently Selected Resource” area. Further specifications of the desired spacecraft data can be made like the time range and the variables. Finally the data can be retrieved by clicking the “Load” button.

2.4 CDA Web +

CDAWeb Plus is a Java based interface for integrated access to all existing SPDF[8] services and public data including CDAWeb itself, SSCWeb, OMNIWeb, COHOWEB, ATMOWeb, ModelWeb as well as the file-level holdings on the NSSDC and selected other FTP sites.

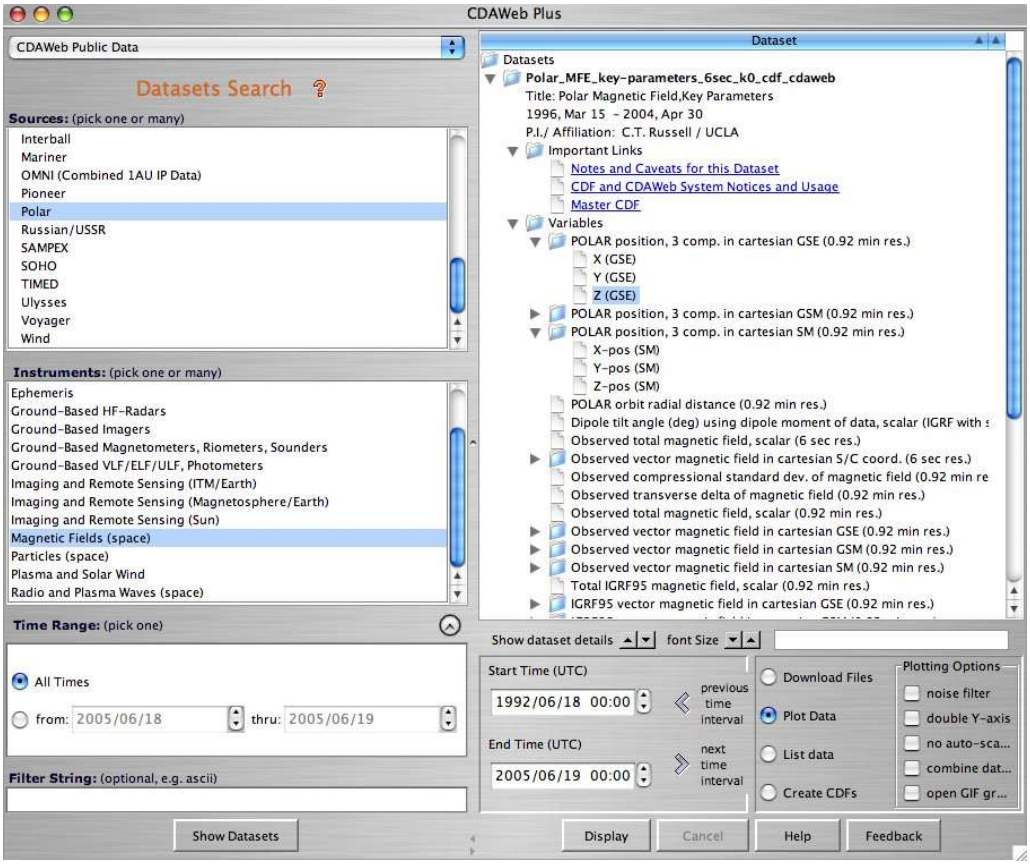
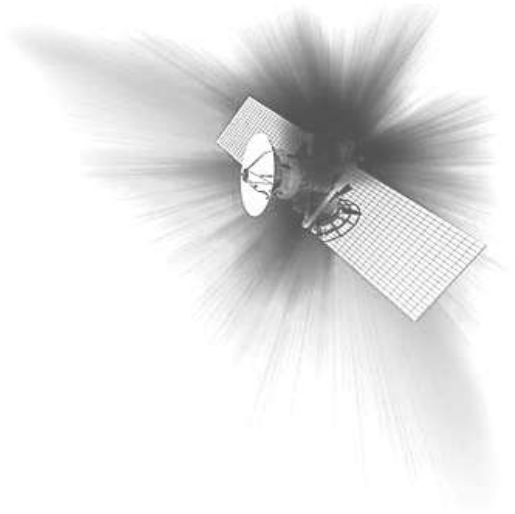


Figure 2.6: CDAWeb Plus

In the “Sources” area the spacecrafts of interest can be selected. As second step every spacecraft has a variety of instruments which can be chosen in the “Instruments” area. After the time range is specified and the button “Show Datasets” was clicked the “Dataset” panel lists the results for the previous selections. The results are a list of datasets, beginning with the name and additional information like the “Date” and “Author” and furthermore “Important Links”. It also lists the “Variables” for each result dataset if they are available. This depends on the previous selection of spacecraft, instrument and time range. The “Variables” can be selected and the four options in the lower right panel can be applied on it. They can be downloaded as files if the option “Download Files” was selected and appear after the download as CDF files. The option “Plot Data” opens after the selection an additional window or

8 SPDF: Space Physics Data Facility, NASA GSFC
http://spdf.gsfc.nasa.gov/

dependent on the chosen option a HTML browser window instead and displays the data as a plot within the created window. “List Data” creates a list of the chosen variables within a new created window with all the detailed header information at top. The last option “Create CDFs” out of the variables opens a dialog box with the request for downloading the created CDF file.



Chapter 3

3. Employed Technologies And Standards

This chapter should provide the reader with some fundamental knowledge of technologies and standards which are used in xSonify. It starts with the motivation of why I chose Java 1.5.0 and will be followed immediately by the introduction of the Java Sound API. The whole xSonify application should be executable via the Internet. The tool which supports us in this issue is the Web Start technology from Sun. The application will retrieve the data via the Internet as well and needs therefor a reliable way to achieve this. SOAP will cover this task and will be explained in detail in chapter 3.4. MIDI is the standard which will be used for the sound generation and handling. Since chapter 3.2 covers mainly the Java Sound API in chapter 3.5 MIDI will be introduced more in detail. To finish this chapter we will have a deeper look in the Common Data Format in which most of the space science data are handled.

3.1 Java 1.5.0

For xSonify the main reason for choosing Java Version 1.5.0 was mainly the enhanced Java Sound API beside all the the other goodies which came with Java Tiger.

- Ports are now available on all platforms
- MIDI device i/o is now available on all platforms
- Optimized direct audio access is implemented on all platforms.
- The new real-time Sequencer works with all MIDI devices and allows unlimited Transmitters
- The `sound.properties` configuration file allows choice of default devices
- MidiDevices can query connected Receivers and Transmitters

- The Sequencer interface is extended with loop methods for seamless looping of specific portions of a MIDI sequence
- Java Sound no longer prevents the VM from exiting

3.2 Java Sound API

3.2.1 Java Sound History

A long time ago before MP3 & Co. was invented when Sun Java Version 1.02 was state of the art, the Java technology only had the capability to play simple AU format sounds with a sampling rate of 8 kHZ. In these days the Java applet demos still came with such classic hits like spacemusic.au and yahoo.au.

In Sun Java 2 Version 1.2, Sun Microsystems improved the quality of Java audio by implementing the Headspace Audio Engine by Beatnik Corporation[9]. Java programmers could now use the same audio interfaces but with the additional capability of playing more formats like AU, WAV, AIFF, MIDI, and RMF sounds. Although the sound quality was improved to CD audio levels which has a sampling rate of 44 kHz. But there still was no programmatic way to pause and resume a sound, display a progress bar, or get a notification that your sound was completed.

Only with Sun Java 2 Version 1.3 the Java Sound API introduced many new capabilities for the audio software developers including pause and resume, progress bars, and sound completion events. Java Sound offered also the software mixer which could mix up to 64 channels of sampled or synthesized audio. The MIDI synthesizer supports since then wave table synthesis that programmers can access by loading the programmable sound bank. The API has also an interface to record and save sampled or MIDI files. With this step in the development of the Java Sound API it has become more mature and therefore more attractive for sound applications developers.

Unfortunately, until today with Sun Java 2 Version 1.5 the Java Sound engine can not take advantage of a very enhanced audio board since the audio hardware acceleration is limited. The synthesis and mixing are software based, so playing MIDI audio with Java technology will have more of an effect on your CPU usage than if you play the MIDI file with a native audio program. This is similar to how Java 2D performs great graphic manipulations but does not take advantage of a hardware accelerated video board. By doing the work in software, Java technology gives you cross-platform portability but at the expense of high performance and low CPU utilization.

9 http://www.beatnik.com/

3.2.2 Class Overview

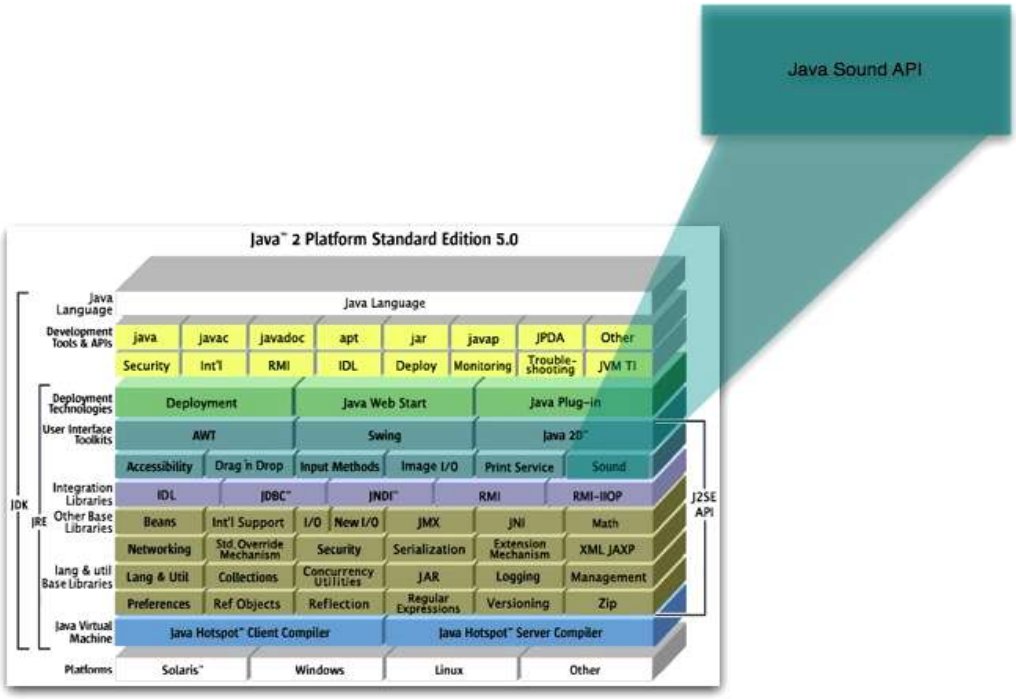


Figure 3.1: Java 2 Components

The Java Sound API[10] includes support for both digital audio and MIDI data. These two major modules of functionality are provided in separate packages:

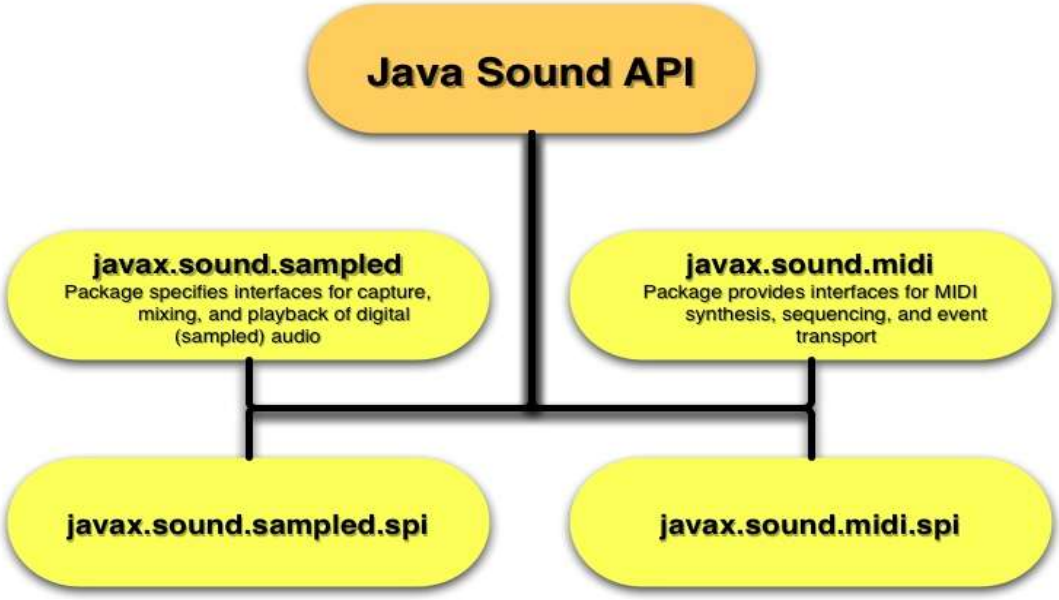


Figure 3.2: Java Sound API

10 http://java.sun.com/j2se/1.5.0/docs/guide/sound

In the left area of figure 3.2 you can see the two “sampled” packages which are handling digital audio. Java Sound API refers to it as sampled audio. Samples are successive snapshots of a analog signal. I will not explain the use of this package further since it is not used in this software application. In the right area there are the two “midi” packages for MIDI synthesis, sequencing and event transport.

