Orca Screen Reader Programmer’s Guide

Willie Walker
Sun Microsystems, Inc.
Accessibility Program Office

Orca Screen Reader Programmer's Guide

by Willie Walker

Revision History

Revision 0.2 03 Mar 2005 Revised by: wdw
More requirements based on discussion with Peter Korn
Revision 0.1 16 Feb 2005 Revised by: wdw
First release - Architecture based on Marc Mulcahy's Prototype

Table of Contents

Chapter 1. Introduction

The purpose of a screen reader is to provide a non-visual interface to applications. While the interface tends to be primarily for "viewing" only, screen readers sometimes also provide alternative means for navigating the screen and providing input to applications.

A screen reader's primary use is as an assistive technology for people with visual impairments. In this respect, the non-visual access tends to be in the form of speech synthesis and refreshable Braille displays. The control of the screen reader tends to be through the system keyboard, but may also include the input keys on a Braille display. In addition, screen readers often coexist and collaborate with other assistive technologies, such as screen magnifiers.

Orca is intended to be a screen reader for the GNOME platform. This document discusses the requirements, functional specification, and architecture of Orca.

Intended Audience

The intended audience for this document are the developers of Orca as well as developers wishing to extend Orca through custom scripts.

Chapter 2. Requirements

The Orca requirements can be broken into two main areas: those which are necessary for end users of Orca, and those which are necessary for people wishing to extend or enhance the capabilities of Orca via scripts.

End-User Requirements

Orca must supply at least the following end-user features.

Available at All Times

Orca will often be the only vehicle by which many users will be able to access the system. As such, it must be able to be made available at all times, including accessing system login and screen-saver screens.

Track and Present Active Application/Window

Orca must be able to track the currently active window, which is the window that has keyboard focus. When first starting up and when the active window has changed, Orca will provide a short summary of the active window. The presentation of the summary will follow the style guide.

Track and Present Object with Focus

Orca must be able to track the currently active object. The active object may be different things depending upon context, but includes the following: the object with keyboard focus and the text caret.

When first starting up and when the active object, Orca will provide a short summary of the active object. The presentation of the summary will follow the style guide.

Permit Presentation of Information from Other Applications/Windows

Orca must also allow for the presentation of information of applications/windows that do not have keyboard focus. This will typically be driven from AT-SPI events from the application/window as well as input device events. The presentation will typically be information that the user has optionally requested. Examples of such information include announcing the time of day, announcing the subject/sender of incoming e-mail, etc.

Configurable Presentation and Interaction

Different users have different capabilities (e.g., some may be able to hear some synthesis voices better than others; some may use Braille while others do not) and desires (e.g., some may prefer faster speaking rates). As such, the general manner and means by which Orca presents information to the user must be configurable by the end user.

TODO: need to define the configurable parameters. This not only include the use of speech/mag/Braille, but also how they are used. The prior paragraph touches on this, but it also includes things such as what is to be spoken and the order in which they are spoken (e.g., role, label, text).

It is desirable, but not required, to allow the user to change (and test) configuration settings while Orca is running.

Speech Synthesis

One of the primary non-visual ways to present a graphical display to a user is via speech synthesis. Note that Orca need not be a speech synthesizer, but it must be able to drive one. The most important functionality Orca needs for speech synthesis are as follows:

- *General Synthesis* - Orca must be able to speak an utterance (e.g., a word, a label, a sentence, etc.) or set of utterances (e.g., a paragraph or entire document). The utterance being spoken must be able to interrupted at any time.
- *Voice Styles* - Orca must be able to identify and allow the user to select between the voices available on the various synthesis engines available on the system. In addition, Orca must allow the user to customize parameter settings for the voices, such as average pitch, speaking rate, and volume. The combination of {voice, pitch, rate, volume} will be known as a "voice style." Orca will provide the user with the ability to select the voice styles to be used for various speaking operations (e.g., default, uppercase, warning, alarm, etc.). At a minimum, Orca will support a "default" style to be used for the majority (if not all) of the speaking operations.
- *Speaking Rate Modification* - While the speaking rate will generally remained fixed once configured, users may sometimes which to speed up or slow down the presentation of speech. While it is ideal that user can do this while the synthesis engine is speaking, such functionality is typically not provided by the majority of speech synthesis engines. Orca must, however, allow the user to change the speaking rate that will be used for the next utterance, should the underlying engine support changes to speaking rate.
- *Multilingual Text* - Orca should provide the ability to speak a single utterance that contains words or phrases from multiple locales. This is an emerging area for speech synthesis engines, however, so Orca will support this feature if the underlying engine(s) support it.
- *Spelling Mode* - Orca must be able to optionally spell out words, either letter by letter or by military (e.g., alpha, bravo, charlie) spelling.
- *Verbalized Punctuation* - Orca must be able to optionally verbalize punctuation.
- *Repeated Character Count* - Orca must be able to optionally compress the repetition of character by saying something such as "25 dashes" instead of "dash dash dash dash dash..."

Refreshable Braille

Braille is another primary non-visual presentation mode for screen readers. As with speech synthesis, Orca need not directly support a Braille display, but it must be able to drive one. The primary end requirements for a Braille display are as follows:

TODO: Obtain the requirements needed for a Braille display. Marc also did some interesting prototype work with using as much of the Braille display as possible to provide as much information as possible (e.g., displaying all items in a menu as opposed to just the one that is currently selected). Given the large variations across Braille displays, this appears to be a fairly in-depth task. In addition, I'm not sure what input events can be generated by a Braille display - this seems to vary as widely as the number of displays. Right now, however, Orca seems to respond to just one input event (onBrlKey), and will merely invoke accessible actions or set cursor positions in response to it. Perhaps the Braille input support will need to mirror the keyboard event handling? In addition, perhaps the Braille input can also be used to drive review mode on the Braille display?

TODO: Look at using Braille display and touch cursors for selection and to relay additional information via speech.

Transparent to Traditional Keyboard Navigation Methods

Orca must allow users to navigate through the desktop and applications on the desktop using the system keyboard navigation gestures (e.g., Alt+Tab to select the next window). In other words, Orca must not interfere with traditional keyboard navigation.

