Design Document

Xdir - X.500 Directory User Agent

Andrew Findlay
Damanjit Mahl
Stefan Nahajski

X500@brunel.ac.uk

Brunel University
M & ES
Kingston Lane
Uxbridge
UB8 3PH
UK

1 Introduction

This document describes the design of a user agent for the X.500 directory service. The design is intended to be implemented as an application running in a windowing environment such as X, NeWS, OS/2 Presentation Manager or MS Windows. It is expected that implementations for X and MS Windows will be produced within the timescale of the project.

The interface design has developed as a result of experience and feedback gained from three prototype DUAs; namely sd, xd and pod.

1.1 Overview of Design Objectives

The design goals of Xdir include:

- Simplicity
  The interface is intended for use by novice users, and yet have the higher level of functionality required by more advanced users. To this end, an approach similar to that taken by the designers of the X.400 Mail User Agent[1] has been adopted. More advanced features are hidden from the novice user but can be explored as experience increases.

- Configurability
  The interface should allow the experienced user to adapt it to their particular requirements, in an interactive manner.

- Consistency
  A consistent approach for invoking commands and performing actions should be used.

- Portability
  The Xdir design should be capable of being implemented on many platforms. Where possible, the look and feel of Xdir, has been designed to be independent of style toolkits, so that a user will be able to recognize it on different platforms.

1 Hugh Smith & Graham Lunt, "The XUA Visual Interface", Communications Research Group, Nottingham University.

2 Visual Interface

This chapter describes the visual interface to Xdir. As Xdir is intended to be implemented on a number of hardware platforms, this design is high level and abstract in it's nature.

Xdir aims to provide a directory navigation tool which does not require the user to have a grasp of tree structures, whilst using the tree to guide the user's enquiry.

The visual interface will be described in two parts. Firstly, basic windowing interface primitives will be defined, and then the way these are used to create a "kit" of interface parts will be described. At the end of the chapter, an example configuration will be described, to demonstrate the use of the kit of parts.

2.1 Windowing Primitives

Many concepts and properties are common to all windowing environments. However, there are also many differences which may result from the use of a particular windowing system or development toolkit. This section specifies the primitives to be used and avoids toolkit/environment differences.

The primitives described are:

- window
- popup
- scroll bar
- button

Window

A window is a rectangular screen area which may consist of a title bar, scroll bars and a number of contained sub-windows.

Popup

A popup is a transient or semipermanent window which is created when required and destroyed when not required. Popups are used to minimize the use of screen real estate.

Scroll Bar

A scroll bar allows a large virtual screen area to be displayed on a smaller real screen area (viewport). The scroll bar associated with the viewport allows control over which part of the virtual area is visible in the viewport.

Button

A button is a screen area used to invoke an action. A button is pressed by moving the cursor into the button area and clicking with the mouse button.

2.2 Kit of Parts

The "kit of parts" concept allows the visual interface to be highly configurable in its layout and mode of operation.

The kit comprises a set of interface objects which can be grouped in many different ways.

The interface objects are described in the following sections. Some sections include example screen dumps. However if the nature of any of the interface parts is still unclear, it may be helpful to refer to the example layout described at the end of the chapter.

2.2.1 Button

A button is a generic method of invoking an action. Each Xdir button has a name and action associated with it. The name given to a button, is used as the label for the button. This can be specified either as a text string, or as a reference to a bitmap file which will be shown in the button. The action associated with a button will often require data which may be the result of a further action, such as clicking on an item or may be retrieved from another object in the grouping (such as a search value for the *search action*). This second method of retrieving data, implies that certain actions are logically bound to other interface objects. These actions can only be used in groups containing the requisite objects.

The actions which may be selected are listed below. The descriptions should become clearer when the interface objects have been described:

- ***Quit Action***
  quit application.

- ***Help toggle action***
  this controls the operation of the help text object.

- ***History action(group_name)***
  a history action passes a list of most recently visited entries to the specified group.

- ***Create window action(group_name)***
  this creates a window containing the objects specified in the list "group name" (see also *keep action*).

- ***Search action(group_name)***
  performs a search using the parameters found in other objects in the same group. The results are passed to the named group.