The two lower packages “spi” in figure 3.2 permit service providers to create custom software components that extend the capabilities of an implementation of the Java Sound API. “spi” stands for service provider interface.

The Java Sound API does not assume a specific audio hardware configuration; it is designed to allow different sorts of audio components to be installed on a system and accessed by the API. The Java Sound API supports common functionality such as input and output from a sound card (e.g. for recording and playback of sound files) as well as mixing of multiple streams of audio.

3.3 Java WebStart

3.3.1 Introduction Of Java Web Start

Java's Web Start is actually the reference to the implementation of the specification of JNLP[11] which stands for Java Network Launching Protocol and API. The JNLP can be defined as a protocol that enables Java clients to deploy themselves on the client and run as if they were local applications. This reminds the Java developer at the beginning probably first of what Java introduced earlier as applet.
Java applets have basically a big size and are slow in execution. Users generally are not patient enough to wait for an applet to load from the network, check security permissions and then see the information. If an applet comes up with swing components the wait is doubled. JNLP has overcome many of the difficulties of applets.
Java Web Start is a JNLP client that allows Java applications with a single click to be downloaded onto the client machine and runs them in the “Java security Sandbox”. Java Web Start is therefor a reference implementation provided by Sun to show the features of JNLP which allows a developer to create Java applications that can be run on a client without any installation procedures.

Java web start brings in all the security of the “Java sandbox” which means that you can run an application with the confidence that it will not over step and meddle with files on the client machine. It also brings the flexibility of using Java applications with swing or AWT without the hassle of downloading huge swing jar files each time the application is run. Java web start is designed to download all the files required the first time it is invoked from a Web page. After the first download the user has even the option to include a shortcut to the application directly on the windows start menu if he uses Microsoft Windows. Subsequent runs of the application can be done using the shortcut which would then run locally. The application can be designed to check for updates on the server to refresh the existing local copy with the changes made since the last access.

11 http://java.sun.com/developer/technicalArticles/Programming/jnlp/

The important features[12] of JNLP are:

- It is a Web-based Application Architecture where applications can run locally using resources spread over the web.
- It provides for an installation free client application that can keep itself in sync with the updates on the server.
- It also provides for a facility to make incremental downloads and updates over a period of time.
- It offers the flexibility to run Java applications on different versions of JRE. The JRE can be can be downloaded if the version is not present on the machine.
- It offers a caching facility which can cache the application locally on the client which saves time the next time the application is run on the client.
- Applications can be run offline on the client machine after they are initially downloaded thereby decoupling them from the server.
- It offers a secure environment to execute applications like applets but it also provides flexibility within the API to do potentially insecure actions with a warning to the user.

The JNLP API and Java Web Start have been part of the J2SE since version 1.4.

3.3.2 JNLP Technology[13]

JNLP is a XML based specification. The heart of JNLP technology is a JNLP file. It is a XML file that describes the different attributes used to describe the application. Shown below is a JNLP file with a few attributes:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"codebase="http://server.com">
  <information>
    <title>Title</title>
    <vendor>MY company</vendor>
   <description>Demo Application</description>
   <icon href="icon.jpg"/>
   <offline-allowed/>
  </information>
  <security>
      <all-permissions/>
  </security>
  <resources>
    <j2se version="1.3"/>
    <jar href="lib/SwingSet2.jar"/>
  </resources>
  <application-desc></application-desc>
</jnlp>
```

The *jnlp* element is the root element that has a set of attributes that are used to specify information that is specific to the JNLP file. The *information* element describes meta-information about the application like title, description, vendor etc. The *security* element is used to request a trusted application environment, why applications need to

12 http://javaboutique.internet.com/tutorials/WebStart/
13 http://javaboutique.internet.com/tutorials/WebStart/

be trusted will be discussed later in the security section.

The *resources* element specifies all of the resources that are part of the application, such as Java class files, native libraries, and system properties. The last part of the JNLP file defines the kind of application. It could be one of the following four options: *application-desc, applet-desc, component-desc, or installer-desc*.

If an application-desc is defined in a JNLP file then it's an application descriptor. The application-desc element describes the application and the attributes required to invoke it.

```
<application-desc main-class ="SampleApplicationClass">

<agrument> arg1</agrument>

</application-desc>
```

The main-class is the Java class file which contains the main method that needs to be run. Any arguments that might be required may be passed in the argument element.

When the applet-desc is defined in the JNLP file then it's an application descriptor for an applet. The applet-desc tag takes the normal parameters an applet would take.

Similarly when a Component-desc is defined in the JNLP file it represents a component extension. A Component extension is used to represent common components that can be shared between applications.

The last type is a installation file. Its called a installer extension and it contains a installer-desc tag. This is used when a application needs to be downloaded and installed for the first time.

3.3.2.2 Step By Step

- **Step 1: Build your application**
  Your application must be available as a jar file.

- **Step 2: Sign the jar file**
  You must sign the jar file in order that people can verify its origin (and decide if they trust the application or not). The JDK contains a tool which allows signing of jar file with a certificate. If you don't have your own certificate, you can use the tool also to build one.

  Create new certificate:

  ```
  keytool -genkey -keystore yourKeystore -alias YourName
  ```

  You will be prompted several questions. At the end, your personal certificate will be in your keystore "yourKeystore". You can check it by calling:

  ```
  keytool -list -keystore yourKeystore
  ```

-

Now you can sign the jar file:

```
jarsigner -keystore yourKeystore test.jar YourName
```

The jar file includes know your signature and people can decide if they trust it.

- **Step 3: Creating the JNLP file**
  The JNLP file describes mainly which file(s) are included in the application (only one jar file in our case), which is the main class, which JRE version to use and the network location. The JRE version is used for update checks.

Here is an example file called webstart.jnlp file:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://www.autexier.de/jmau/dev/webstart"
href="webstart.jnlp">
  <information>
    <title>WebStart Demo</title>
    <vendor>Jean-Marc Autexier</vendor>
    <homepage href="http://www.autexier.de/jmau" />
    <description>A Java Webstart test</description>
    <offline-allowed />
  </information>
  <resources>
    <j2se version="1.4+" />
    <jar href="JnlpTest.jar" />
  </resources>
  <security>
    <all-permissions />
  </security>
  <application-desc main-class="jnlptest.Main" />
</jnlp>
```

- **Step 4: Web server installation**
  Both the jar and the JNLP file must be available on a HTTP server. The web server must return a special MIME type for JNLP files:

  ```
  application/x-java-jnlp-file
  ```

  If you have an apache webserver, the easiest way is to place a .htaccess file in the same directory than the application.

  ```
  AddType application/x-java-jnlp-file    .jnlp
  AddType application/x-java-archive-diff .jardiff
  ```

- **Step 5: Test**
  Now when you call the Url of the JNLP file and your browser is configured correctly, Java Webstart client will open, download the application, ask for security reasons if you trust the certificate owner and execute it.

3.3.2.3 Updates And Caching[14]

The JNLP application provides three different download protocols by which the application on the client can be made current. The first is the Basic download Protocol which downloads resources without any version information. The second is a Version based Download protocol which identifies all resources by a URL and version Id. In this case when the JNLP client starts up an application it sends the current version to the server as part of the request. If the server has a newer version it would download the newer version. The third protocol pertains to extensions where a URL or a URL and version id can be specified to download the Extension Descriptor. If only a URL is specified the extension is downloaded using the Basic download protocol. If a URL and a version is specified the version based download protocol is used with a few additional parameters. The extra parameters are used to identify the extension type and the platform for which it is needed.

JNLP also provides a facility of providing incremental updates. When the server finds that the client already has a version on the local machine and all it requires is a new version it sends an incremental upgrade instead of the whole application thereby reducing download time.

A JNLP client can cache the application to make it run faster in the subsequent runs. If an application is downloaded using the basic download protocol it would download the application without a version. When the application is downloaded a time stamp is downloaded on the client to keep track of updates. In the version based download protocol timestamps are not stored but the version would be part of the request.

When the request is made the JNLP client checks for the version already existing in the cache and the version in the request. If they match no download needs to be done.

3.3.2.4 Security

The Java Web Start enforces the strict security rules of the Java language. Like Java applets all Java Web Start applications execute within the Java Sandbox. By default all applications are deemed to be malicious and access to local resources is restricted. However applications can be signed using security certificates to allow limited access to system resources and files. This may not be of great benefit to Intranet users who want to capitalize on the power of Java Webstart in a secure intranet. The JNLP API provides for some basic operations which can be done without securing or signing the application. Some of them are discussed in the next section.

3.3.3 JNLP API

Sun has included the Java Web Start in the download for J2SE 1.4. It has introduced a new extension package to include all the JNLP specific files called the javax.jnlp.*.

The JNLP API contains a few important services that would be helpful for applications that would want to do some operations on the client that are not allowed by the security manager. This does not mean that the security manager is bypassed by

14 http://javaboutique.internet.com/tutorials/WebStart/

the JNLP API. It provides for a platform independent mechanism to interact with the client resources after obtaining the permission of the client. Java Web Start displays a warning window when a operation outside the control of the sandbox is requested.

An object of the service required can be obtained using the static lookup method of the ServiceManager Class. The lookup method takes a String parameter which is the string representation of the service name and returns a handle to the service requested.

3.3.4 JNLP's Main Services

BasicService
This is a mandatory service and does operations similar to the AppletContext from the Applet class. The getCodeBase method provides access to the codebase of the application. The isOffline method which can be used to determine if the application is running offline or online. Finally the isWebBrowserSupported method can help in finding the browsers supported by the JNLP client. The class is *javax.jnlp.BasicService*

DownloadService
This is a mandatory service that needs to be provided. It allows the application to control the resources being downloaded and cached on the client. It can check for already existing resources, load new resources, force caching of resources and remove resources from the client machine. Only resources available for the application can be downloaded. They need to be specified in the JNLP file. The class is *javax.jnlp.DownloadService*.

FileOpenService
This service provides the flexibility to access the files on the client machine even if they are in a untrusted environment. The openFileDialog or openMultiFileDialog can be used to access the file required. The contents of the file are returned in a FileContents object which provides access to the contents of the file. This service cannot be used to find the directory structure of the client machine. The JNLP client needs to show the security dialog box to warn the user of the operation. The class is *javax.jnlp.FileOpenService*.

FileSaveService
This service provides a mechanism for storing files on the client machine, even if they are in a untrusted environment. The file save dialog box is displayed by invoking the saveFileDialog or saveAsFileDialog methods. This method returns a FileContents object representing the file that was saved. The class is *javax.jnlp.FileSaveService*.

ClipboardService
This service provides an interface to access the contents of the Clipboard even when running in a untrusted environment. It consists of two methods, setContents and getContents, that help set or retrieve the contents of the clipboard. The service has to warn the user of the potential dangers of letting a untrusted application access clipboard data. The class is *javax.jnlp.ClipboardService*.

PrintService
This service provides access to printing from an untrusted application. The application

submits a request to the JNLP client for a print job which is in turn passed to the client machine for permission. If accepted the print job is executed. The class is *javax.jnlp.PrintService*.

PersistanceService
This service provides a mechanism to store data on the client side even in a untrusted environment. The service is similar to the service provided by the cookies in HTML. Unlike cookies there is no maximum limit on the data that can be stored on the client. The maximum limit is decided by the JNLP client implementing this service. The class is *javax.jnlp.PersistanceService*.

ExtensionInstallerService
This is a mandatory service that provides methods to provide an extension installer to manipulate the progress bar shown during installation. It provides methods like updateProgress and hideProgressBar to update the progress bar during installation. It also provides information on native libraries. The class is *javax.jnlp.ExtensionInstallerService*.