Key Echo

Orca must provide a feature that optionally speaks the name of the key that the user just typed. A key being spoken must be able to be interrupted at any time.

Text Echo

Orca must provide a feature that optionally speaks text insertion and deletion events, typically echoing the word just typed or deleted. Text being spoken must be able to be interrupted at any time.

Command Echo

Orca must provide a feature that optionally speaks the command that was invoked when the user pressed a key that is to be interpreted by Orca. This feature is generally used as a learning feature to permit users to learn the effects of Orca key bindings. A command being spoken must be able to be interrupted at any time.

Review Mode

Orca must provide the ability for a user to review the contents of the desktop or a given application. This is typically done by the user making a single well-known keyboard gesture indicating "review," with Orca reacting by presenting the area to be reviewed. At any time, the user can interrupt the review mode, causing presentations such as speech output to stop immediately.

Another option for review mode includes the ability to use a set of well-known keyboard gestures to quickly skim the contents of the display.

The review of the desktop will follow the style guide, and will at least provide a short summary of the available applications. While the review of an application will also follow the style guide, the overall presentation depends largely upon the application being reviewed.

TODO: Add notions of "flat review," "hierarchical review," and "hybrid review (in/out of lists)".

Where Am I?

Orca will provide a means for a user to determine where the current focus is on the desktop, including which object has focus, which window it is in, which application, which workspace, etc.

List All Applications

Orca will provide a means for a user to determine all applications running on the desktop as well as all open windows on the desktop.

Watched Objects

Orca will allow users to indicate interest in an object. Once interest has been given, Orca will notify the user of changes to that object, whether it or its window has focus or not.

Bookmarked Objects

Orca will allow users to indicate interest in an object. Once interest has been given, Orca will provide a means for a user to quickly give that object keyboard focus.

Search for Objects

Orca will allow users to enter a string to search for an object in the current window or entire desktop. If the object is found (i.e., the string matches the accessible name), the object (and it's window if necessary) will be given keyboard focus.

Search for Text Based on Attribute

Orca will allow the user to search for the next occurence of a given attribute (e.g., italics) in a region of text.

Document Reading

Orca must provide the ability for a user to read the contents of a document, such as e-mail or a word processing document. As the document is read, Orca will instruct the associated application to scroll so as to keep the portion being read visible on the screen. The invocation of the document reading will be triggered via a well-known keystroke and may be stopped at any time by the user. *TODO:* Specify behavior - does same keystroke start/stop the reading?

While the reading of a document will follow the style guide, the overall presentation depends largely upon the document being read (e.g., is it a text document, spreadsheet, web page, etc.?).

Customizable Behavior Per Application ("Scripting")

It is expected that the default behavior will provide good access to all applications that use the AT-SPI. However, to provide dramatically improved access, Orca must be able to provide customized behavior for individual applications. For example, one can envision a script for an e-mail application that can provide prioritized access to one's inbox. Another example may be that the script provides keyboard access to select and copy displayed text to the system clipboard in the event the application doesn't support this (e.g., the only way to select text in a terminal window is to use a mouse - a script might create new keybindings to allow a user to do this from the keyboard).

Co-exist Seamlessly with Other Assistive Technology

It is not uncommon for users to use other assistive technologies, such as screen magnifiers, simultaneously to access their displays. As such, Orca must be able to co-exist (i.e., not interfere) with other assistive technologies in use by the user.

Drive Region of Interest for Screen Magnifiers

In addition to co-existing with each other, a screen magnifier and a screen reader must be in sync with each other. That is, while the screen reader is presenting a portion of the display, the screen magnifier must also be showing that portion of the display.

A stand-alone screen magnifier can simply track keyboard focus and other such events, allowing it to operate somewhat independently of a screen reader. A difficulty arises, however, when a screen reader is reviewing an area of the screen that is larger than the screen magnifier can display at once. In this case, the screen magnifier needs to know that the region of interest has changed, and the screen reader is the one that knows the new region of interest.

Note that Orca must not only be able to detect the existence of a screen magnifier when it starts up, but it must also be able to detect if a screen magnifier has started at some point in the future.

Acceptable Response Time

The screen reader must not degrade the perceptible performance of the system. That is, user should be able to detect any decrease is responsiveness of the desktop when the screen reader is being used. In addition, a user's interaction with the screen reader should appear as crisp and as lively as normal interaction with the display via traditional interfaces (e.g., the keyboard).

Since speech synthesis will often be the primary presentation mode of Orca (Braille is the other), acceptable performance of the speech synthesis output is very important. Orca must be able to provide speech synthesis that meets or exceeds the following performance metrics:

- *Time to First Sound* - The time between when a speech synthesizer gets a request to speak and when the synthesizer actually starts speaking must be minimal (e.g., 30ms).
- *Time to Cancel* - Orca must be able to cancel speech synthesis in progress, and the time to cancel must be minimal (e.g., 30ms). Furthermore, the time between when a cancel is issued and the time the next utterance is to be spoken must be minimal (e.g., 30ms).

TODO: Determine reasonable Braille response times.

> **Performance Scope:** It is understandable that much of the response time may be due to other factors such as the response time of Bonobo and the underlying speech engine, etc. As such, the primary responsibility of Orca and each application script is to process AT-SPI and keyboard events as quickly as possible.

Failure Resilient

In the event that Orca fails or a component of the system that Orca depends upon fails, the system should be able to heal (and perhaps restart) itself appropriately.

Consistent Style

It is expected that access to applications will be driven primarily via customized "scripts," with a fallback ("default") script to be used in the absence of a customized script.

While each script can provide dramatically different access to an application, it is expected that scripts will provide a users with a consistent style to access applications. Orca will provide the style guide, a well-documented "default" script, and several application scripts that demonstrate and promote this style.

Documentation and Tutorials

Although it is a reasonable goal that Orca should attempt to achieve, users cannot be expected to be able set up and use Orca without documentation. Like other systems, such as JAWS, Orca must provide documentation and tutorials on the installation, configuration and use of Orca. This documentation must come in form(s) that are accessible to people who need to use the screen reader (e.g., accessible text and audio).