- ***List Action(group_name)***
  lists children under the ‘current position’ and passes the results to the specified group.

- ***Config action***
  allows configuration of many aspects of Xdir.

- ***Keep action***
  By default only one instance of a group may be visible at a time. If a second instance is requested, the first instance is cleared and reused to display the information. However, it is sometimes useful to be able to keep a number of instances of a particular group. If an instance of a group is to be kept, then the *keep action* will prevent it from being reused. The *close action* will always destroy the instance of the group.

- ***Close action***
  see *keep action*.

- ***Save action***
  save selected entry to all channels specified by current configuration. Typically these will be save to buffer or pass to XUA.

- ***Process save buf action***
  causes a menu to appear showing a number of destination/format descriptions, to which the entries in the save buffer will be saved.

- ***Delete save buf action***
  this requires selection of an entry from the save buffer. The selected entry will be removed from the buffer.

- ***Add save buf action***
  the entry currently being read is added to the save buffer.

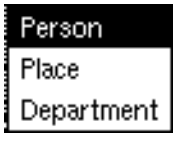
- ***Modify action***
  this gives a menu of modification actions that can be applied to the entry currently being read (modify, delete, move).

- ***Add attributes action***
  gives a menu of attributes which can be added to the entry currently being modified.

- ***Submit modified entry action***
  takes the current status of the entry being modified and tries to apply it to the directory.

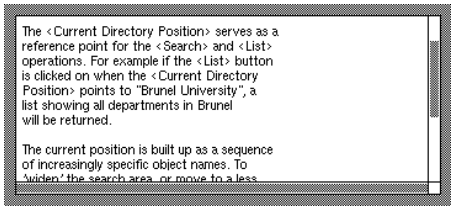
2.2.2 Menu

A menu object is very similar to a button object. However, clicking on a menu object, produces a list of actions to choose from. The actions then behave in a similar way to that described for the button object.



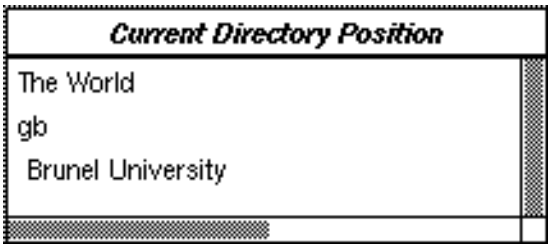
2.2.3 Help Text

The help text object provides a user driven help facility. When the object is active (controlled by the *help toggle action*), clicking on an area of Xdir, will display descriptive text relating to the object being selected. When the help text object is inactive, the last description displayed is kept, and actions have their normal effect.

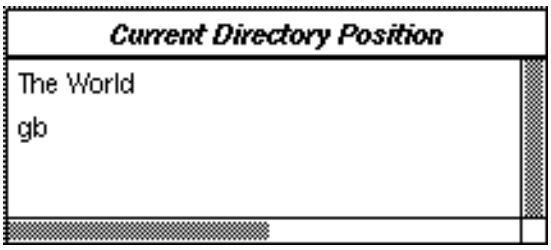


2.2.4 Current Position

A current position object displays the current directory position with the component RDN's shown separately. Clicking on an RDN causes it to become the current position. So, for example, if the current directory position is:



Clicking on 'gb' will make the current position:



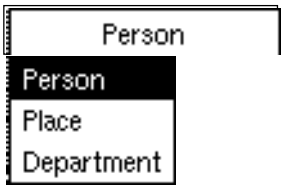
All current position objects are synchronized, so changing the current position in one window changes each current position object.

2.2.5 Search Value

The search value object has a name and a text editing area. The name can be set in a similar way to that used for button objects. The editing area allows a search value to be entered and/or edited, and key-bindings will be definable.

2.2.6 Search Type

The search type object also has name and value areas. The value area shows the current directory entry type being searched for. This will be set automatically to a default type. However, clicking on the search type object gives the user a choice of search types from a menu.