3.4 SOAP

3.4.1 Introduction

SOAP stands for Simple Object Access Protocol and is a communication protocol for the communication between applications. It can be also considered as a format for sending messages and is designed to communicate via the Internet. SOAP is a XML based protocol and therefor platform and programing language independent for the information exchange in a decentralized, distributed environment. SOAP consists of three parts:

- an envelope that defines a framework for describing what is in a message and how to process it
- a set of encoding rules for expressing instances of application-defined datatypes
- a convention for representing remote procedure calls and responses

The Internet has become one of the most important medium for the worldwide information exchange and it is meanwhile for the development of applications very important to allow Internet communication between programs. Today's application communicate using Remote Procedure Calls between objects like DCOM and CORBA. HTTP however was not designed to support these services. Remote Procedure Call represents a compatibility and security problem. The proxy and firewall servers will normally block this kind of traffic. Therefor it was necessary to develop a way to communicate between applications via HTTP. A consortium of the big companies like UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft, and SAP proposed to W3C, in May 2000, the SOAP Internet protocol. December 2001 the first public working draft on SOAP was published from the W3C. The latest version of SOAP which is SOAP V1.2[15] was

15 http://www.w3.org/TR/soap12-part1/

recommended at June 24 2003 by the W3C. At present, SOAP has been implemented in over 60 languages on over 20 platforms.

3.4.2 SOAP Message

A SOAP message is an ordinary XML document containing the following elements:

- A required *Envelope* element that identifies the XML document as a SOAP message
- An optional *Header* element that contains header information
- A required *Body* element that contains call and response information
- An optional *Fault* element that provides information about errors that occurred while processing the message

All the elements above are declared in http://www.w3.org/2001/12/soap-envelope

The SOAP message exchange process[16]:

1. The client application builds a SOAP message which is an XML document. It can now perform the desired request/response operation.
2. The client sends the SOAP message to a JSP, PHP or ASP page on a Web server listening for SOAP requests.
3. The SOAP server parses the SOAP package and invokes the appropriate method of the object in its domain, passing in the parameters included in the SOAP document. Optionally, intermediate processing nodes may have performed special functions as indicated by SOAP headers prior to receipt of the message by the SOAP server.
4. The request object performs the indicated function and returns data to the SOAP server, which packages the response in a SOAP envelope. The server wraps the SOAP envelope in a response object, such as a servlet or a COM object, which is sent back to the requesting machine.
5. The client receives the object, strips off the SOAP envelope and sends the response document to the program originally requesting it, completing the request/response cycle.

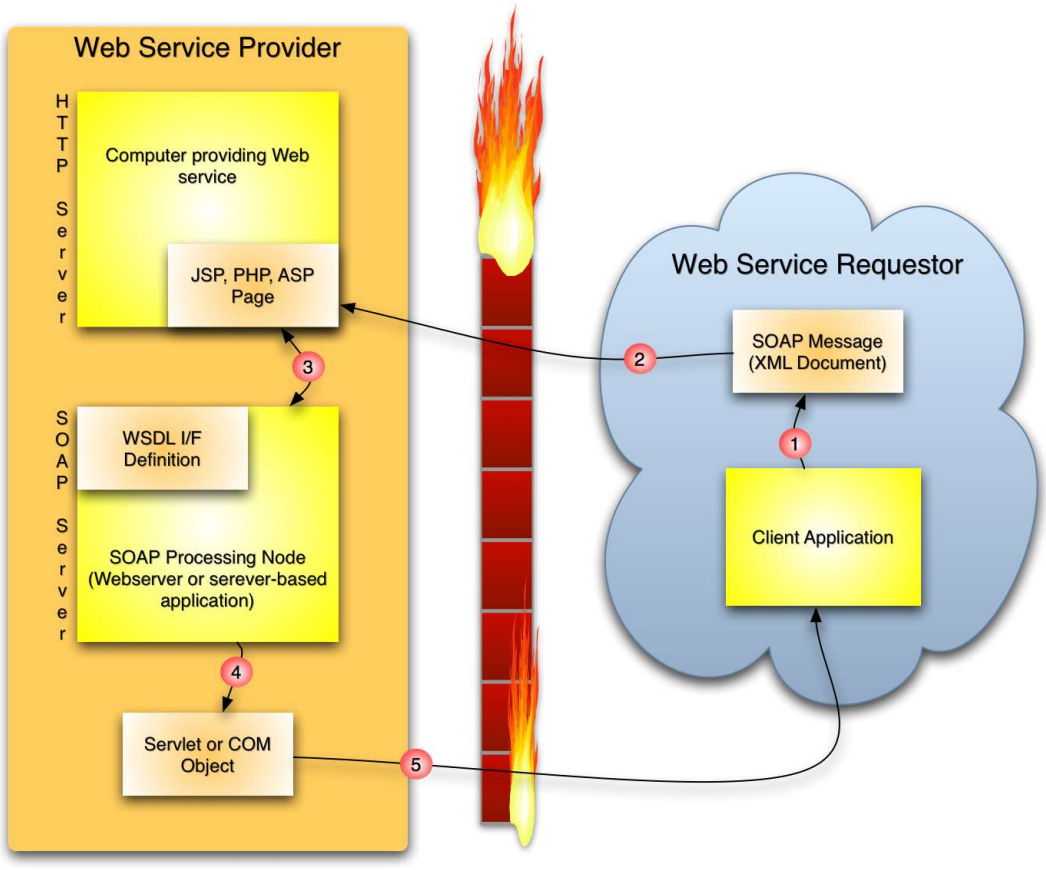


Figure 3.3: SOAP Message Exchange Process

16 http://java.sun.com/developer/technicalArticles/xml/webservices/

3.4.3 SOAP Envelope[17]

```
<SOAP-ENV: Envelope
  xmlns:
    SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
        <t:Transaction xmlns:t="some-URI">
                SOAP-ENV:mustUnderstand="1"
                    5
        </t:Transaction>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="some-URI">
                <symbol>DEF</Symbol>
        </m: GetLastTradePrice>
      </SOAP-ENV:Body>
</SOAP-Envelope>
```

In this example a `GetLastTradePrice` request is being sent to a stock-quote service somewhere on the Web. The request takes a string parameter, a ticker symbol, and returns a float in the SOAP response.

The SOAP envelope is the top element of the XML document that represents the SOAP message. XML namespaces are used to disambiguate SOAP identifiers from application specific identifiers. XML namespaces are used heavily in SOAP to qualify or scope elements in the message to a specific domain. To understand SOAP namespaces, it helps to be familiar with the namespace spec for XML. If you're not, simply think of namespaces as neighborhood identifiers that help uniquely identify SOAP elements by associating them with specific locations, real or imagined.

Namespaces

The first namespace in the example references the SOAP schema which defines the elements and attributes in the SOAP message. The second namespace refers to SOAP encodings, the "Section 5" data types discussed earlier. Since no additional per-element encoding is specified, this encoding applies to the whole document.

Header

The first element identified in this sample SOAP envelope header is a transaction element, accompanied by a namespace attribute and by the `mustUnderstand` attribute with a value of `1`. Since `mustUnderstand` is set to `1`, the server accepting this message must perform intermediate processing on this transaction node. You can interpret this to mean that the server and client have previously agreed upon the semantics that govern the processing of this header element, so that the server knows exactly what to do with the contents of the element, in this case `5`.

If the server receiving this message doesn't understand the semantics of the transaction header, it is required to reject the request completely and throw a fault. A fault element is a special part of the SOAP body and a well-defined mechanism to ship error information back to the client.

Intermediate processing nodes like this are an example of SOAP's extensibility. Clients include such nodes in a SOAP message to indicate that special processing

17 Example from: http://java.sun.com/developer/technicalArticles/xml/webservices/

needs to take place before the contents of the message body can be processed. Ensuring backward compatibility with existing servers not capable of providing such processing is simply a matter of setting the `mustUnderstand` attribute to `0` which makes the action optional.

In addition to defining transaction nodes like the one described above, a SOAP message may optionally contain header entries specifying nodes that perform authorization processing, encryption, persistence of state, business logic processing and so on. Headers help make SOAP a modular, extensible packaging model. Just keep in mind that header processing is entirely independent of the SOAP message body.

Body

The SOAP body in the example contains an XML payload, which we can surmise, without really seeing it spelled out for us, does RPC. SOAP is not only a modular packaging model, it's also a fairly cryptic packaging model.

Nothing here explicitly shows that RPC is begin done. All we see in the body are a couple of XML elements, one qualified by a namespace. It's up to the SOAP server to understand the document semantics and do the right thing. The server, in effect, provides a framework for dealing with the XML payload in a meaningful way. "Meaningful" here implies that the server invokes a remote procedure call on some back-end database to receive the stock price for the stock-symbol element contained in the message body. All the magic takes place behind the SOAP Remote Procedure Call curtain.

3.4.4 SOAP Remote Procedure Call[18]

SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but SOAP messages are often combined to implement request/response mechanisms. To do RPC using SOAP, a few conventions must be followed. First of all, request and response messages must be encoded as structures. For each input parameter of an operation, there must be an element (or member of the input structure) with the same name as the parameter. And for every output parameter, there must be an element (or member of the output structure) with a matching name.

Here's a foreshortened, Remote Procedure Call based view of the SOAP message presented earlier. Only the body portions of the SOAP request and response envelopes are shown.

Request

```
<SOAP-ENV:Body>
     <m:GetLastTradePrice xmlns:m="some-URI">
          <symbol>DEF</Symbol>
     </m:GetLastTradePrice>
</SOAP-ENV:Body>
```

18 Example from: http://java.sun.com/developer/technicalArticles/xml/webservices/

Response

```
<SOAP-ENV:Body>
     <m:GetLastTradePriceResponse xmlns:m="some-URI">
          <price>22.50</price>
     </m: GetLastTradePriceResponse>
</SOAP-ENV:Body>
```

The request invokes the `GetLastTradePrice` method. Notice the response defines a `GetLastTradePriceResponse` operation. A convention common to SOAP calls for appending `Response` to the end of a `Request` operation to create a `Response` structure. This output structure contains an element called price which returns the results of the method invocation presumably as a float.

It's important to note that nowhere in the SOAP envelope are data types explicitly delineated, so we really don't know the type of the symbol or the type of the result parameter price just by looking at the SOAP message. Client applications define data types either generically through "Section 5" encodings, or privately via agreed-upon contracts with servers. In either case, these definitions are not explicitly included in the SOAP message.

Finally, in order to do Remote Procedure Call, a lower-level protocol like HTTP is needed. Although the SOAP 1.0 specification mandated the use of HTTP as the transport protocol, SOAP 1.1 1 (and its sister specification "SOAP Message with Attachments") permit the use of FTP, SMTP or even (possibly) raw TCP/IP sockets. All the serialization and encoding rules general to SOAP apply to RPC parameters as well.

3.4.5 SOAP Example

In the example below a `GetStockPrice` request is sent to a server. The request has a `StockName` parameter and a `Price` parameter will be returned in the response. The namespace for the function is defined in "http://www.stock.org/stock" address.

