Software tools for blind and visually impaired in GNOME2 –Accessibilty in GNOME2 –

Dr. Victoria Iordan, Associate Professor
University of the West, Timisoara, Romania

REZUMAT. Această lucrare are drept scop prezentarea şi analizarea posibilităţilor ca o persoană cu handicap vizual să utilizeze calculatorul folosind desktopul GNOME. În prima parte lucrarea prezintă noţiunea de **accesibilitate** ("Accessibility") în general şi identifică soluţii existente pentru a da sens noţiunii de accesibilitate referitor la persoanele cu deficienţe vizuale, făcându-se analiza desktopului GNOME şi a suportului pe care acesta îl oferă pentru accesibilitate. Partea a doua este dedicată analizei unicului screen reader pentru GNOME (cel puţin până în momentul de faţă): **Gnopernicus**. Acesta este parte a **GAP** (Gnome Accessibility Project), managementul proiectului fiind realizat de Sun Microsystems în colaborare cu BAUM Retec AG dar implementarea efectivă realizându-se în România, de către BAUM Engineering S.R.L. Este analizat modul în care este posibil să i se ofere utilizatorului un feedback auditiv (comunicarea cu diferite mecanisme TTS – Text To Speech) şi de asemenea modul în care se poate oferi un feedback Braille (cum se pot accepta mai multe tipuri de dispozitive Braille, mai multe tabele braille).

1. What Accessibility means?

Accessibility is about enabling people with disabilities to participate in substantial life activities that include work and the use of services, products, and information.

Everybody must be sure that PC technology can be tremendously empowering. A significant portion of the growth in the U.S. economy during the 1990s can be directly attributed to productivity improvements

brought on by PC technology. With each new generation of PCs, new applications are enabled that offer more benefits. As PCs become more powerful, they improve most people's lives in more ways every day.

The PC technology can be particularly empowering for people who are blind or visually impaired. In an increasingly information-driven economy, the potential benefit of personal computers on employment for blind and visually impaired people is dramatic. The unemployment rate for blind and severely visually impaired people in the U.S. is currently over 70%. The PCs, paired with accessibility technology, have allowed the disabled users to effectively overcome their visual disabilities, and to live and work with able-bodied people in ways that would have otherwise been impossible. By opening up new opportunities to hold mainstream jobs – good jobs that make full use of a person's mental talents and abilities – PCs can bring blind and visually impaired people out of a world of dependence and isolation and into one of full productivity and participation. Indeed, PCs have promised to be the most important tool for empowerment of blind and visually impaired people since the invention of Braille or the magnifying glass.

- **History of the Development of Blind and Low-Vision AT**

The history of accessibility technology for blind and visually impaired users is one of rapid innovation and continual adaptation. In the early days of computers, accessibility for blind users was not terribly difficult. A standard 80x24 text screen used by DOS and other text-only operating systems could be reliably translated into refreshable Braille or speech (using special hardware connected to a serial port). These were very exciting days for blind people. Using the new technologies, they could stand practically level with other sighted colleagues.

Then came Windows… Windows 3.0 and 3.1 didn't work with the traditional DOS screen readers. The visual interface was fundamentally more complex, as well as the underlying operating system technologies. Unfortunately, Windows 3.1 did not come with any support for accessibility utilities. Screen reader companies had to re-invent their products, using much more complex methods for gathering and interpreting information about what was happening on the screen. It took a while for screen reader companies to develop these utilities, and in the interim a number of blind people lost their jobs as their employers switched to Windows. As adoption of Windows accelerated and the introduction of Windows 95 (another major architectural revision of the operating system) loomed, blind people saw that they could once again lose jobs by the thousands. To remedy the situation, people within the Massachusetts Commission for the Blind, the

Massachusetts Assistive Technology Partnership (MATP), the National Council on Disability, and the National Federation of the blind lobbied Microsoft to fix the problem by making windows accessible. Specifically, they asked for standard "hooks" in the operating system that would support the creation of robust and reliable accessibility utilities. Their requests were generally rebuffed. When these methods failed, the advocates began to lobby state governments to get them to boycott Microsoft products until adequate accessibility technologies could be developed. That got Microsoft's attention.