2.2.7 Status

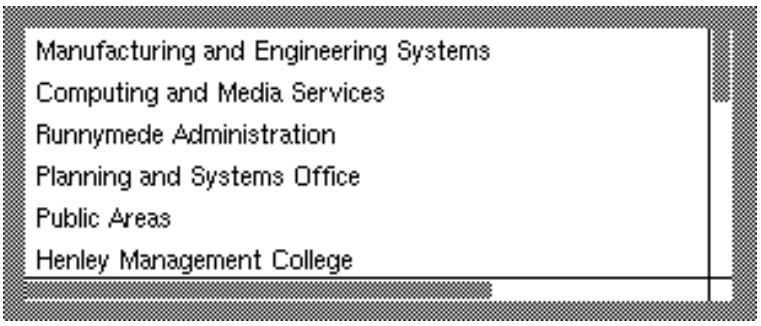
A status object is used to display errors and administrative limits which have occurred as a result of an action. A status object can take one of two forms, a simple text area or a popup. These can be configured for use by status messages of different severity. The object that is trying to display the status message will look for a status object in its group. If one is found, then it will be used, otherwise the message will not be displayed.

2.2.8 Title

A title object is similar to a status object but is used specifically to show titles. A title might show the DN of an object being read, or the parent of a list, or might simply be a text string.

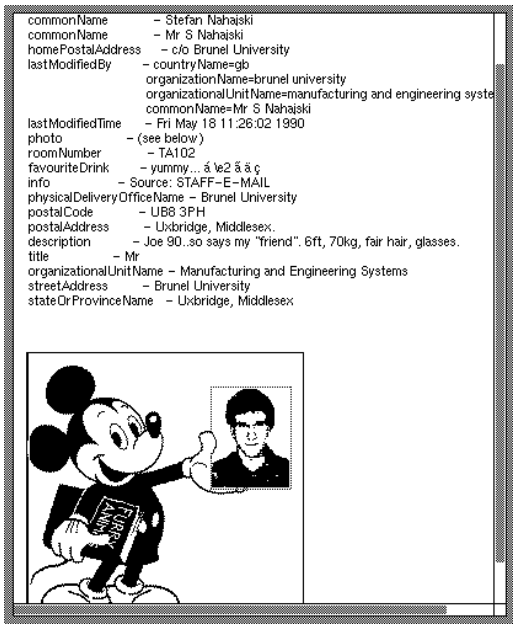
2.2.9 List

List objects are a general way of displaying lists, be they the result of list, search or history actions. The list to be displayed is shown in a scrolled text area and selections can be made from the list by using the mouse. The selected item is then passed to a context specified action.



2.2.10 Entry display

An entry display object displays the attributes of a directory entry. The list of attributes to display and the format used are configurable. Clicking on an attribute can be configured to perform an action using the selected attribute.



2.2.11 Modify Entry

The modify entry object is similar to the entry display object. The major difference being that the attribute values can be edited.

2.2.12 Add attributes

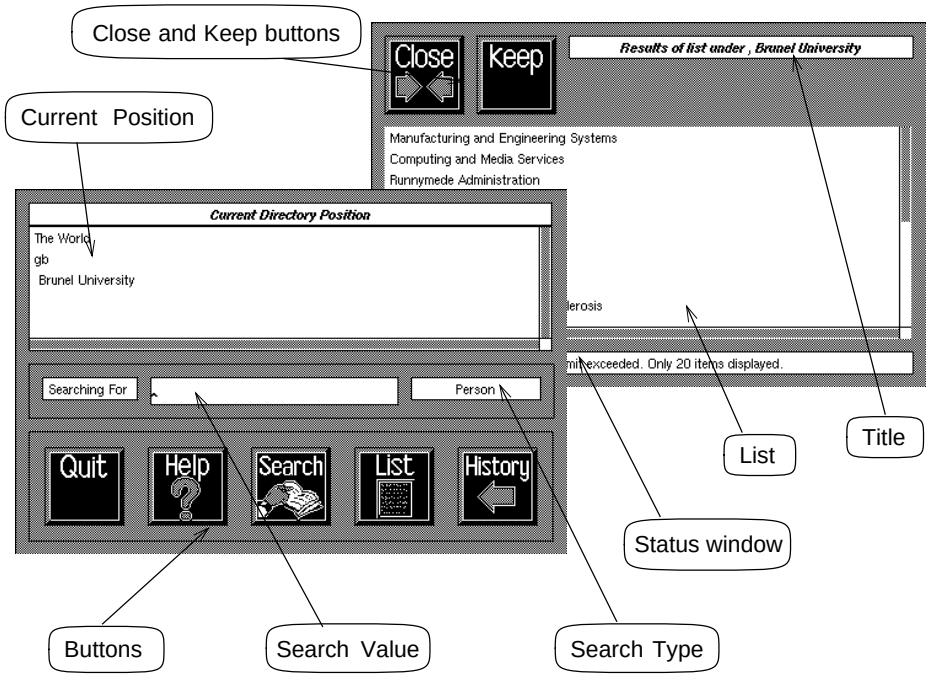
An add attributes object provides a pop down menu of attributes that can be added to an associated modify entry object.