Script-Writer Requirements

The following requirements are needed for Orca to be an effective and extensible system.

Extensible by Third Parties

Orca is to be an extensible platform to which third parties can add value to without the need to modify the core system. The primary method for extension will be the ability to add customized scripts. Orca will detect these scripts in a well-known location {*TODO:* TBD}, and will also be able to dynamically load new and modified scripts at run time.

Ease of Script Development

In order to attract script developers and to create a thriving script developer community, the creation of scripts for Orca must be a relatively simple and well documented process. Ideally, the learning curve should be minimal and a skilled developer should be able to start writing scripts after approximately one day of learning.

The script development environment must also include debugging and profiling tools to aid in script development.

Script-Writing Specification and Examples

While many developers will likely develop their scripts by example, Orca must also provide the specification for a script in order to document the supported behavior. In addition, the "default" script for Orca will serve as an example of appropriate style, and must be well documented.

Online Script Repository

To also help foster a thriving community of script developers, there should be a publicly accessible repository for people to contribute and access scripts.

Access to AT-SPI Objects and Events

Because the AT-SPI will be the means by which Orca obtains information about the desktop and its applications, and will also be the means by which Orca will manipulate (e.g., activate buttons) objects on the desktop, script writers need access to AT-SPI objects and events.

Intercept and Interpret Keyboard Events

The keyboard will be a primary method for the user to interact with Orca. Orca may introduce new keyboard gestures, such as those for review mode, that should not be interpreted by applications. As such, Orca will need to be able to intercept and interpret keyboard events and potentially prevent them from reaching the GNOME desktop.

Orca must also not interfere, however, with other assistive technologies such as the AccessX functionality of XKB [*XKB*>].

Chapter 3. Style Guide

TODO: this section is under construction.

TODO: Include "where am I?"

TODO: Include what to do when events come from objects in windows that do not have focus.

TODO: Include what to do when an object has more than one kind of accessible action.

Default Presentation of Individual Objects

Orca's goal is to present individual objects with a consistent style for both speech and Braille. The default.py module contains the logic for this consistent style, which custom scripts are welcome to override, extend, or model their presentation after.

The default script for Orca provides a number of "presenter functions", each of which specializes in presenting an object based upon its role and whether or not it just received focus. The default script also provides a fallback presentation function (the "default presenter") to handle the case where a more specialized presenter function doesn't exist.

The mode of presentation (e.g., speech or Braille) determines how the script presents the object. In the case of speech, the primary goal is to succinctly present the most useful information early followed by more detailed information. At any time, the user can interrupt the speech output and/or move on to the next object of interest. Although the speech output may be ungrammatical at times, this presentation style permits the user to navigate a GUI using speech in a quick fashion, yet still be able to learn more detail about an object of interest.

In the case of Braille, the primary goal is to make effective use of the potentially limited number of Braille cells on the refreshable display while also providing a consistent spatial location for common information. For example, the role of the object being presented will typically be represented as an abbreviation in the first three cells of the Braille display. Like speech, the goal is to be able permit a user to navigate the GUI using Braille in a quick fashion, yet still be able to learn more detail about an object in question.

TODO: The discussion so far treats speech and Braille as separate systems. There is ample room for research on how the two can be used to provide complimentary access.

When a typical Orca script receives an event from the AT-SPI, it determines the object associated with that event and chooses whether to present the object or not. The choice to present typically involves the type of event and whether the associated application has focus or not. If it chooses to present that object, the typical Orca script will initially structure its presentation of the object on two factors: the role name of the object and whether or not the object just received keyboard focus.

When an object first gets keyboard focus, the speech output of the default presenter is as follows for the en_US locale:

```
label role [value] ["shortcut" shortcut] ["accelerator" accelerator]
```

Where:

- Specific text to be presented is contained in quotes.
- At most one space character is used between the elements.
- Optional elements are contained between the characters "[" and "]". For example, if an object does not have a value or a shortcut does not exist, then Orca will present nothing for the value or shortcut.

- *label* is the first non-empty string obtained by first looking at the label of the LABELED_BY object for this object (if it exists), then the accessible name of this object, and finally the accessible description of this object.
- *role* is the localized human consumable string describing the AT-SPI rolename of this object as determined by the rolenames.py module.
- *value* is the value of this object, if the value exists. Note that the value depends upon the role of the object. For example, for check box items, the value will be one of "checked" or "not checked." For single line text fields, the value will be the line of text. Please see the specific sections below for how the value is obtained from individual objects.
- *shortcut* is the string representing the keyboard shortcut to activate the object (e.g., "Alt f n").
- *accelerator* is the string representing the keyboard accelerator to activate the object (e.g., "Control n").

If an object's value changes while it already has keyboard focus, then the only thing spoken will be the new value.

For Braille, the behavior of the default presenter is as follows for the en_US locale:

```
TRN label [value] ["(" shortcut ")"] ["(" accelerator ")"]
```

Where:

- Label, shortcut, and accelerator are the same as in the speech output.
- *TRN* is the object's "three letter rolename" as determined by the rolenames.py module.

For refreshable Braille displays with cursor buttons (i.e., buttons on each cell), the default Braille presenter will also position the Braille cursor under the first cell of the value if it exists, or the first cell of the label if the value does not exist.

The following sections describe the specialized presentation functions, where the AT-SPI role name for which they are specialized is listed in parentheses in the section title. The notation in the following sections is the same as above, with the addition of "{" and "}": mutually exclusive alternative elements are contained between the characters "{" and "}". For example, given the alternative list of {"available", "unavailable"}, Orca will present one or the other to the user.

invalid accelerator label alert animation arrow calendar canvas color chooser column header dateeditor desktop icon desktop frame dial dialog directory pane drawing area file chooser filler fontchooser frame glass pane html container icon image internal frame label layered pane list list item option pane page tab page tab list panel password text popup menu progress bar root pane row header scroll bar scroll pane separator slider split pane spin button status bar table table cell table column header table row header terminal text tool bar tool tip tree tree table unknown viewport window header footer paragraph application autocomplete edit bar embedded component extended hyper link link multi line text single line text table line table columns header title bar tree item

pushButtonPresenter ("push button")

The pushButtonPresenter presents objects of role "push button" and does so identically to the defaultPresenter. No value will be presented. The pushButtonPresenter will place the Braille cursor under the first character of the button's label; pressing any cursor button associated with the label will cause a click action on the associated button.

toggleButtonPresenter ("check box", "toggle button")

Speech:

```
label role {"checked", "not checked"} ["shortcut" shortcut] ["accelerator" accelerator]
```

If the button’s value changes while it has focus, the presenter will only speak the new value.