The SOAP request:

```
POST /InStock HTTP/1.1

Host: www.stock.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

A SOAP response:

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

3.5 MIDI

Unlike sampled audio MIDI is an indirect representation of a sound itself. MIDI data can be imagined like a recipe for creating a sound especially a musical sound. It describes events that affect the sound a synthesizer is making. MIDI data is analogous to a graphical user interface's keyboard and mouse events.

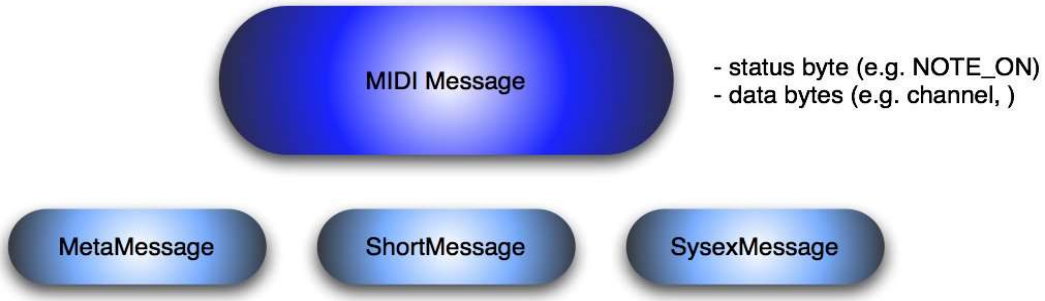


Figure 3.5: MIDI Message

In the case of MIDI the events can be thought of as actions upon a musical keyboard along with actions on various pedals, sliders, switches, and knobs on that musical instrument. These events need not actually originate with a hardware musical instrument. They can be simulated in software and they can be stored in MIDI files.

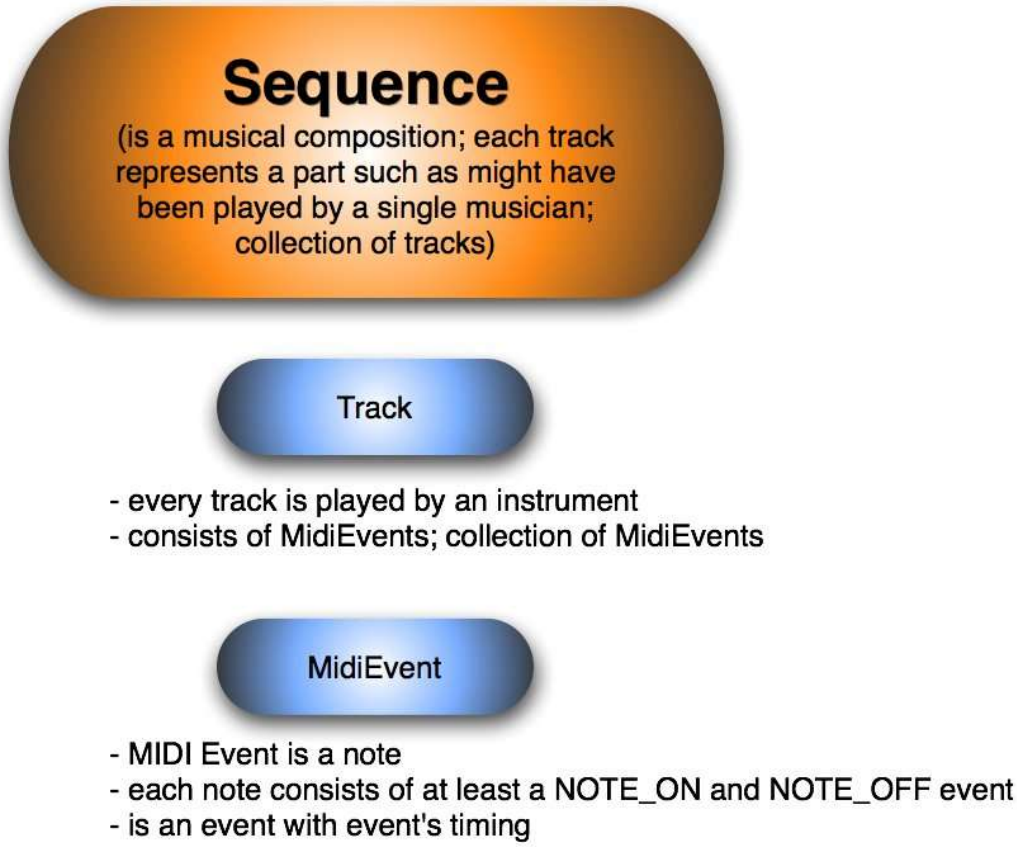


Figure 3.6: MIDI Sequence

A program that can create, edit, and perform these files is called a sequencer.

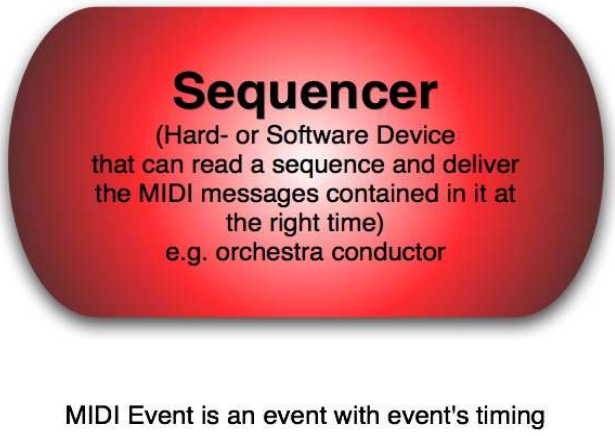


Figure 3.7: MIDI Sequencer

Many computer sound cards include MIDI-controllable music synthesizer chips to which sequencers can send their MIDI events. Synthesizers can also be implemented

entirely in software. The synthesizers interpret the MIDI events that they receive and produce audio output. Usually the sound synthesized from MIDI data is musical sound (as opposed to speech, for example). MIDI synthesizers are also capable of generating various kinds of sound effects.

Some sound cards include MIDI input and output ports to which external MIDI hardware devices, such as keyboard synthesizers or other instruments, can be connected. From a MIDI input port an application program can receive events generated by an external MIDI-equipped musical instrument. The program might play the musical performance using the computer's internal synthesizer, save it to disk as a MIDI file, or render it into musical notation. A program might use a MIDI output port to play an external instrument, or to control other external devices such as recording equipment.

The following diagram shows the functional relationships between the major components in a possible MIDI configuration based on the Java Sound API. The flow of data between components is indicated by arrows. The data can be in a standard file format, or (as indicated by the key in the lower right corner of the diagram), it can be audio, raw MIDI bytes, or time-tagged MIDI messages.

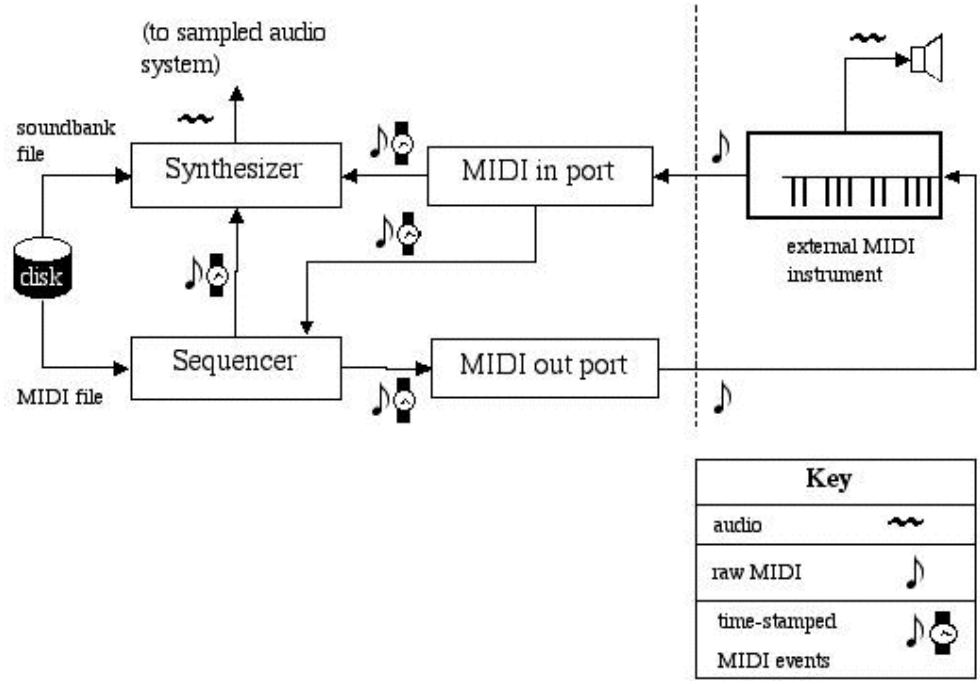


Figure 3.8: MIDI Overview

The application program prepares in this example a musical performance by loading a musical score that's stored as a standard MIDI file on a disk. A Standard MIDI file contains tracks, each of which is a list of time-tagged MIDI events. Most of the events represent musical notes which are pitches and rhythms. This MIDI file is read and then "performed" by a software sequencer. The sequencer performs its music by

sending MIDI messages to some other device like an internal or external synthesizer. The synthesizer itself may read a soundbank file containing instructions for emulating the sounds of certain musical instruments. If not, the synthesizer will play the notes stored in the MIDI file using whatever instrument sounds are already loaded into the synthesizer.
As illustrated the MIDI events must be translated into raw, non-time-tagged MIDI before being sent through a MIDI output port to an external synthesizer. Similarly, raw MIDI data coming into the computer from an external MIDI source like a keyboard instrument in the diagram. They are translated into time-tagged MIDI messages that can control a synthesizer or that a sequencer can store them for later use.

3.6 The Common Data Format[19]

Common Data Format (CDF) is a self-describing data format for the storage and manipulation of scalar and multidimensional data. It is platform-independent and provides programming interfaces for C, Java, Perl and Fortran applications to guarantee a device-independent view of the CDF data model.

The current version is CDF V3.0 and was released on February 10, 2005.

The CDF software, documentation, and user support services are provided by NASA and available to the public free of charge. There are no license agreements or costs involved in obtaining or using CDF.