Microsoft's response to the commitment to provide "hooks" in the operating system for use by accessibility utilities was the creation of the MSAA interfaces to facilitate direct communication between the applications and utilities. Initially it was hoped that this technology would solve the problem of making application accessible. Unfortunately, it took several years to complete the first version of MSAA, relatively few applications outside of Microsoft use it, and prospects for widespread adoption are not good. No corresponding standard lower-level hooks have been provided to implement reliable support for legacy and non-compliant applications (which grow geometrically in number every year).

So the current situation is one of mixed progress and a similarly mixed outlook toward the future. While the overall reliability of screen readers for Windows has increased in the last 5 years, they are still plagued by technical problems. The Windows operating system has increased support for accessibility, but it also introduces significant new problems with each major update.

But what about Linux? There are some solutions to allow people with visual disabilities to use it? The answer is YES: visually impaired people are able to use GNOME desktop (there are many signals that KDE will be also accessible in the near future) by using Gnopernicus as a screen reader and magnifier.

- **How Screen Readers Work – Overview**

Screen readers are special utility programs that run concurrently with other regular programs to figure out what is on the screen, interpret it, and then interact with the blind or visually impaired user through text-to-speech or refreshable Braille. Screen readers use two primary methods to gather information about what is on the screen:

a) directly communication with the applications that are running. This is preferred, but requires additional work on the part of the application vendor

b) "hooks" in the operating system that are used to intercept low-level operations, such as text output, to deduce what is going on.

These and other sources of information are combined in real time to construct an alternate view of the world, frequently called an "*off-screen model*" (OSM).

Since utility programs do not have eyes to observe the output to the screen, they take a more sophisticated approach. The observation of normal output to the screen involves collecting (directly or indirectly) information from multiple sources within the system during normal program operation. Considered abstractly, some information is collected at a higher level (closer to the user) and some information is collected at a lower level (closer to the hardware). An example of a highest-level operation would be direct communication between the screen reader and a running application. In this instance the screen reader might ask, "What is under the mouse cursor?" or "What text is in this rectangle on screen?" This kind of direct communication represents the ideal situation, because the application has the opportunity to provide information about what is happening on screen that is sufficiently detailed and complete such that the screen reader can provide an adequate translation for the user. In the less ideal (though more common) situations, if the application does not provide a direct method for querying what is happening on the screen, the alternative fallback strategy for the screen reader is to infer, using lower-level information, what is happening. For example, all word processors generally send blocks of text to the screen using APIs provided by the operating system. By intercepting calls to the text-display APIs, the screen reader can see where most of the text is on the screen, and can then try to infer what is actually going on. This is a simplified example. Screen readers frequently rely on several types of intercepted information when making inferences about what is happening on the screen because each source of information is incomplete.

The process of intercepting API calls in the system is called "hooking". Historically there have been two primary methods of hooking calls in the operating system. "Hooking high" means hooking the APIs that the operating system provides for use by applications. "Hooking low" means hooking the display driver interfaces (DDIs), a step closer to the hardware (video card).

The logic that must be used for interpretation and translation of the available high-level and low-level information is frequently very specific to individual applications. This means that the screen reader must effectively be tuned to work with each new application most effectively. This kind of application tuning is done through creation of application scripts. Without

this additional tuning, the user would have a very difficult time making sense of the information that is provided by the OSM.

- **How Magnifiers Work – Overview**

At the most general level, screen magnifiers intercept drawing operations within the system before those operations result in actual colored pixels on the screen. They then stretch the resulting content, modify the content slightly to make it more legible (perhaps changing the colors of text or applying additional smoothing effects to make the text legible when stretched to larger sizes), and then send the stretched and modified content to the screen.

In practice, screen magnifiers frequently intercept drawing operations within the system at the display driver level using methods very similar to those used by screen readers. By replacing the actual display driver, screen magnifiers "see" almost all drawing operations, interpret those drawing operations to create a private version of the screen image, and then send modified output to the actual display driver.