Braille:

```
TRN {"-", "="} label ["(" shortcut ")"] ["(" accelerator ")"]
```

The unchecked value of the button is represented by "-", which is dots 3-6 in Braille, and the checked value is represented by a "=", which is dots 1-2-3-4-5-6 in Braille. The toggleButtonPresenter will place the Braille cursor under the first character of the button’s value ("-" or "="); pressing any cursor button associated with the label or the button’s value cause a click action on the associated button.

radioButtonPresenter ("radio button")

Radio buttons occur in radio button groups. As such, knowing both the name of the radio button group and the number of buttons in the group is important.

Speech:

```
group label role {"checked", "not checked"} ["shortcut" shortcut] ["accelerator" accele
```

Where group is the name of the radio button group.

Braille:

```
TRN group ("(" {"’", "7"} label")")*
```

With this, entire radio button group is presented on the Braille display, with each radio button surrounded in parentheses. The unchecked state of a radio button is represented by a "’" which is dot 3 in Braille, and the checked state is represented by a "7", which is dots 2-3-5-6 in Braille (CBC). The Braille cursor is placed under the radio button with keyboard focus. If the user presses any cursor button associated with a radio button, it will cause a click action on that button.

TODO: Would like to also present the shortcut and accelerator somehow. Perhaps this could be one of those things where speech compliments the Braille? Or, perhaps we can take advantage of Braille displays that have two buttons per cell to do this (i.e., pressing the other button causes Orca to speak the extra info)?

menuBarPresenter ("menu bar")

It is rare that the menu bar alone will be presented. Instead, the menus in the menu bar will be presented. However, in the event a menu bar is to be presented, it will be presented as follows.

Speech:

```
role (menulabel)*
```

Where menulabel is the label of a menu in the menu bar.

Braille:

```
TRN ("("menulabel")")*
```

With this, entire contents of the menu bar is presented on the Braille display, with each menu surrounded in parentheses. The Braille cursor is placed under the menu with keyboard focus. If the user presses any cursor button associated with a menu, it will cause a click action on that menu.

menuPresenter ("menu", "menu item", "check menu item", "radio menu item", "tear off menu item")

Like menu bars, menus and their associated contents will be treated as a related group of objects.

Speech for "menu":

```
label role ["shortcut" shortcut] ["accelerator" accelerator] n " items. "
```

Where n is the number of keyboard accessible items in the menu.

Speech for "check menu item", "radio menu item", and "tear off menu item":

```
label role ["shortcut" shortcut] ["accelerator" accelerator]"
```

Braille:

```
TRN label ("(" {"", "o", "___", "-", "=", "’", "7"} itemlabel")")*
```

With this, entire menu is presented on the Braille display, with each menu item surrounded in parentheses. "Normal" menu items are merely presented using their label. Menus within menus are prefixed with an "o" to indicate they are menus (NOTE: you can view the "o" as meaning "openable" or view it as a triangle shape - dots 1-3-5 in Braille). Tear off menus have no label and are represented by "___". Check menu items are prefixed by their value, "-" (unchecked) or "=" (checked). Radio menu items are prefixed by their value, "’" (unchecked) or "7" (checked). The Braille cursor is placed under the menu item with keyboard focus. If the user presses any cursor button associated with a menu item, it will cause a click action on that item.

TODO: Would like to also present the shortcut and accelerator somehow. Perhaps this could be one of those things where speech compliments the Braille? Or, perhaps we can take advantage of Braille displays that have two buttons per cell to do this?

comboBoxPresenter ("combo box")

Like menus, combo boxes and their associated contents will be treated as a related group of objects.

Speech:

```
label role value ["shortcut" shortcut] ["accelerator" accelerator]
```

Where value is the text of the current selection. As the selection changes, only the the text of the new selection will be spoken. %todo; Need to be able to tell whether the combo box has editable text or not.

Braille:

```
TRN label ("("itemlabel")")*
```

With this, the entire combo box is presented on the Braille display, with each item surrounded in parentheses. %todo; Need to worry about combo boxes with lots and lots of entries. Also need to indicate if a combo box has editable text or not.

Chapter 4. Functional Specification

Orca's functional specification is driven by the requirements.

Prerequisites

To narrow the scope of Orca, Orca will use existing software where available. For example, as mentioned in the requirements, Orca is a screen reader that needs to be able to interact with speech synthesis, Braille, and screen magnification services, but it need not be the provider of such services. Given this, Orca has the following dependencies.

gnome-speech v0.3.4 or better

With respect to speech synthesis, there is an existing component of the GNOME platform that provides fundamental speech synthesis support: gnome-speech [*Gnome-Speech*>]. The gnome-speech component provides a CORBA-based approach to access speech synthesizers as network services. Because gnome-speech appears to meet the underlying speech synthesis requirements of Orca, Orca will depend upon gnome-speech for TTS support.

brltty v3.6.1 or better

With respect to Braille, there is an existing package that provides fundamental Braille I/O support: brltty [*BRLTTY*>]. Brltty provides access to a variety of Braille displays, and consists of a library and a daemon to provide programmatic interaction with the display. Because brltty appears to meet the underlying Braille requirements of Orca, Orca will depend up brltty for Braille support. *TODO:* There is also gnome-braille which might be more in line with gnome-speech and gnome-mag.

gnome-mag v0.11.11 or better

With respect to magnification, there is an existing component of the GNOME platform that provides fundamental screen magnification support: gnome-mag [*Gnome-Mag*>]. The gnome-mag component provides a CORBA-based approach to access and manipulate a screen magnifier as a network service. Because gnome-mag appears to meet the underlying screen magnification requirements for Orca, Orca will depend up gnome-mag for screen magnification support.

at-spi v1.6.2 or better

Orca's means of gathering information about the desktop as well interacting with the desktop will be done through the AT-SPI [*AT-SPI*>]. As such, a functioning AT-SPI environment is mandatory. The AT-SPI provides a CORBA-based approach to detect, examine, and manipulate desktop and application content. It supports the registration of event listeners for changes to desktop and application content. Finally, the AT-SPI supports the registration of listeners for input device events, with an option for these listeners to intercept (and possibly consume) the events before they are processed by the desktop or applications on the desktop.

python v2.4 or better

Orca will be written in the Python programming language, and will depend upon features found in Python versions 2.4 and greater. *TODO:* Verify this. We may be able to get by with earlier versions.