2.3 Object Layout

In order to maintain a consistent layout throughout Xdir, objects are laid out in a predefined way. This can be overridden for a particular group of objects.

2.4 An Example Configuration

The diagram below depicts possible ‘main’ and ‘list’ windows using the described configuration scheme.



3 The Query Engine

The services provided by the directory are powerful when used correctly, but require the user to have some knowledge of the structure of the DIT and the kind of information contained therein. The solution to this is to make as many ‘reasonable’ assumptions as possible about what the user is looking for and the composition and structure of the directory database.

Some of the ideas presented here will require an asynchronous interface to the directory. At present only a synchronous interface is available.

3.1 What does the query engine do?

The query engine attempts to formulate directory queries that are likely to succeed, but not at the expense of overloading a DSA. Thus a trade-off is needed between the complexity and number of queries made against the probability of success.

The services provided by the query engine are closely aligned to those defined by the X.500 standard. These being:

- Search
  Search for entries within some defined part of the DIT
- Read
  Read the contents of a directory entry.
- List
  List the children of a directory entry.
- Modify Entry
  Modify the contents of a directory entry.
- Add Entry
  Add an entry to the database.
- Delete Entry
  Delete an entry from the database.

- Rename Entry
  Modify the directory name of an entry.

The most important of these is the search operation, as this constitutes the basic function of Xdir, to search through data.

3.2 Search

The query engine offers several named search strategies. These are defined in a configuration file. The circumstances in which each strategy is useful can be defined so that the most appropriate one is always offered by default.

Possible strategies range from the simple ‘exact match on a single attribute in one level of the DIT’, to the environment sensitive searches suggested in work from University College London[1].

3.2.1 Object Types

In Xdir, searches take the form of a search within some part of the DIT for an object of some particular type, an object type. It could be a search for a person, a place, an organization etc. This contrasts with the X.500 search type, which defines only the kind of attribute to match against, not the type of entry that the attribute is contained in.

A complete definition of an Xdir object type is somewhat complex. It is not only the kind of object that is being searched for that matters, but also the way in which this object relates to other parts of the DIT. For example, once an object has been located and moved to, it is also necessary to know how that object relates to it’s children (or indeed if it has any children) in order to simplify any searches below the entry. The information that defines an Xdir object type is:

- Type definition
  This refers to the information that distinguishes one Object Type from another. The data contained in the ‘objectClass’ attribute will be used for this purpose, though this is limited and not always consistent.

- Matching information
  The attributes that the object is likely to contain and that a user is likely to try to match against. When

1 S.E. Kille, "Using the OSI Directory to achieve User Friendly Naming", Computer Science, University College London, April 1990.

searching for a person one is likely to try to match using his/her personal name, when searching for a 'person with a role' one would try to search using the name of the person's role.

- Structural Information
  How does a specific type of entry relate to the rest of the DIT? Specifically is this a leaf entry, or else what is the organization of data held below this entry?

- Naming Information
  How is this type of entry to be named in an addressing or searching context?

As an example the generic type 'Place' may then be informally defined as:

Type denoted by:
objectClass contains 'locality'
OR objectClass contains 'room'
OR objectClass contains 'country'

Match on:
countryName
OR stateOrProvinceName
OR locality
OR friendlyCountryName
OR commonName

Label: 'Place'

Structural Information:
for 'country' entries or 'locality' entries
children may be: organization, use search single level
locality, use search single level
for 'room' entries
no children

The generic type definitions used by Xdir are provided in a set of user configurable files, in order to allow for the variation of structure and content in different parts of the DIT.