Screen magnification utilities often contain some speech-output capabilities that are complementary to the magnification functions (for example, speaking the thing that is being magnified), and are increasingly maintaining their own off-screen models as with screen readers. In this way the line between screen magnifiers and screen readers is blurring.

2 What is GNOME?

The **GNOME** (**G**NU **N**etwork **O**bject **M**odel **E**nvironment) project has built a complete, free and easy-to-use desktop environment for the user, as well as a powerful application framework for the software developer

GNOME 2.0 is an open source, free software project that runs on most UNIX and GNU//Linux platforms. Sun Microsystems provides and supports GNOME desktop as the default desktop for the Solaris Operating Environment.

GNOME is part of the GNU project. The GNU Project was launched in 1984 to develop a complete Unix-like operating system, which is free software: the GNU system. (GNU is a recursive acronym for "GNU's Not Unix) Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as "Linux", they are more accurately called GNU/Linux systems.

The GNOME desktop was designed with the user in mind —it is very easy-to-use and highly personalizable, allowing individuals to tailor its behavior to suit their specific needs. This includes users with physical disabilities such as poor vision, blindness, deafness, or impaired motor skills —GNOME is built around an architecture that allows for the easy creation of accessible solutions that break down the barriers these users often face when using computer technology.

GNOME 2.0 has been also designed with accessibility considerations in mind and provides a robust framework that makes it much easier for accessible applications to be created. It also provides a standard interface for integrating assistive technologies such as screen readers and screen magnifiers. Notably, the fragility of accessible solutions is largely eliminated when deployed for the GNOME desktop. With GNOME's accessibility architecture, an assistive technology and an application have no direct dependencies on one another. This means, for example, that if an application is updated, the assistive technology does not necessarily have to be updated as well —and vice versa.

The GNOME 2.0 platform includes several built-in solutions designed to assist users with disabilities. In addition to a core set of accessible applications and utilities provided with the desktop, the following two assistive technologies are available:

- **GNOPERNICUS** SCREEN READER &MAGNIFIER.Gnopernicus is designed for people who are blind or who have low vision. It combines the functions of a screen reader and a screen magnifier in a single solution. The reader can follow the focus of an accessible application and generate synthesized speech, using several text-to-speech engines, or send output to a refreshable Braille display. The magnifier enlarges the area of the screen around the area of focus and automatically updates the display as the focus changes.
- GNOME ON-SCREEN KEYBOARD. GOK, the GNOME on-screen keyboard, is for users who have difficulty using a standard keyboard or mouse, and is commonly referred to as an alternative input system.

Accessible applications you create for the GNOME 2.0 platform will work with these assistive technologies without you having to know anything about how they work. All that is important is that they are integrated using the GNOME accessibility framework. The technologies can also be used with Java technology-based applications that use the Java Foundation Classes (Swing).

- **GNOME Accessibility Architecture**

GNOME 's accessibility architecture, much of it modeled after Sun's pioneering work for the Java platform, has been carefully crafted to allow applications that use standard user interface widgets (provided by GNOME's **GTK+** widget toolkit) to inherit a considerable amount of accessible behavior for free.

Even applications using custom widgets can be made accessible using the facilities of the GNOME Accessibility Toolkit (**ATK**), which allows applications to expose their properties and attributes to assistive technologies in a consistent, well-defined manner.

The architecture for integrating assistive technology into the system is also compelling. GNOME 2.0 provides a standard interface, called the Assistive Technology Service Provider Interface (**AT-SPI**), that brokers all communication between the AT and running GNOME applications.