Configuration

Orca will provide a utility (e.g., a command-line script and/or GUI) that permits the end user to configure at least the following features. Note that the configuration utility must be accessible to the end user (e.g., via speech synthesis and/or Braille).

- *Speech Synthesis* - The use of speech synthesis will be enabled by default, but some users (e.g., those that solely use Braille) may not want to use it.
- *Voice Styles* - Orca will, at a minimum, permit the user to specify the default voice to be used for the majority of presentations that use speech synthesis. The default voice not only includes the speech synthesis driver and speaker to be used, but also includes the average pitch, rate, and volume parameters for the voice. If available, Orca will also identify and allow the user to configure other speaking styles (e.g., "upper case," "warning," etc.).
- *Text Generation Template* - Orca will provide the user with the ability to select (and perhaps define) the template that will be used for generating the text to be spoken. For example, the user may wish to have a template of "{object role} {object label}" for speaking buttons. *TODO:* this could be an endless pit of customization. We need to define the things that are of use to most people.
- *Key Echo* - If enabled, Orca will speak each key that the user presses.
- *Text Echo* - If enabled, Orca will speak each word that the user types.
- *Command Echo* - If enabled, Orca will speak each command that the user types.
- *Braille* - If a Braille display is available, Orca will offer the option to use it.
- *Screen Magnification Synchronization* - If Orca detects a screen magnifier is present, this feature will determine if Orca drives the region of interest of the screen magnifier. Furthermore, if a screen magnifier is started after Orca is already running, then Orca will detect this and synchronize with the screen magnifier. *TODO:* Should the end-user configuration include the option to start Orca and the screen magnifier at the same time? If so, is this part of the Orca configuration or part of some other system setting?

Whenever the configuration utility is run, the configuration parameters will be saved.

Behavior

This section describes the overall behavior of Orca. Details about the architecture and implementation can be found in the architecture section.

Start Up

When Orca is run, it will check for user settings. In the event that the user has not created any specific settings, Orca will fallback to the default settings. If the user wishes to customize Orca, they will need to run the configuration utility, which guides them through the Orca configuration process.

Orca will then present a localized message to the user that is has been activated (e.g., "Welcome to Orca version 1.0").

Orca will then "attach" to the desktop via the AT-SPI, obtaining information about the applications on the desktop. As it discovers each application, Orca will instantiate a custom script for each application.

TODO: Define location and naming convention for scripts.

The custom scripts will typically extend/override the behavior of the "default" script by providing specialized handlers for specific AT-SPI object and input device events (keyboard and Braille).

After obtaining desktop information and creating custom scripts for each application, Orca will then detect the currently active window and use its associated script to present a brief summary of the window's content following the guidelines in the style guide.

Use

As the user navigates around the desktop and within applications using "normal" keyboard navigation gestures, Orca will receive events (e.g., focus, window activation, etc.) from the AT-SPI. It will associate each event with a custom script and pass that event onto the script for handling. The typical script will present the information to the user according to the guidelines in the style guide.

As applications come and go on the desktop, Orca will be notified via the AT-SPI, and will create/remove custom scripts for those applications accordingly.

In addition, as the user types on the keyboard, Orca will receive the keyboard events prior to their interpretation by the desktop. Orca will pass the event on to the active script for interpretation. The script may consume the event, preventing it from being interpreted by the desktop.

Orca itself may also provide global keyboard bindings to control its behavior, such as the following: changing speaking rate and volume, entering/exiting review mode, navigating during review mode, etc. *TODO:* Need to specify how to do this; perhaps through a "global script" that Orca itself maintains?

Finally, the user may use keys on their Braille device to drive Orca. Orca will receive these events and process them in a manner similar to the way it manages keyboard events.

If at any time Orca detects a failure in any component that it depends upon, it will announce this to the user if possible. Orca will then attempt to "heal" itself and move on if appropriate (and possible). In the event that Orca itself suffers a catastrophic failure, the system should be able to detect this and restart Orca.

A user will typically use Orca during the entire time the X server is running, including during system login as well as when screen saver is running. Thus, in addition to providing access to the "normal" desktop, Orca will also provide access to these special screens.

Reasonable Access to Key Desktop Apps

Related to customizable behavior is the need to provide reasonable access to "key" applications. That is, while Orca's default script will generally provide some level of access to applications, it is highly desirable to provide compelling access to key desktop applications. The list of these key applications has yet to be identified, but include the web browser, e-mail application, collaboration tool, calendar, word processor, and terminal.

Furthermore, Orca must also provide documentation and tutorials on how to access and use these key applications.

Chapter 5. Architecture and Implementation

As illustrated in the Orca architecture diagram, the main components of Orca are as follows: interaction with desktop applications that support the AT-SPI, interaction with system services (e.g., speech, Braille, magnification), Orca itself, and Orca extensions.