The CDF software package is used by hundreds of US government agencies, universities, and private and commercial organizations as well as independent researchers on both national and international levels. CDF was adopted by the International Solar-Terrestrial Physics (ISTP) project as their format of choice for storing and distributing key parameter data.

Here are some examples listed where CDF has been accepted in public software projects:

- Interactive Data Language (IDL)
- MathWorks MATLAB Language (MATLAB)
- Application Visualization System (AVS)
- Weisang GmbH & Co. KG Data Analysis and Presentation (FlexPro)
- IBM Visualization Data Explorer (DX)

There is also software which can convert non-CDF data files into CDF files. For example MakeCDF is a CDF application that reads flat data sets in both binary and text and generates a IST CDF data set from that data. The other way around there is CDFexport which is a tool that can generate an ASCII text file of the selected

19 http://cdf.gsfc.nasa.gov

variables from a CDF file.

Chapter 4

4. xSonify – The Application

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

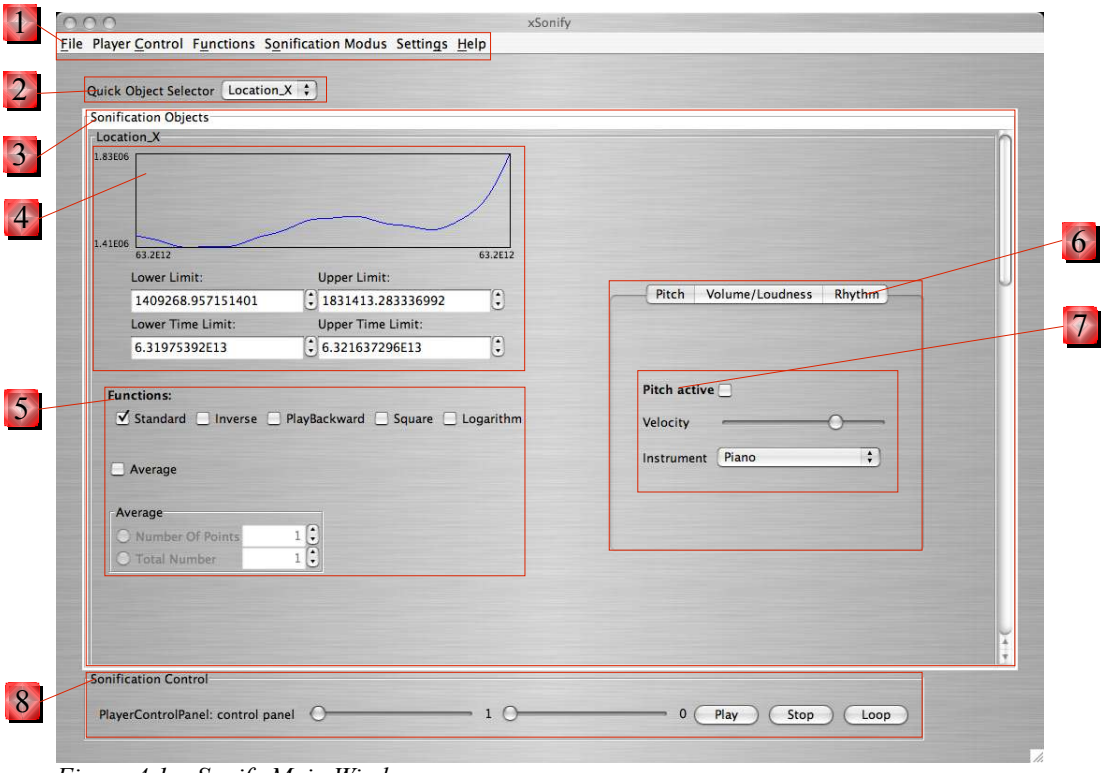


Figure 4.1: xSonify Main Window

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.

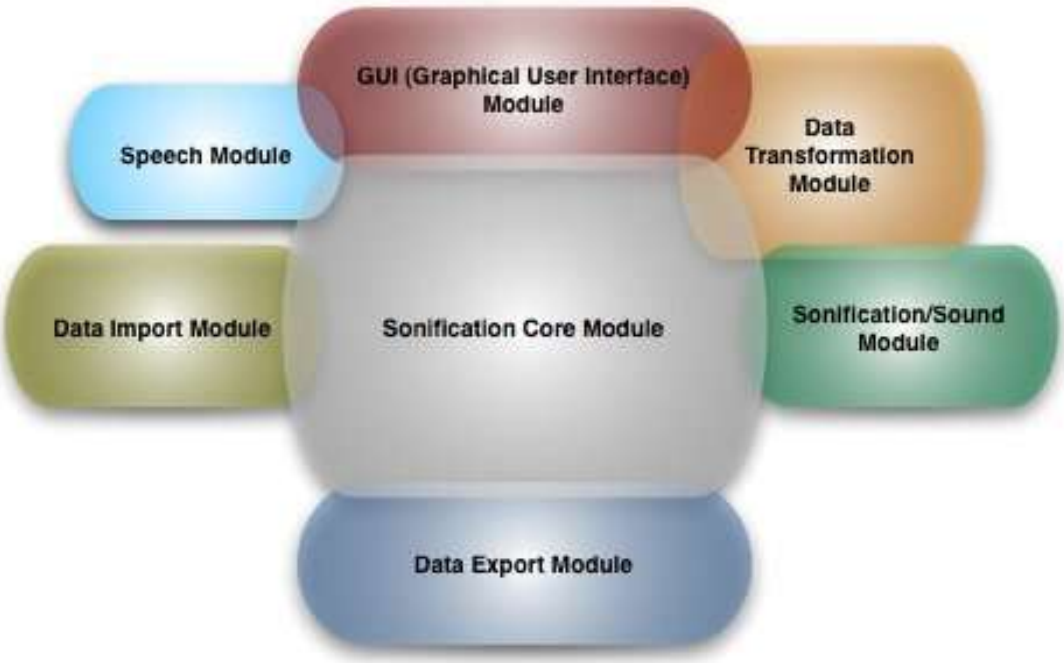


Figure 4.1: Module Overview

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.

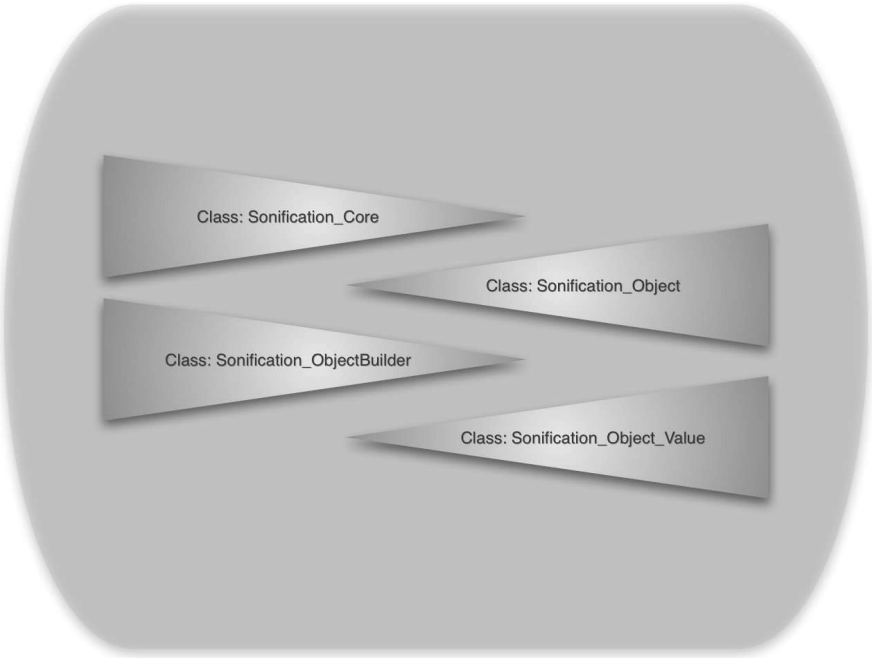


Figure 4.2: Sonification Core Module

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.

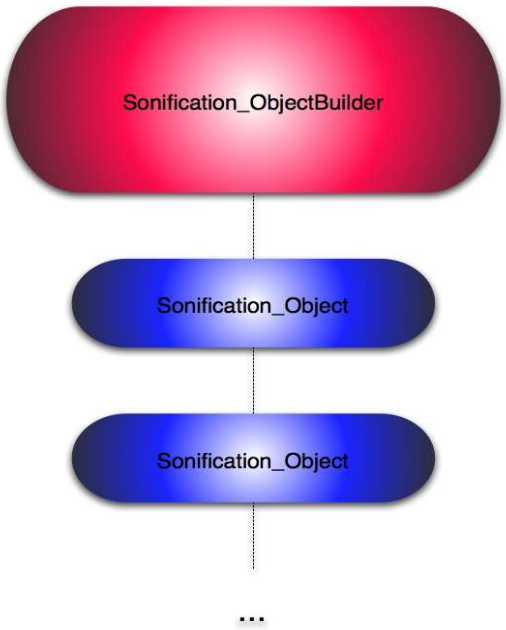


Figure 4.3: Class - Sonification_ObjectBuilder

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 sobjectname)* in class *Sonification_Object_Trans form*).

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.

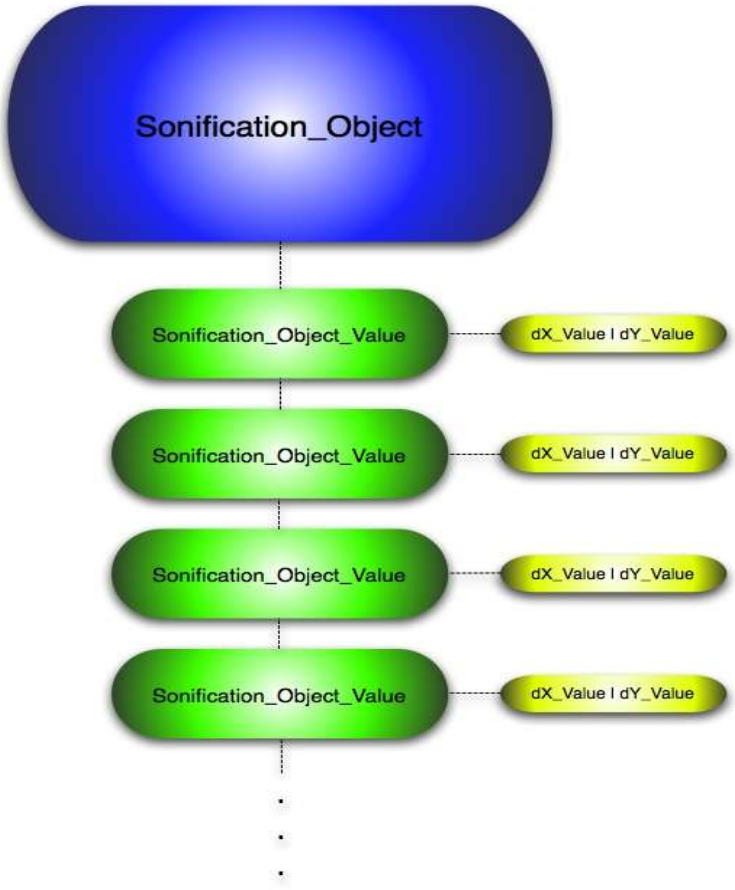


Figure 4.4: Class - Sonification_Object

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.

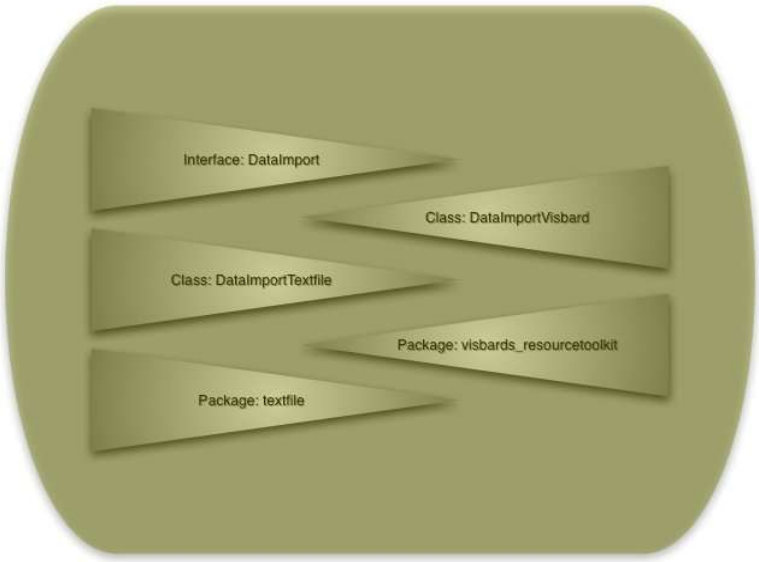


Figure 4.5: Data Import Module

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.

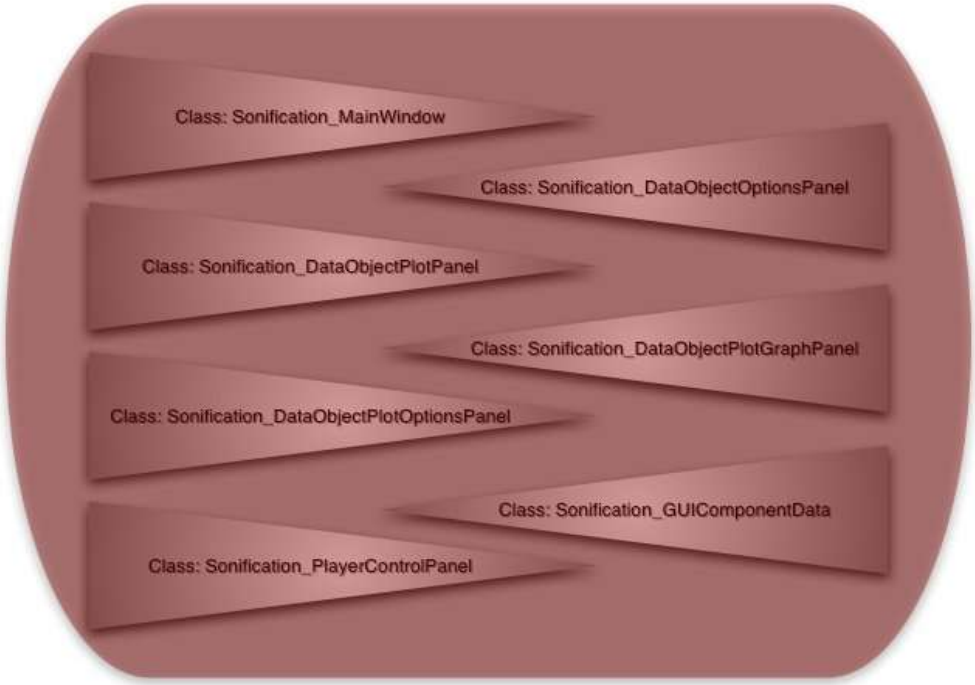


Figure 4.6: GUI Module

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 *SonificationSound* 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_Data ObjectPlotGraphPanel* 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*.

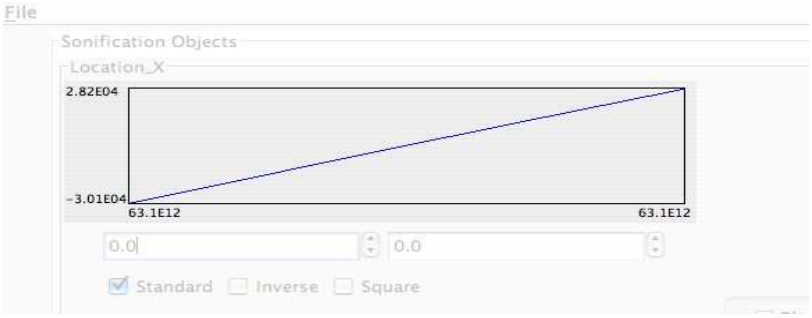


Figure 4.7: Class - DataObjectPlotGraphPanel

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.

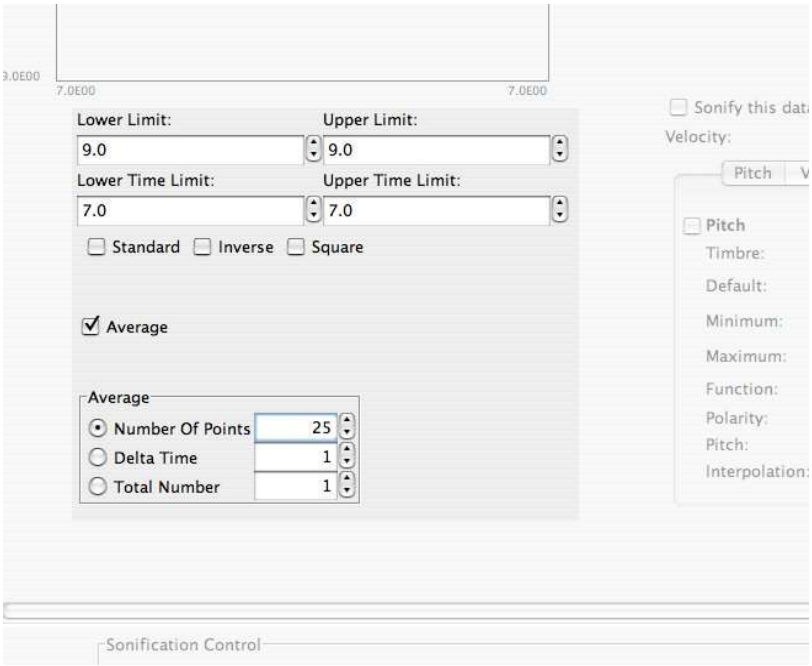


Figure 4.8: Class - Sonification_DataObjectPlotOptionsPanel

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 *JTabbedPanel* and can be selected and configured.

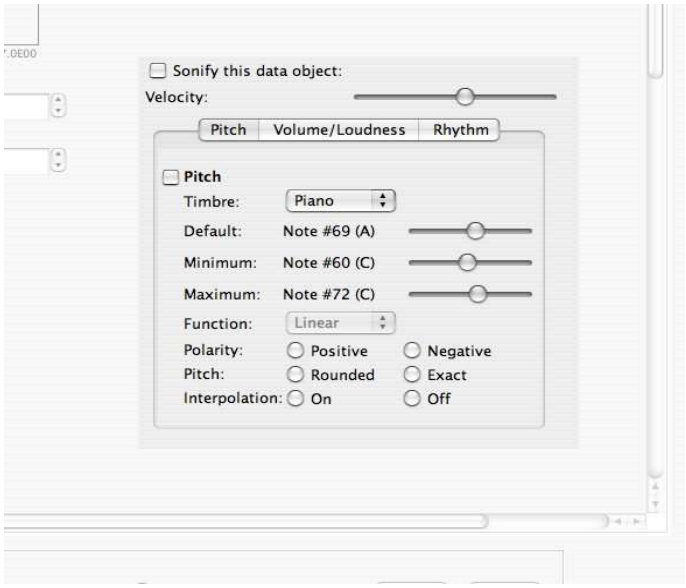


Figure 4.9: Class - Sonification_DataObjectOptionsPanel

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.

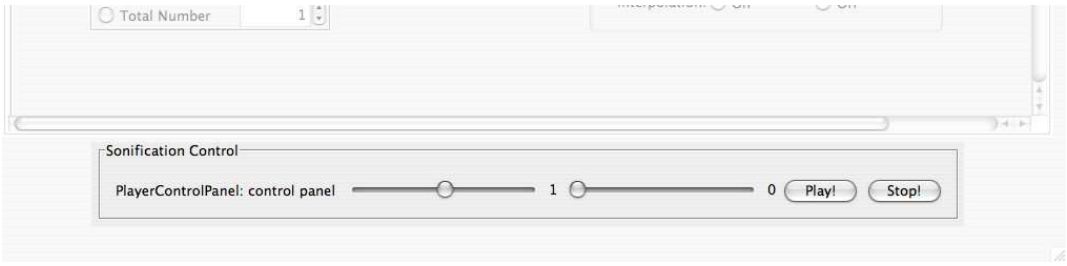


Figure 4.10: Class - Sonification_PlayerControlPanel

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.

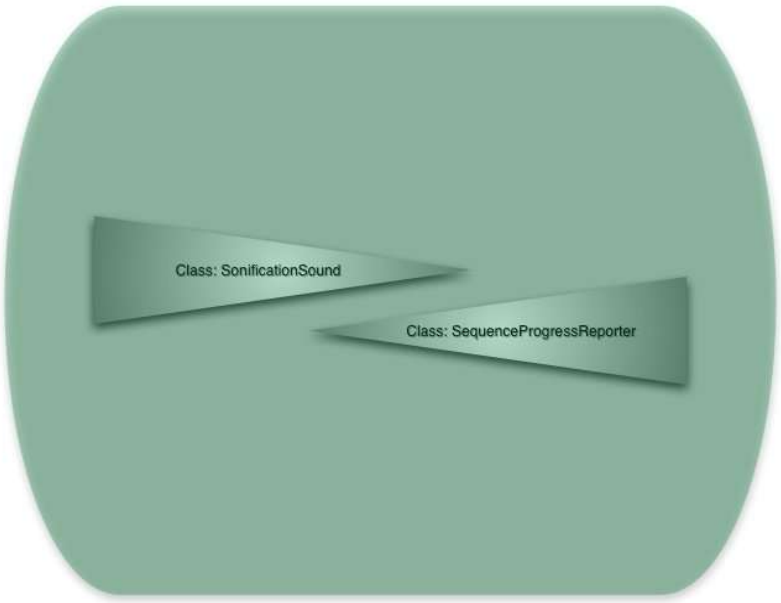


Figure 4.11: Sonification/Sound Module

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.

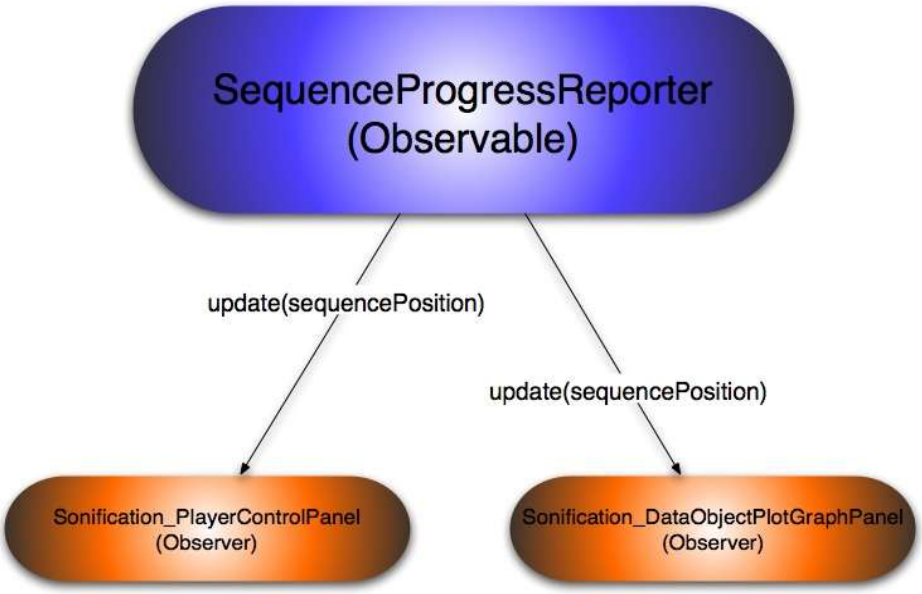


Figure 4.12: Class - SonificationSound

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 *SequenceProgress Reporter* 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.

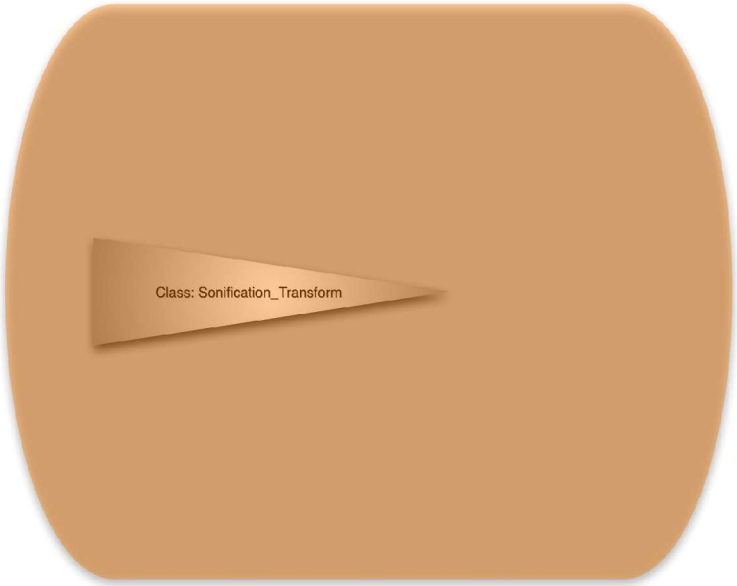


Figure 4.13: Data Transformation Module

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.

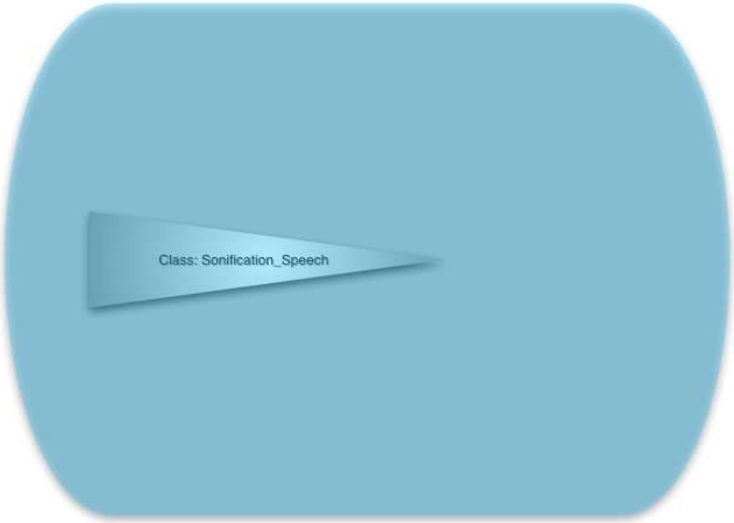


Figure 4.14: Speech Module

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. Wherever 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 4.15: 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.

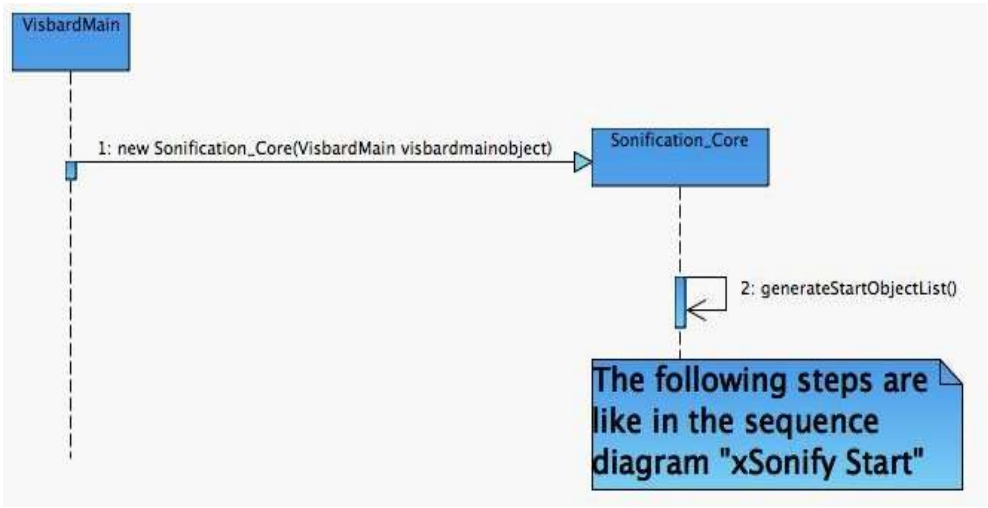


Figure 4.16: Implementation Of xSonify

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.

Chapter 5

5. Sonification Details

Sound has many attributes that can be used for representing various data dimensions, including pitch, loudness, rhythm, damping or attack/decay rate, direction, duration and repetition, timbre and harmonics, phase, and rest periods. The preferred attributes provide independence, resolution, continuity, and ease of perception by untrained users.

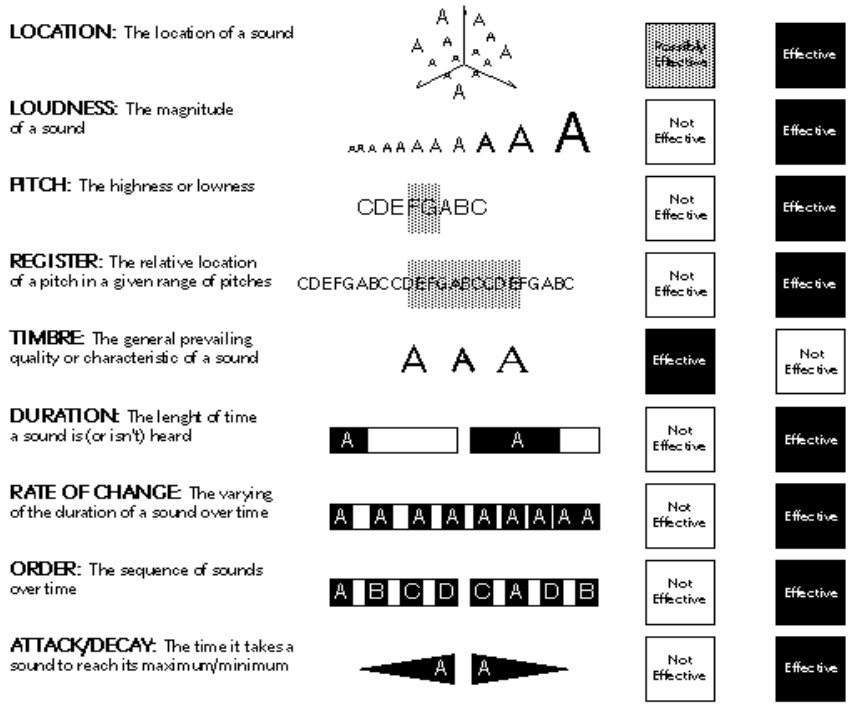


Figure 5.1: Sound Attributes

- **Location:** The location of a sound in a two or three dimensional sound space.