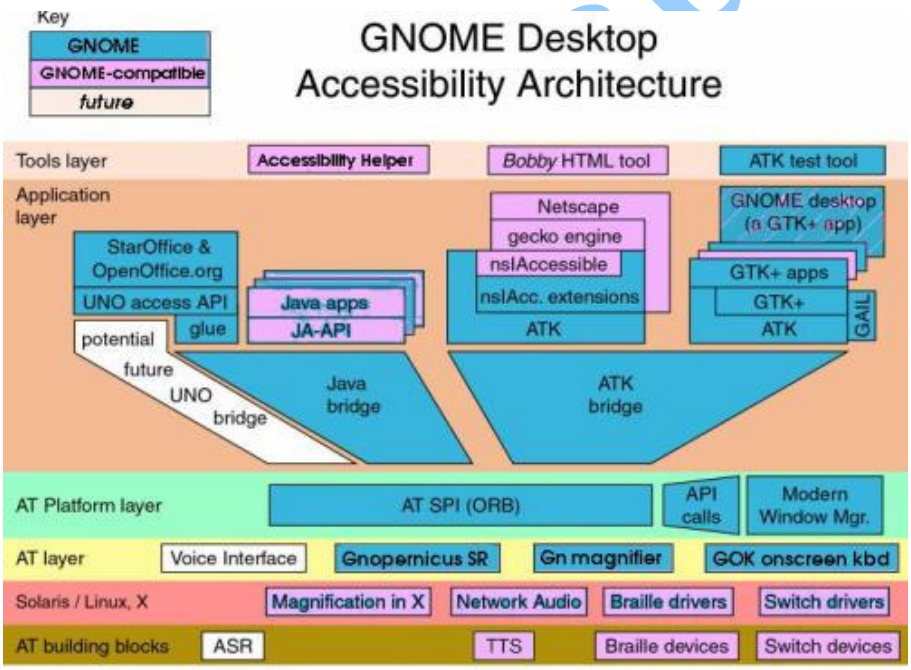

fig. 1. *Accessibility Architecture in GNOME Desktop*

In addition, a bridge is provided between the AT-SPI and Java technology-based applications that use Swing user interface components so that the very same AT can be used with these applications as well.

As the previous diagram show it, the communication process between an assistive technology and an application (in order to have an appropriate magnifier/audio/Braille feedback for a visually impaired user) has a layered architecture.

Between AT layer and application layer we have a layer, which includes AT-SPI – the core of the GNOME Accessibility concept. Basically, the entire communication between the screen reader and the application are made via at-spi events and at-spi calls.

In order to provide useful information, at-spi needs to "know" what happens in the current application. There are two ways to communicate with the application: by using ATK bridge (in case of applications developed with GTK+) or by using Java bridge (in case of Java-based applications).

The three main concepts in GNOME Accessibility – GTK, ATK, AT-SPI – will be analyze in the following sections:

- **The GTK+Toolkit**

GTK (GIMP Toolkit) is a library for creating graphical user interfaces. It's called the GIMP toolkit because it was originally written for developing the GNU Image Manipulation Program (GIMP), but GTK has now been used in a large number of software projects, including the GNU Network Object Model Environment (GNOME) project.

GNOME 's primary widget toolkit, GTK+, is made up of a comprehensive set of user interface widgets -check boxes, radio buttons, combo boxes, editable text fields, tables, and many more (On the Microsoft Windows platform, objects such as these are commonly referred to as controls. Under the Java programming language, they are called components.). Most applications designed for the GNOME desktop use this standard set of widgets in the windows and dialog boxes they present to users, thus ensuring users experience a consistent look and feel not only within a single application, but also across the entire desktop.

For GNOME 2.0,the GTK+ widgets share an important characteristic —*they all support GNOME's accessibility interface*, an interface that makes it possible for the state of a widget to be probed and queried at run time so that information an assistive technology requires can be retrieved. Since accessibility support has already been integrated into GTK+, if you build an application around the GTK+ widgets, no "heavy lifting " is required to provide basic accessible functionality. It is provided automatically.

Of course, there are other design guidelines that you should follow in order to ensure an application has a user interface that is well suited for use by people with disabilities. These guidelines are not particularly onerous since they embody good programming design concepts anyway. Some examples:

- Applications should be fully accessible from the keyboard by having the tab key move the focus from one widget to the next in a logical order and by having accelerator keys for all the menu selections.
- Frequently used items should not be buried deep in menu trees.
- Attributes such as colors and font sizes should not be hard coded in the program.
- Information that is to be presented audibly should be presented visually at the same time.