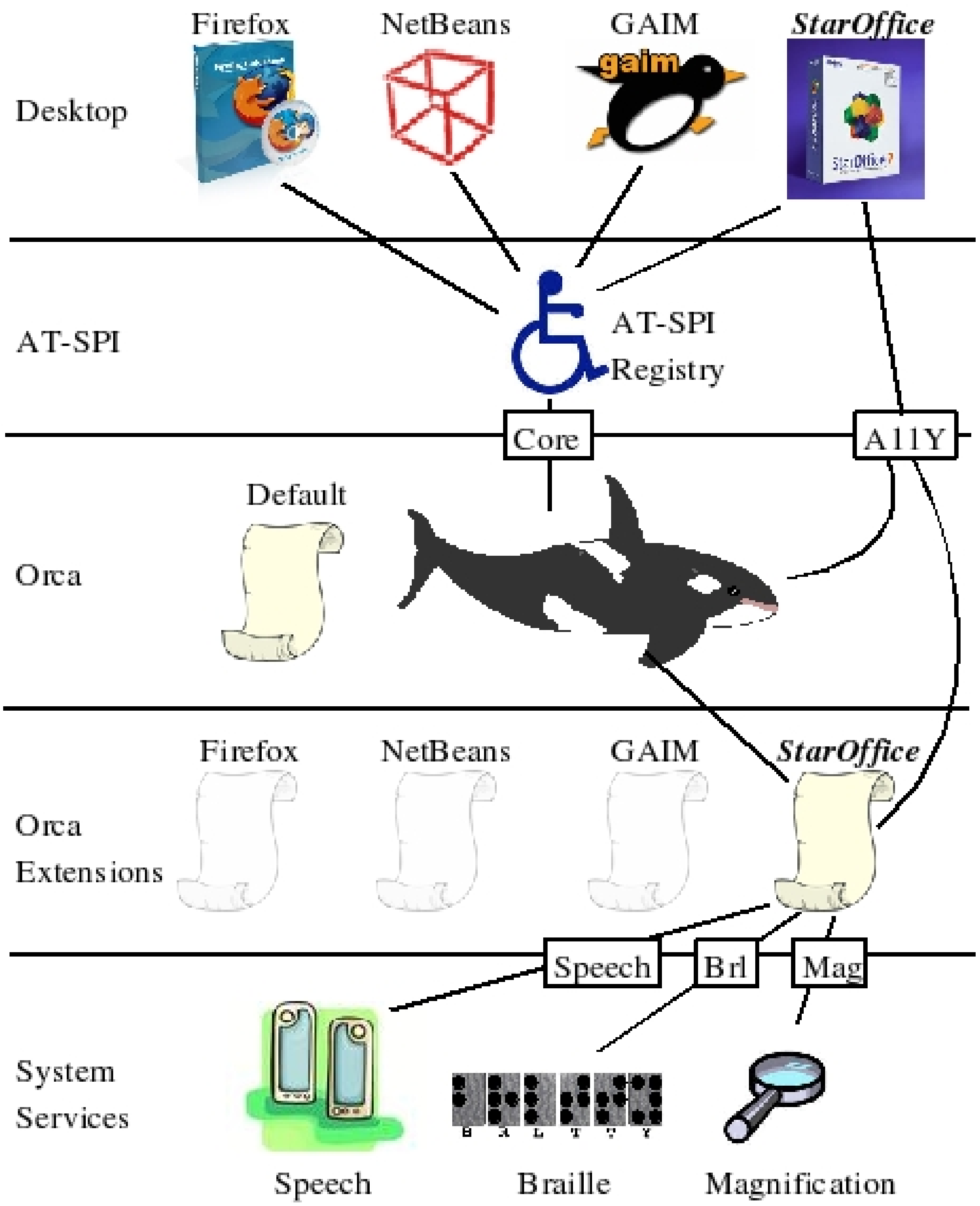


Figure 5-1. Orca Architecture Diagram. The main sections are the desktop applications, AT-SPI, Orca, Orca extensions (scripts), and system services. The key communication between the components is depicted.

The following sections describe the architecture in more detail.

Desktop and AT-SPI

Orca's sole view of any application on the desktop is via the AT-SPI [*AT-SPI*>]. The AT-SPI is a Bonobo-based technology [*Bonobo*>] that provides a common interface for the desktop and its applications to expose their GUI component hierarchy to assistive technologies such as Orca. AT-SPI support is provided by toolkits such as GNOME's GTK+ toolkit (via gail [*GAIL*>]), and has been created for the Java platform as well as the custom toolkits used by applications such as Mozilla and Open Office.

Orca's interaction with the AT-SPI is managed through the following Orca modules:

core

The *core* module (Python support written in C) provides Orca's Python interface to the system's AT-SPI Registry.

The core module provides Orca with the ability to enumerate the various workspaces and applications running on the display, as well as determine the existence of various physical heads (e.g., dual headed system). This also gives Orca the ability to register/deregister listeners for AT-SPI object (e.g., window activation, focus changed, etc.) and device (e.g., keyboard) events. In addition, the core also provides Orca's interface to the Bonobo main and main_quit methods, allowing Orca to interact with the AT-SPI.

IMPLEMENTATION DETAIL: To reduce system traffic, the core module will only register a maximum of one AT-SPI listener for any given AT-SPI event type, and will multiplex the notification of AT-SPI events to any listeners registered by Orca.

IMPLEMENTATION DETAIL: Because processing AT-SPI object events can be time consuming, and because the notification of AT-SPI object events is relatively "bursty," the core maintains a queue of AT-SPI object events. Events are added to the queue when they are received, and events on the queue are dispatched via the GLib idle handler. *TODO:* Need to understand relation between the GLib idle handler and the Bonobo main loop.

IMPLEMENTATION DETAIL: Like the AT-SPI object events, the core module only permit a maximum of one device event listener to be registered with the AT-SPI. Unlike AT-SPI object events, Orca must process keyboard events immediately and quickly. As such, the core component will dispatch keyboard events to the keyboard event listeners registered by Orca.

IMPLEMENTATION DETAIL: Because Orca is most often interested in the last of any particular event type to happen (e.g., the last focus event or the last window activated event), there may be an opportunity to compress the event queue. This is an optimization that might be looked at in the future, but will not be done for now.