Location is analogous to location in the two dimensional plane of the map. As a sound variable location requires stereo or three-dimensional sound displays. Two- and three-dimensional sound allows for the mapping of left/right, up/down, (and in 3-D) forward/backward locations. Location can represent nominal and ordinal data. For example, a two-dimensional stereo sound map could use location to direct attention to a specific area of the graphic map display where the fastest change is occurring in a spatial data set over time.

- **Register:** The relative location of a pitch in a given range of pitches. Register describes the location of a pitch or set of pitches within the range of available pitches. Register is a more general case of pitch, where one can specify a high, medium and low register, each retaining a full set of chromatic pitches. (note 4) It can add to pitch as a broader ordinal distinction. An application which uses register and pitch is discussed later in this paper.

- **Timbre:** The general prevailing quality or characteristic of a sound. Timbre describes the character of a sound and is best described by the sound of different instruments: the brassy sound of the trumpet, the warm sound of the cello, the bright sound of the flute, etc. Timbre, then, implies nominal differences (Risset and Wessel 1982, Kramer and Ellison 1992). For example, a brassy sound could be used to represent an urban phenomena while a warm or mellow sound could be used to represent a rural phenomena. Such an example draws attention to the evocative nature of sound.

- **Order:** the sequence of sounds over time. The order in which sounds are presented over time can be "natural" - such as the progression from a low pitch to a high pitch - and this means that it should be easy to detect general trends (patterns) in data presented with sound variables such as pitch or loudness. The "natural order" of sounds can be manipulated to represent data "disorder" or different orders. For example, if a natural order of sounds (say pitch from low to high) is matched to chronological temporal order, any non-ordered sound will be recognizable as an indication that data are out of chronological order. An example will be discussed later in this paper.

- **Attack/Decay:** the time it takes a sound to reach its maximum/minimum. The attack of a sound is the time it takes for a sound to reach a specific level of loudness; the decay is the time it takes to reach quiet. Attack has been found to be much more successful in conveying information than decay (Lunney and Morrison 1990, 144). Attack/decay could be used to represent the spread of a specific data variable in a given unit: for example, pitch may represent an average value for the income in a county and attack/decay the spread of values; a long attack and decay would represent, then, a wide range of incomes in that county. Attack/decay may also be used to represent rates of diffusion or recession.

The sound attributes according to the sound attribute overview which haven't been explained yet are issued in the following sub chapters. They are emphasized in separate chapters because they are part of the application xSonify.

5.1 Sonification Modus: Pitch

The probably most famous way to display information in an acoustic way is to choose the pitch attribute of sound. The lowness and highness of a sound (the frequency) respectively contains in that case the information. Pitch is highly distinguishable and is one of the most effective ways of differentiating order with sound. On average, individuals can easily distinguish 48 to 60 pitches over at least four or five octaves, and this implies that pitch, divided up by octaves can be used to represent more than a single variable in a sonic display (Yeung 1980, 1121).

But beside all the positive aspects of this Sonification method there are also things which have to be considered.

- Judgments of pitch will vary somewhat from person to person.
- Western music has traditionally employed a scale of eight octaves comprised of twelve pitches each; extreme pitches, however, are hard to distinguish.
- Mapping with pitch is appropriate for ordinal data. In addition, pitch may imply direction, where, for example, an increasing pitch represents upward movement.
- Tonal sharps and flats can be used to some effect also, possibly to represent a second variable such as variations in data quality. Every twelfth pitch has the same pitch color (chroma) and this may serve to represent nominal or ordinal data (Weber 1993b). Pitch, then, can represent quantitative data, primarily ordinal. Time can be added to pitch to create a sound graph which tracks ordinal change in data over time.