- **The Accessibility Toolkit (ATK)**

ATK : *in-process* accessibility API, written to by applications and implemented on behalf of widgets by GAIL.

It is the ATK that defines the accessibility interface that is implemented by standard GTK+ widgets so that accessibility information can be stored and retrieved in a consistent manner throughout the system (The implementation is in a library called *GAIL -the GNOME Accessibility Implementation Library*). As you would expect, this interface defines the precise list of functions that must be made available in order to support accessible behavior. The accessibility interface defined by ATK is designed to be compatibly extensible to support applications and assistive technologies as they evolve.

The accessibility functions permit assistive technology developers to extract important information from GNOME 2.0 widgets. ATK contains text-related functions, screen-coordinate information, image-related functions, selection-related functions, table-related functions, and value-related functions.

In simple terms, these functions provide a standard technique for assistive technologies to extract vital information from a running application, such as the name of the widget with focus, the state of the widget, and the role of the widget. Armed with this information, assistive technologies can adapt dynamically and interact with the user in a manner that makes sense for a given situation.

What if you choose not to use the standard GTK+ widgets, and to create and use custom widgets instead? This is a choice that is made very frequently, usually to give a unique look and feel to an application or to

create a library of reusable objects designed for a specific market. The good news is that you can still use custom widgets when building GNOME 2.0 applications without sacrificing accessibility. Although additional effort is required, accessibility support can be integrated by implementing the interfaces defined by ATK, just as they have been implemented in the GAIL library for the standard GTK+ widgets.

- **The Assistive Technology Service Provider Interface (AT-SPI)**

AT-SPI: *out-of-process* accessibility API used by AT.

AT-SPI is the module in GNOME2 that allows the user to spy all events in system and get information from objects that generate these events. As it can be seen in next picture, this module gets information from all processes in system(fig. 2).

Events are passed from application to the registry, and dispatched to listening AT. API calls to AT-SPI are made by ATs to both the registry and the applications (via their bridges).

What "registry" means in GNOME context will be shortly explained by saying what GConf means: GConf is a system for storing configuration information, that is, key-value pairs. GConf provides a notification service so applications can be notified when a key's value is changed. GConf also allows for pluggable storage mechanisms (text files, databases, etc.); allows administrators to install default values; and allows application authors to document their configuration keys for the benefit of administrators.

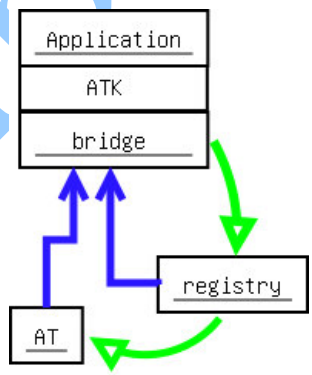

fig 2. *How AT-SPI works*

GConf was inspired by Wichert Akkerman's configuration system specification, originally developed for the Debian project. Other sources of ideas include the Windows registry.

ATK is in-process API used within the application domain. The information that could be set with this API (name for objects, description for actions, pictures etc) could be obtained by this toolkit only in current process. Applications use it to explicitly provide accessibility information. This information is exported by the process via a bridge. Another process could get information if it implement an AT and ask this bridge for what it need.

AT-SPI requires bridges and implementations for application UI toolkits. Any other application cannot communicate with it. The offered API is uniform for heterogeneous environment, because the interprocesses communication is made via CORBA.

The main purposes of the module are to offer object-based interfaces and services. Relative functions to a special kind of object (ex: text, value etc) are applied to an appropriate interface (ex: AccessibleText, AccessibleValue etc). And events are object based. For example "object:selection-changed" and "object:state-changed" could be emitted by an object.

Every object has some properties, which exist for all objects and some, which have sense only for some. General properties are name, description, role, state. The special properties could be obtained by querying an appropriate interface. For text object, some properties are: caret position, number of characters, all or a part of the text etc.

The objects are arranged in a hierarchical parent/child order. In any moment an object can be queried about its parent or about a child.