TODO: The system needs to be able to cope with failure. Right now, if Orca fails while handling a keyboard event, the whole desktop can hang. I'm curious if the AT-SPI-enabled app can heal when the connection to Orca is broken or if there is a timeout? Conversations with Bill Haneman lead me to believe there is no such thing as a timeout for the synchronous form of device event notification.

a11y

The *a11y* module provides an Accessible class (written in Python) that serves as a delegate to CORBA objects that implement the AT-SPI Accessible and Application interfaces. These objects can be obtained in three ways:

1. From the AT-SPI Registry (via the core module)
2. From an AT-SPI Registry event (listener registered via the core module)
3. As a child from another Accessible

The Accessible class permits Orca to obtain information about Accessible objects as well as manipulate those objects (e.g., activate actions, set cursor position, etc.). The a11y module also provides a set of convenience utilities for examining Accessible objects and their children.

The a11y module also maintains a "dispatcher" table that maps AT-SPI event types to function names. For example, this table will map the AT-SPI event type of "window:activate" to the Orca function name of "onWindowActivated". This table is used primarily by scripts, which are described in the next section.

IMPLEMENTATION DETAIL: For efficiency purposes, the a11y component maintains a cache of Accessible objects, copying the values of common Accessible object attributes (e.g., name, description, role, state, etc.) to a local store. As such, no external entity should create an Accessible using the class constructor. Instead, all Accessible instances are created using the a11y.makeAccessible function. This lowers the network traffic of Orca by reducing the need to make repeated CORBA calls for common attributes. The a11y module also registers for AT-SPI object events for the purposes of keeping the cache consistent with the AT-SPI object state.

Orca

The main logic of Orca lives primarily in the orca module, but support is broken into other modules as appropriate (e.g., a kbd module needed to provide convenience classes and methods for keyboard events and keyboard event history).

The following sections describe the orca module and its supporting modules in more detail.

orca

The orca module is the "main entry point" of Orca. It initializes all the components that Orca uses (core, a11y, speech, brl, mag) and loads the user's settings.

At initialization time, the orca module also determines all the applications on the desktop and registers the "windowActivated" function for "window:activate" events (tells Orca when a window receives keyboard focus) and registers the "childrenChanged" function for "object:children_changed:" events (tells Orca when a window is created or destroyed). These functions permit the orca module to keep its apps and scripts attributes up to date.

For each application that it discovers on the desktop, the orca module creates a script instance to be used for handling that application (see the "script" description in the next section).

Finally, the orca module registers the "processEvent" function with the core module to be notified whenever an AT-SPI object event is received. When an AT-SPI object event is received, the orca module determines the application script associated with the event and passes the event onto that script for processing. Note that the event is passed on regardless of whether the application associated with the script has focus or not. That is, it is the script's responsibility to determine if the presentation of the information is dependent on focus.

settings

The *settings* module holds preferences set by the user during configuration. These settings include the following: use of speech and/or Braille, voice styles, key echo, text echo, and command echo (see the requirements and functional specification for details on these features).

When starting, the orca module will first look for the settings module in the user's ~/.orca directory. If this module exists, it will be used. If it doesn't, then the default settings module will be used.

kbd

The *kbd* module provides the keyboard-event handling support for Orca. Keyboard events are handled via two key elements of the kbd module: the "keybindings" attribute and the "onKeyEvent" function.

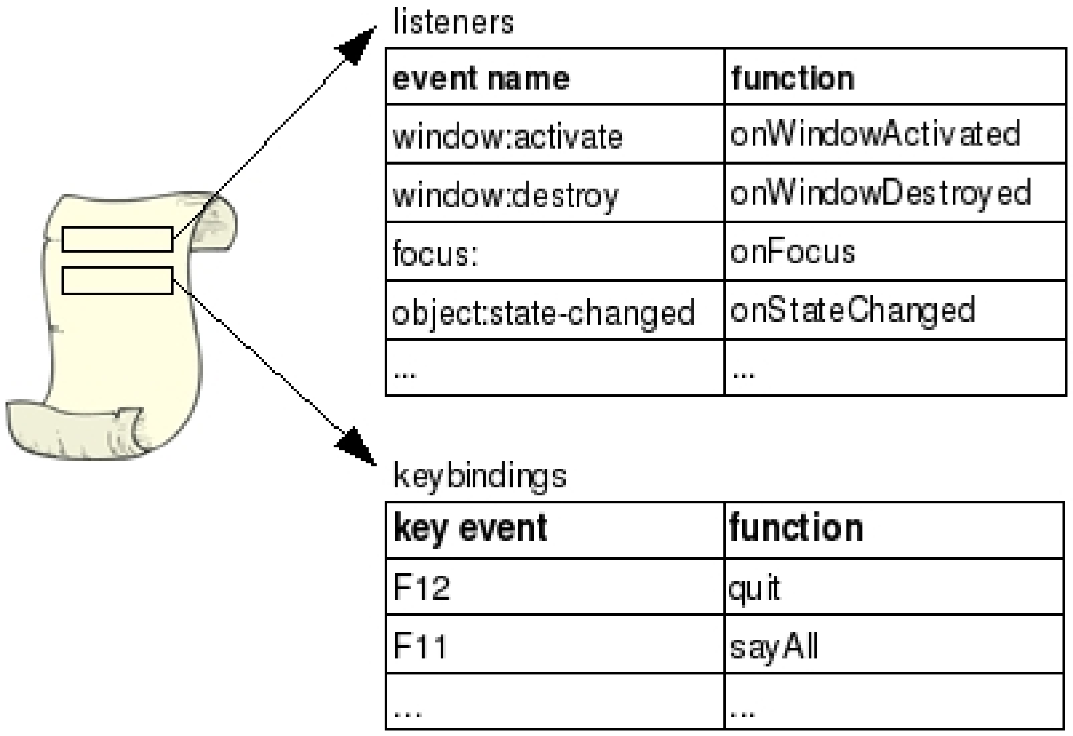
The "keybindings" attribute is a table that maps key event names (e.g., "F12") to functions in scripts. Orca updates this table each time a new script is activated, effectively replacing the existing table with the keybindings table from the script.

When the orca module initializes the kbd module, the kbd module registers the "onKeyEvent" function as the function to dispatch all AT-SPI keyboard events received by Orca. The onKeyEvent also handles the key echo feature of Orca.