5.2 Sonification Modus: Volume/Loudness

The second parameter of sound which is used to hear the data is the volume of a certain tone. According to the current values in the data sequence the volume/ loudness is measured in terms of the decibel and implies an ordinal difference. The average human can just detect a one decibel sound, can detect differences in loudness of about three decibels, and can tolerate up to approximately 100 decibels (e.g. the loudness of a jet taking off). Loudness is inherently ordered and thus seems appropriate for representing ordinal level data. Loudness may be used to imply direction and can be varied over time to represent ordinal change in data over time (e.g. to alert one to important but infrequently occurring phenomena).

It is known that humans usually become unconscious of constant sounds (Buxton 1990, 125). For example, although the hum of a computer's fan in becomes inaudible soon after switching it on, even a slight variation in the fan will be instantly noticed. This effect can be used to represent information where a quiet tone represents a steady state and any variation represents change.

5.3 Sonification Modus: Rhythm

The last modus in our software application is the idea to use a certain rhythm of a beat instrument to display the information. During this modus the whole range of data values are fragmented into a certain number of groups. Default amount of groups are 20 but can be changed in the average panel. The idea is that every group has the value of the average of all values inside a group and the value represents a certain rhythm.

According to the sound attribute overview in the introduction of this Chapter this modus covers the sound attributes “Duration” and “Rate Of Change”.

- **Duration:** The length of time a sound is (or isn't) heard. Duration refers to the length of a single sound (or silence) and can represent some quantity mapped to that duration. Silence must be used in tandem with duration if one is to distinguish the duration of multiple sounds (Yeung 1980, 1122). Duration is naturally ordinal.

- **Rate of Change:** The relation between the durations of sound and silence over time. Rate of change is primarily a function of the varying (or unvarying) durations of sounds/silences in a series of ordered sounds over time and can represent consistent or inconsistent change in the phenomena being represented.

5.4 Sonification Process

Ed Chi [16], worked on a way how to display a visualization process and named it the "Data Pipeline". Analogously to this process Sylvain Daude[17] and Laurence Nigay presented a Sonification process which transforms and prepares the raw data to be finally sonified.

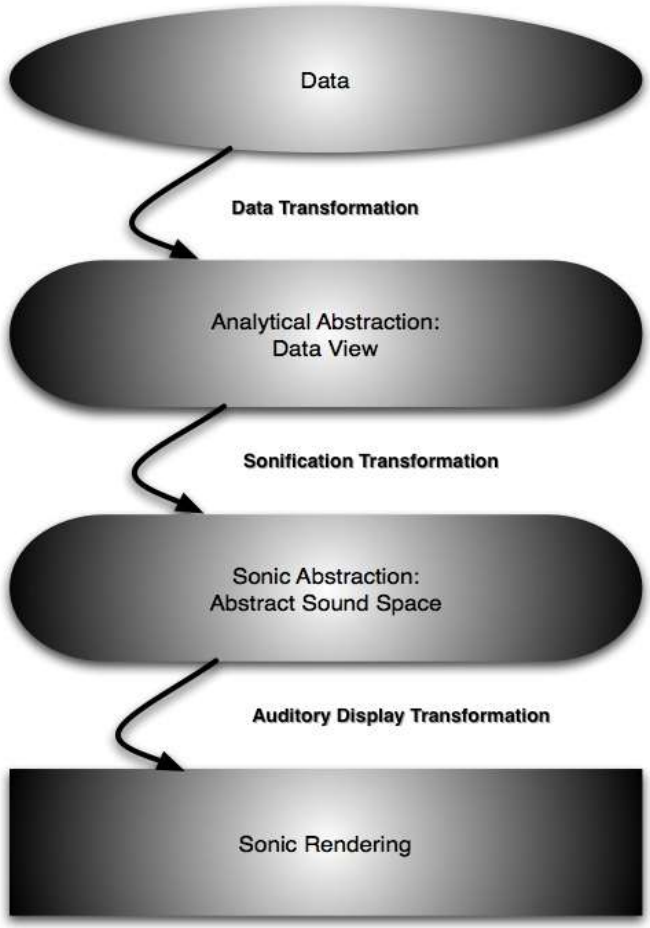


Figure 5.2: Sonification Process

The single points of this process will be explained in the following steps:

From Data To Data View: **Data Transformation**
In this step the raw data will be mapped value by value into a range of values between $0 < x < 1$. This process can be considered as a kind of standardization of values which is necessary to make the data available independent from their unit and scale.

From Data To Abstract Sound Space: **Sonification Transformation**
During this step the data are prepared according to the chosen Sonification modus. Every value will be assigned to a certain position at a "time line". This "time line" is represents the time line of the Sonification sequence which will be played in xSonify

in the MIDI player.

From Abstract Sound Space To Sonic: **Auditory Display Transformation**
This is the part where the signal is finally displayed on a physical device. In xSonify this is the part where the MIDI sequence will be transmitted to the MIDI player and played.

5.5 Delivering Of Information Through Sound And The Difficulties

Beside all the tempting opportunities and chances of the usage of the different sound attributes for Sonification I would also like to point to difficulties and challenges which can appear during the design phase of an application for Sonification.

Some complications are the direct result of how our nervous system processes sound, while other problems can be associated with the task presented or the environment in which the Sonification is used.

The following aspects which are representing some difficulties of Sonification should only give an idea of the problems which have to be faced during the design of an application:

- low resolution of some auditory variables
- limited spatial precision
- lack of absolute values
- absence of persistence
- no printout

Naturally the kind of application automatically determines if the usage of Sonification is appropriate at all and if so what kind of sound attributes or rather Sonification modus should be applied in the program.

Literature Directory

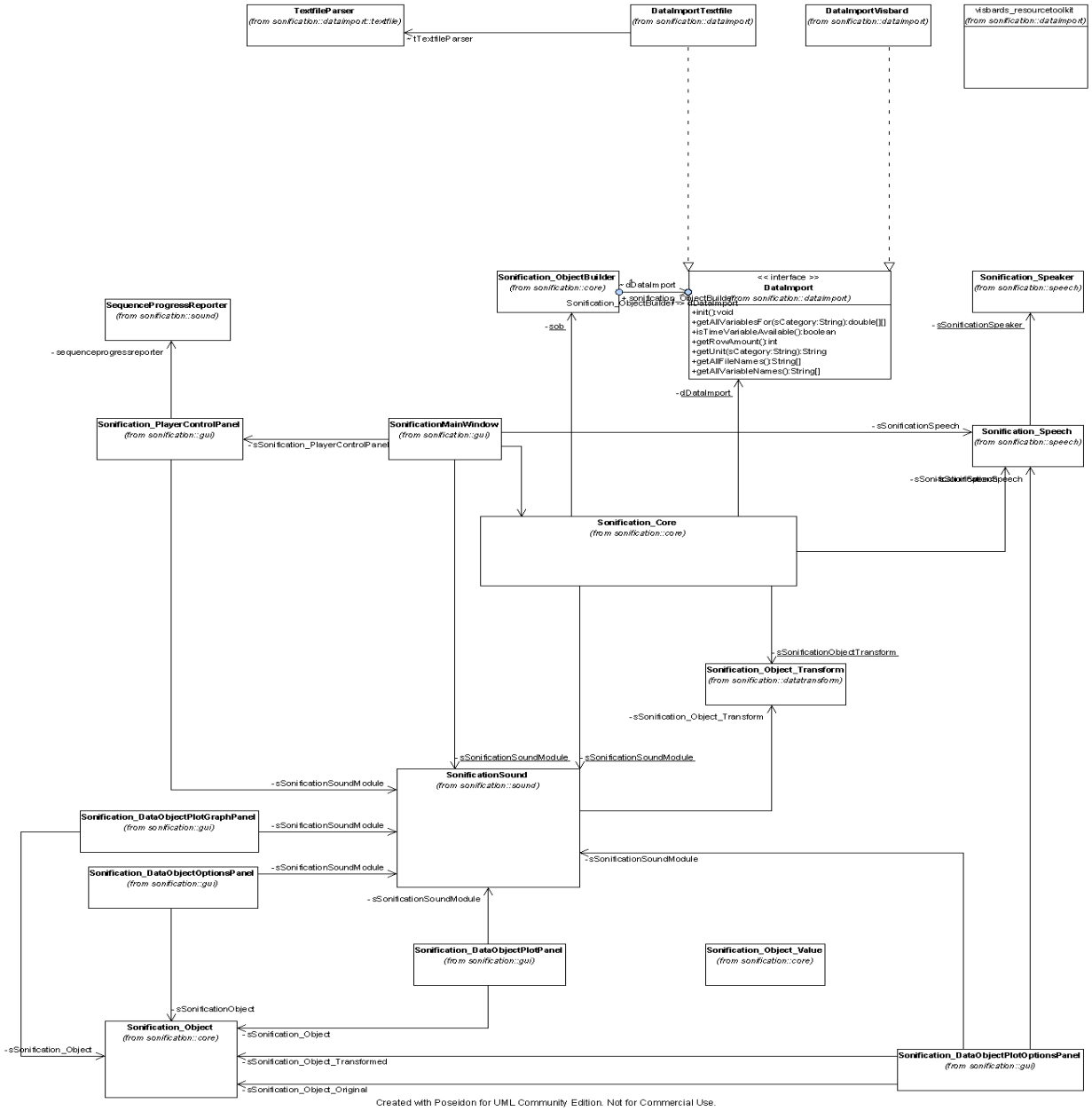
[1] Barrass Stephen, “Sonification Design Patterns”. Proceedings of the 2003 International Conference on Auditory Display, Boston, MA, USA
[2] Kramer, Gregoy, “Auditory Display”. Santa Fe Institute (1994) 186-188
[3] Pollack, I., and L. Ficks. “Information of Elementary Multidimensional Auditory Display”. J. Acoustic Society America (1954)
[4] Speeth, S. D. “Seismometer Sounds”. J. Acoustic Society America (1961)
[5] Chambers, J. M., M. V. Mathews, and F. R. Moore. “Auditory Data Inspection”. Technical Momorandum no. 74-1214-20, AT&T Bell Laboratories (1974)
[6] Edward Yeung, “Auditory Display”. Santa Fe Institute (1994) 38
[7] Fubini, E, A. De Bono and G. Ruspa. “System for Monitoring and Indicating Acoustically the Operating Conditions of a Motor Vehicle.” U.S. Patent #4,785,280, U.S. Patent and Trademark Office (1986)
[8] Smith, S. “An Auditory Display for Exploring Visualization of Mutlidimensional Data.” In Workstations for Experiment, edited by G. Grinstein and J. Encarnacao. Berlin: Springer Verlag
[9] Kramer, G. “Audification of the ACOT Predator/Prey Model.” Unpublished research report prepared for Apple Computer's Advanced Technology Group, Apple Classrooms of Tomorrow (1990)
[10] Kramer, G. “Audification: Using Sound to Understand Complex Systems and Navigate Large Data Sets.” Proceedings of the Santa Fe Institute Science Board, Santa Fe Institute (1990)
[11] Kramer, G. “Audification: The Use of Sound to Display Multivariate Data.” In Proceedings of the International Computer Music Conference, 214-221. (1991)
[12] Official Website of ICAD, http://www.icad.org/
[13] Scaletti, C., and A. Craig “Using Sound to Extract Meaning from Complex Data.” In Extracting Meaning from Complex Data: Processing, Display, Interaction II, edited by Edward J. Farrell, SPIE 1459, 207-219 (1991)
[14] Rabenhorst, D. A., E. J. Farrel, D. H. Jameson, T. D. Linton, and J. A. Mandelman. Complementary Visualization and Sonification of Multi-Dimensional Data, Extracting Meaning from Complex Data: Processing, Display, Interaction, edited by E. J. Farewell, SPIE Vol. 1259, 147-153 (1990)
[15] Official Website of the project “Sonification Sandbox”, http://sonify.psych.gatech.edu/research/sonification_sandbox/sandbox.html
[16] Ed Chi, J. Riedl, “An Operator Interaction Framework for Visualization Systems”, Proceedings Info Vis '98.
[17] Sylvain Daude, Laurence Nigay, “Design Process For Auditory Interfaces”, Proceedings ICAD 2003

[18] http://www.wikipedia.de

Appendix A

xSonify Class Diagram

This UML Class Diagram illustrates the classes and interfaces. It also shows the classes aggregation relationships marked with an arrow.



Appendix B

xSonify Sequence Diagrams

The sequence diagrams should help to understand xSonify and the applications inner life better. The following chosen standard processes in the program are the most common sequences and also the sequences with the most interactions between the relevant objects.

- **xSonify Start:**

  This sequence represents the initial start of the application.

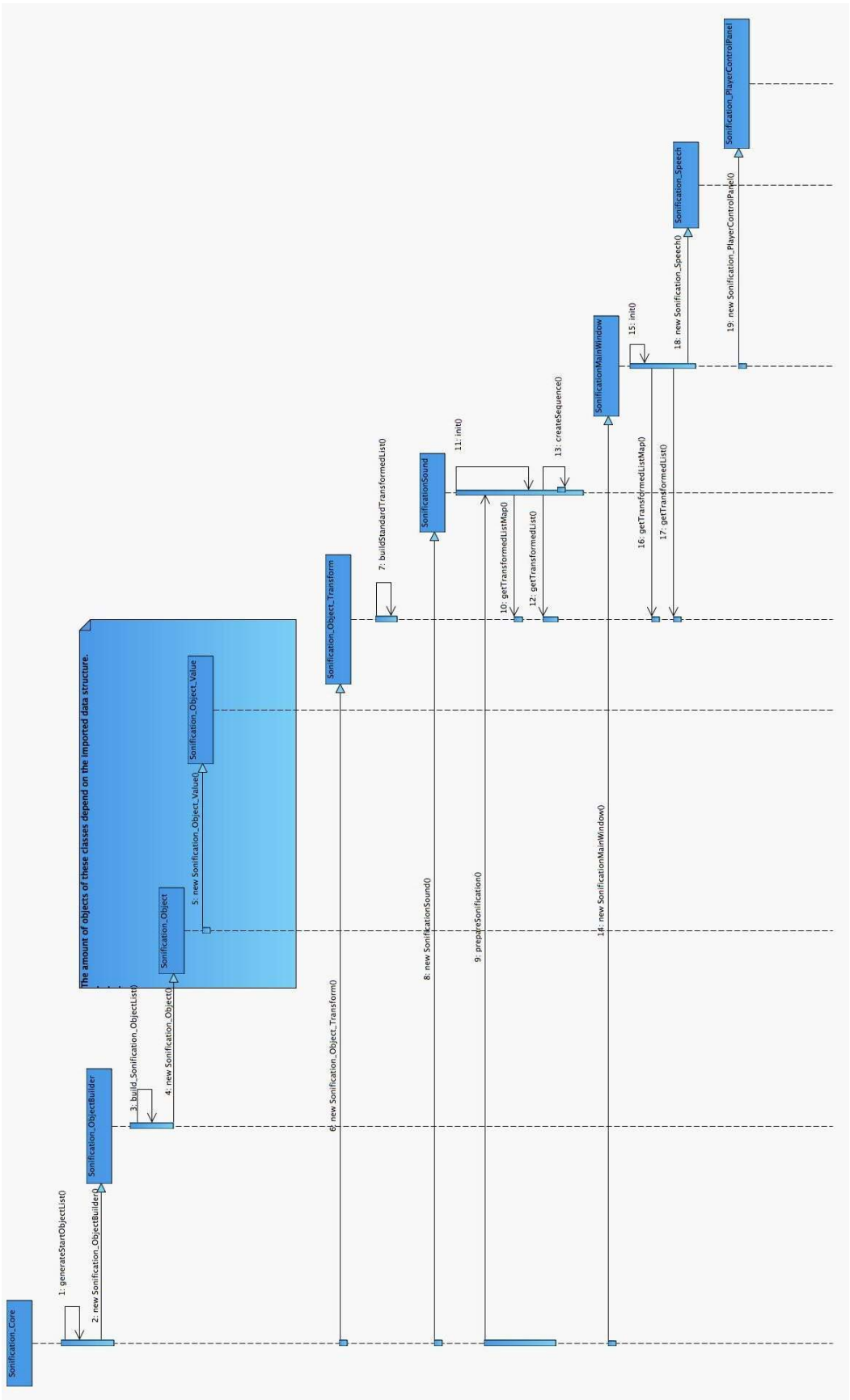
- **xSonify DataImport:**

  This sequence diagram actually contains to individual sequences. The data import via the ViSBARD Resource Toolkit and the data import from a regular local file via a text parser.

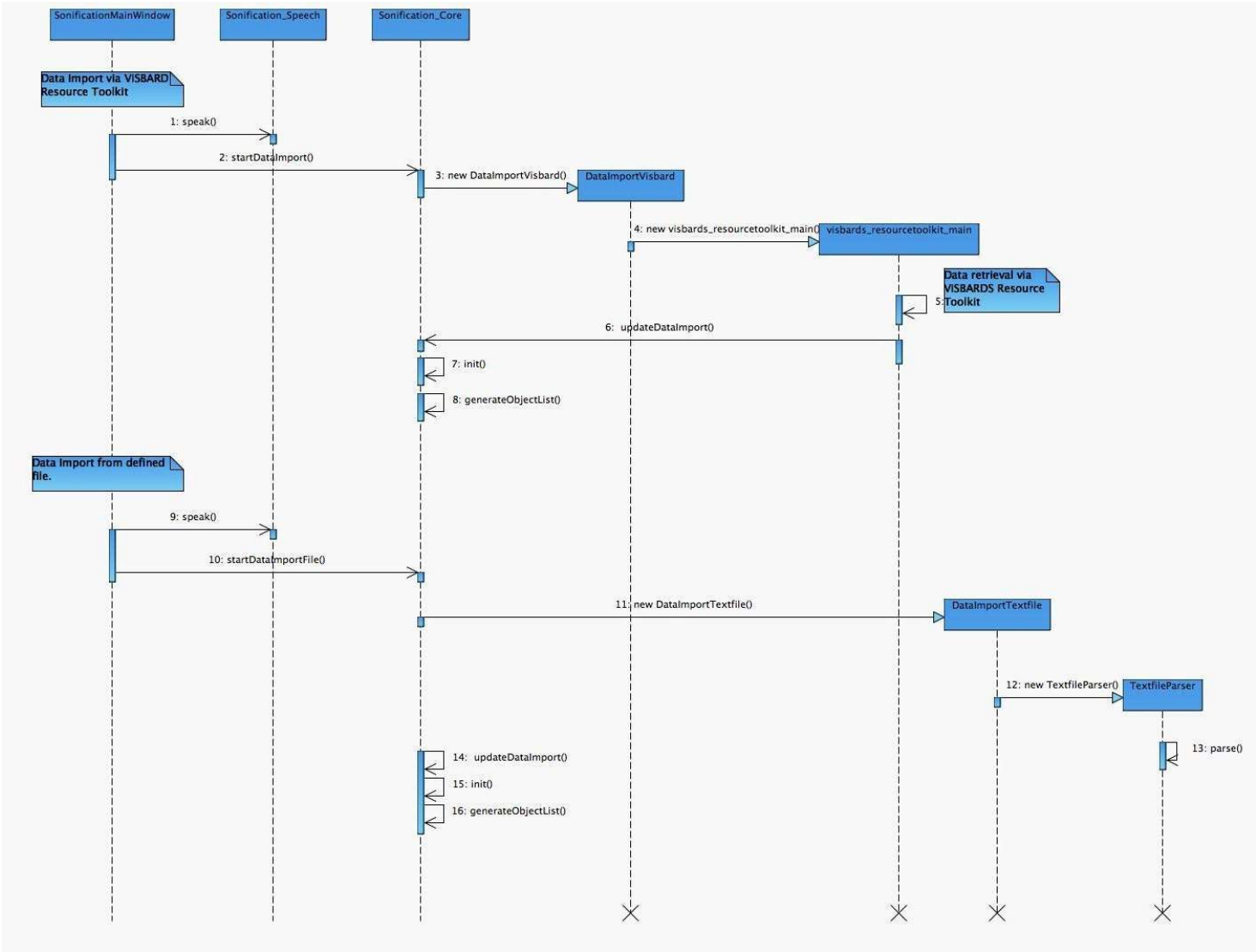
- **xSonify Generate Sound:**

  This display of a sequence shows the process after the “Play” button was clicked.

B1. xSonify Start



B2. xSonify DataImport


SonificationMainWindow
Sonification_Speech
Sonification_Core
Data Import via VISBARD Resource Toolkit
1: speak()
2: startDataImport()
3: new DataImportVisbard()
DataImportVisbard
4: new visbards_resourcetoolkit_main()
visbards_resourcetoolkit_main
Data retrieval via VISBARDS Resource Toolkit
5:
6: updateDataImport()
7: init()
8: generateObjectList()
Data Import from defined file.
9: speak()
10: startDataImportFile()
11: new DataImportTextfile()
DataImportTextfile
12: new TextfileParser()
TextfileParser
13: parse()
14: updateDataImport()
15: init()
16: generateObjectList()

B3. xSonify Generate Sound