Also relations between objects are supported. For example, if a check box has property to make active or inactive a collection of other object, it is a **control for** these objects, which are **controlled by** the first. In this way information for logical grouping of objects can be obtained. Other kind of relation are: **label for, labeled by, member of** (ex: for a group of radio buttons) etc.

3. Gnopernicus – a general presentation

The Gnopernicus project will enable users with limited vision, or no vision, to use the Gnome 2 desktop. Gnopernicus is part of the **GAP** (Gnome Accessibility Project), developed by Sun Microsystems and BAUM RETEC AG.

By providing automated focus tracking and full screen magnification, Gnopernicus will aid low-vision Gnome users, and its screen reader features

will allow low-vision and blind users access to standard GTK+ and Java-based GUI applications via speech and Braille output. By leveraging Gnome 2's built-in accessibility framework, Gnopernicus will make interacting with applications more efficient for these users, and enable use of the Gnome 2 desktop for some users who otherwise would have no access to Gnome.

Gnopernicus provides two main services:

1. *focus tracking mode*: the user is always informed about the current focused object on the screen
2. *flat review mode*: the user can obtain information about the widgets in the current window "line by line", so he/she can have a "global view" in this way.

Another important and useful feature provided by Gnopernicus are the keypad layers. Gnopernicus functions are mapped on fixed keys on numeric keypad. Because the number of keys on numeric keypad is limited, the "layer concept" was introduced. That means that a key can have multiple meanings, depending on which layer the user is. Now there are 11 layers available, from 0 to 10. To switch between them you have to press 0 key from numeric keypad, followed in 5 seconds by the layer number, or DEL key for layer 10.

Gnopernicus is built from more modules, which are shown in the following picture:

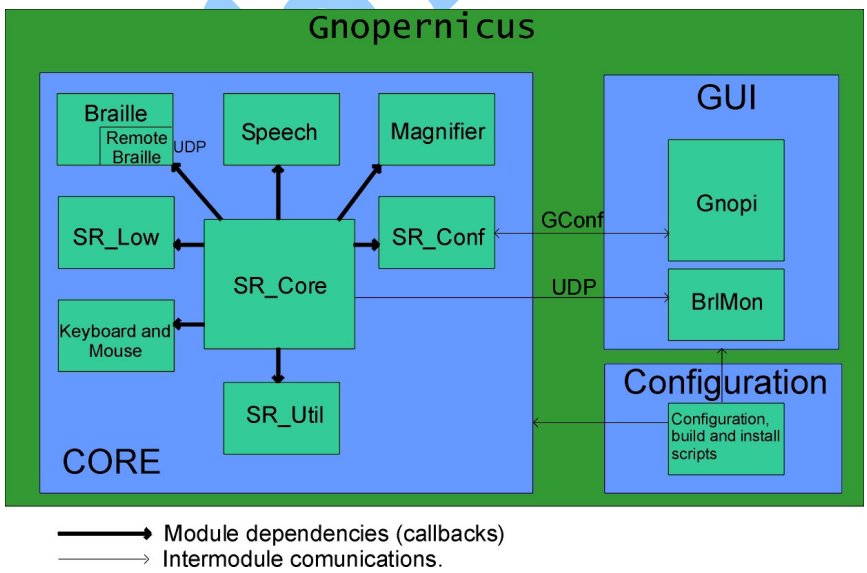

fig. 3. *Gnopernicus Architecture*

SRCore component

SRCore is the component of Gnopernicus Screen Reader that gathers information from all input sources, organize it and then present it to the three outputs: Braille, Speech, Magnifier. It is also aware about user settings and configuration, and notifies the other modules about any change that occurs. Also, the user can use the Braille displays sensors, and SRCore knows how to react to this, and to offer the requested information.

SRUtil component

SRUtil is designed to provide an uniform way to communicate between modules. Modules are communicating one with other via events. This module defines all event types that can appear and provides a structure, which encapsulates all necessary dates for processing an event.

SRLow component

SRLow component is module that listens all the systems events and generates his own events as response to the user actions. For every of generated events it provides information for the object with which user interact. All these information let the user get a feedback from the UI of an application.