The kbd module also maintains a record of the last key pressed, enabling scripts to refer to it for hints about why a particular AT-SPI object event may have happened.

script

The actual presentation of information to the user is done via "scripts," which are instances of the Script class that is defined in the *script* module.



event name	function
window:activate	onWindowActivated
window:destroy	onWindowDestroyed
focus:	onFocus
object:state-changed	onStateChanged
...	...

key event	function
F12	quit
F11	sayAll
...	...

Figure 5-2. Orca Script Diagram. The main components are the "listeners" and "keybindings" attributes.

As depicted in the previous diagram, each script instance keeps track of the following attributes:

- *listeners* - a table that maps AT-SPI object event names (e.g., "window:activate", "object:state-changed", "object:link-selected") to functions
- *keybindings* - a table that maps keystroke events (e.g., "F12") to functions

When the orca module creates a script for an application, it passes the Accessible application object to the Script constructor. When orca then tells the script instance to load itself, the script instance effectively performs the following operations:

1. *initialize "listeners" attribute from the default script* - the script will identify all functions from the default script that are listed in the "dispatcher" table of the a11y module, and then populate the "listeners" attribute of the script with the matching functions that it finds.
2. *override/extend "listeners" attribute using the application-specific script* - the script will then attempt to locate a custom script for the application, based upon the application name (*TODO:* not sure this naming scheme will work across locales). If a custom script can be found, the script will identify all functions in the custom script that are listed in the "dispatcher" table of the a11y module. For each matching function, the script will override or extend the "listeners" attribute accordingly. As such, the custom script will always override or extend the default script.
3. *initialize "keybindings" attribute from the default script* - the script will populate its "keybindings" attribute using the "keybindings" attribute from the default-keybindings module.
4. *override/extend "keybindings" attribute from the application-specific script* - just as it handles the "listeners" attribute, the script will attempt to find a keybindings module for the application. The name of the keybindings module will be the name of the application with a "-keybindings" suffix, and will contain a "keybindings" attribute that defines the keybindings specific to the script. As with the "listeners" attribute, the custom script will always override or extend the "keybindings" attribute of the default script.

TODO: Loading a script is a very efficient and fast process in Python, especially for the expected size of scripts intended for Orca's use. As such, Orca may reload a script each time the window for the script is activated. Alternatively, Orca may provide a global keyboard "refresh" gesture that will reload all scripts in use.

default

The default script is intended to be the base for all scripts that provide customized behavior per application, and is also intended to provide the behavior for when there is no custom script for an application.

As described previously, the default module provides a set of functions whose names match those in the a11y module's "dispatcher" table. Each of these functions, in turn, will call out to a set of "presenter" functions that presents the information to the user. Custom scripts wishing to override the behavior of the default module merely need to define new functions with the same name; the orca module will give preference to the functions in the custom script and will fallback to those in the default script in the event that the custom script has not provided customized behavior.

Associated with the default module is a default-keybindings module, which defines a "keybindings" table to be used by the kbd module as described previously. *TODO:* probably merge this with the default module.

i18n

To be written. Discuss orca_i18n, chnames, and rolenames.

Orca Extensions

Extending Orca's behavior is done by writing two new modules for each application: one to handle the events and presentation of information, and one to define the keybindings table. *TODO:* probably merge these into one module.

As mentioned previously, it is expected that most extensions will extend/override functions in the default module. The primary reason for this is that the default module will typically provide the majority of desired behavior. As such, modules that extend the default module will tend to have most of their work done for them and will only need to focus on those aspects that need specific handling for the application.

Orca extensions will be installed in a "well known place" {*TODO:* TBD}, and can be read and managed by Orca even after Orca has been started.

System Services

Orca relies on existing system services to provide support for speech synthesis, Braille, and screen magnification. To interact with these services, Orca provides the modules described in the following sections.

speech

The *speech* module provides Orca's Python interface to the system's gnome-speech [*Gnome-Speech*>] CORBA service(s). The speech component is used for interacting with speech synthesis engines during both the configuration and use of Orca. The speech component provides methods for the following capabilities:

- *list of available drivers* - several speech drivers (engines) may be available on the system. The speech module permits Orca to identify and interact with each of the drivers. The list of drivers will be available in a human-readable form.
- *list of available voices* - permits Orca to enumerate the available voices for each driver, along with the settable parameters for each voice (e.g., pitch, rate, etc.). The list of voices is available in a human-readable form.
- *voice selection* - permits Orca to select which voice is to be used for speech synthesis, and also permits Orca to set the parameters of the voices.
- *speak text* - provides Orca with the ability to request that text be spoken. Repeated calls to speak text will interrupt and cancel any prior speak operation in progress. The speech module will support Unicode strings.
- *cancel* - the speech module permits Orca to cancel any speech operation in progress.
- *event notification* - permits the registration for notification of speech started and ended events for a a call to "speak," and will also permit for the registration for notification of speech progress events (e.g., word started).

brl

The *brl* module provides Orca's Python interface to the system's brltty [*BRLTTY*>]. daemon. The brltty daemon, in turn, provides the interface to Braille devices for both displaying Braille and receiving input from the user. (Note that the spelling "brl" is used because that is the common English Braille spelling for "Braille.")

TODO: Need to determine the exact interface here.

mag

The *mag* module provides Orca's Python interface to the system's gnome-mag [*Gnome-Mag*>] CORBA service(s). The magnification component provides methods that permit Orca discover screen magnification services and set their desktop region of interest.

Chapter 6. Script Specification

To be written.

Bibliography

Notes

1. http://directory.fsf.org/accessibility/at-spi.html
2. http://lidn.sourceforge.net/articles/gnomenclatureintrotobonobo/
3. http://directory.fsf.org/accessibility/brltty.html
4. http://freshmeat.net/projects/gail/
5. http://directory.fsf.org/accessibility/gnome-mag.html
6. http://directory.fsf.org/accessibility/gnome-speech.html
7. http://directory.fsf.org/accessibility/gnopernicus.html
8. http://www.freedomscientific.com/fsproducts/softwarejaws.asp
9. http://matrix.netsoc.tcd.ie/hcksplat/work/XKBlib.pdf