SRConf component

It has a role of communication proxy between CORE and Gnopi. It make possible that the two packages could be communicate with each other.

Braille component

The Braille component is a SO (shared object) which accepts input in BML (Braille Markup Language, a XML dialect developed at Baum) and are able to provide the output on many known Braille devices. Braille is a medium, which allows a non-sighted person to read text by touch, is also a method for writing tactile text. This component makes a translation from a screen object in to the readable Braille code set. The Braille code is physically presented as raised dots, usually arranged in cells of up to 6 (or 8) dots. The module can be configured from a Gnopi user interface.

Magnifier component

Screen Magnifiers are assistive technologies that enable users with low vision to enlarge the computer screen and interact with the magnified portion of the screen they are accessing, such as menus, toolbars, graphics, and text.

In the gnopernicus project the magnifier technology is offered through the **magnifier**–named module. The magnifier can be used by user (visually impaired) with the help of the GUI (Graphical User Interface), SRCore being not just a proxy, but a logic component that helps interaction between the two components and takes decisions in how the information will be presented.

Speech component

Screen readers are for people who are blind, but they can be useful even to the visually impaired. These aids make on-screen information available as synthesized speech or a refreshable Braille display. They can only translate text-based information. Graphics can be translated if there is alternative text describing the visual images.

The synthesized speech makes use of TTS (Text To Speech) Engines. **TTS** is the creation of audible speech from computer readable text. The information that is important for the blind user can be splited into two parts (from the presentation point of view): some information can be provided throw the Braille displays or can be spoken by an engine. These parts can be combined for a better presentation.

In the gnopernicus project the speech feature of the Screen Reader is offered through the speech–named module. The speech can be used by user (blind or even visually impaired) with the help of the GUI (Graphic User Interface), **SRCore** being not just a proxy, but a logic component that helps interaction between the two components and takes decisions in how the information will be presented.

Libke component

Libke contains the keylistener code. The key listener translates the key or key-combinations to a string, in order to allow the screen reader to execute the required command. This is an important part for a screen-reader, because there are a lot of commands (i.e: say the current window title, increase the speech volume, scroll the text on the Braille display with one display width etc.) mapped on some keys/key combinations.

The most interesting and useful part is that the user has the possibility to map the commands on its own key combinations, even there is a default commands distribution in gnopernicus.

Gnopi component

Gnopi is the configuration user interface for Gnopernicus. Like functionality the component has a role to configure and to save the own

settings for a different component. The module is in **GUI** (Graphic User Interface) package, and this module gives the executable file for the whole Gnopernicus.

BrlMon component

It is the Braille device debugger and simulator. It shows in character text format the list of symbols from Braille device. This component makes part from GUI package. It works like a client/server program, at which the client is the CORE and the server is the BrlMon.

4 Conclusion

- **GNOME Accessibility + Assistive Technologies = Productivity Gains**

The growing popularity of the GNOME desktop is driving the need for accessible desktop software for UNIX and Linux platforms, allowing organizations to provide functional solutions for their employees with disabilities. In addition, the U.S. government and its agencies now require that its suppliers provide accessible products, as stated in Section 508 of the U.S. Federal Rehabilitation Act.

Using the open Assistive Technologies Service Provider API to develop the solutions quickly, because it eliminated the need to reinvent assistive technology software for different UNIX platforms, saving us time and costs".

References

1. **Jeff Witt**: *The PC's Unfulfilled Promise. How Most Blind and Visually Impaired People Are Left Behind*
2. **GNOME**: *http://www.gnome.org/, http://www.gnu.org/*
3. **Accessibility**: *http://developer.gnome.org/projects/gap/, http://developer.gnome.org/doc/*
4. **ATK/AT-SPI**: *http://developer.gnome.org/projects/gap/tech-docs/, http://developer.gnome.org/doc/API/2.0/at-spi*
5. **GConf**: *http://developer.gnome.org/doc/API/gconf/c21.html*

6. **GTK**: *http://www.gtk.org/docs/gtk.html*
7. **BAUM**: *www.baum.de, www.baum.ro*