3.2.2 Search Strategy

The Xdir search strategy operates at two levels:

- Matching values against a given set of entries
- Ascertaining the X.500 distinguished name of an entry.

Matching against Entries

The available matching algorithms have different strengths and weaknesses. Any one of them cannot cater for all likely situations on it's own. For example a simple search under Brunel University, with surname approximately matches the value 'nahajski', returns a list containing around fifty entries. This is unreasonable behaviour. Substring or exact matching are also not particularly useful, as the name 'nahajski' is one that many are likely to misspell.

In an attempt to allow for this Xdir will make multiple queries, using similar filters but employing matching algorithms with increasing levels of 'looseness', and merge the results. Thus a first search might use 'exact' or 'substring' match. If very few or no matches are made then a further search is made, using an approximate matching algorithm. The results of the two searches are then merged with results from the initial search placed at the top of the displayed list.

Ascertaining X.500 Distinguished Names

As well as performing searches on a single value, Xdir will allow the user to enter a user-friendly name that can be used to ascertain the distinguished name of an entry. The framework of such a search is illustrated in the following example. The user supplied name (not necessarily presented in this way) is,

> polonius plum, parkway polytechnic, great britain

leading to a search operation that goes through the following steps:

> i) search at top level for 'great britain', making the assumption that 'great britain' is a country or organization.
>
> ii) search below all entries returned by step (i), for 'parkway polytechnic' making the assumption that 'parkway polytechnic' could be any of the following: locality, organization.

iii) subtree search below all entries returned by step (ii) for 'polonius plum' which is assumed to be a personal name.

If any of the individual steps return too many entries, then the user is asked for more or better information.

Obviously this example makes the unreasonable assumption that it is always possible to perform subtree search within an organization, but it has been made simple in order to illustrate the approach taken and to give an idea of how it will work for the user.

This type of heuristic search will only perform well when an asynchronous programming interface becomes available. With the current synchronous interface such searches can be very time consuming. It is expected that the implementation of Xdir produced within the timescale of this project will be based on a synchronous interface.

3.3 List

As with current DUA prototypes, a list invocation will actually be a one level search using a filter that removes unwanted information, e.g. entries for DSAs.

3.4 Read

The query engine will provide a straight interface onto the X.500 read function.

3.5 Modifying The DIB

The modify entry, add entry, rename and delete entry services will be straight interfaces to the services provided by X.500.

4 Xdir Configuration

Configuration within Xdir is extensive and simple. It allows tailoring of the following aspects of the application.

- Layout of the visual interface
- Object Type definitions and Search Strategies
- X.500 service parameters

Xdir will contain tools that allow interactive editing of the configuration files. This simplifies the process by bypassing the need to edit user-unfriendly tailor files.

All configuration options will be read in from a single file. For administrative purposes this can be made up from a number of included files. The configuration file will consist of a set of unsequenced attributes.

4.1 Layout of the Visual Interface

As described in the previous chapter the user interface is built out of "kit of parts", where each part is a primitive object that may have an associated action. For example a quit button is a primitive that has the associated action of quitting from the interface.

Xdir contains a set of in-built configuration tools that allow users to modify the organization of these primitives in a totally flexible way. This, for example would allow all input and output windows to be contained in a single main window like Xd, or in a separate windows like Pod.

As a more likely example the main window may be configured to contain a search value dialogue box, a start search button and read window. This could be a suitable configuration for a screen in a reception area, allowing simple search for a person within an organization.

4.2 Display and Save Formats

The selection of attributes and the format used to represent them can be configured. A single set of configuration options is applicable both to display windows and save channels.

4.3 Search Strategies

Search strategies and their default usage can be user configured. This will allow the behaviour of Xdir to be modified according to future changes in the structure of the DIT or local structure.

4.4 X.500 Service Parameters

X.500 Service parameters associated with individual requests can be modified via an options menu. Default service parameters are retrieved from a tailor file.

5 Internal Design

5.1 Internal Structure

The internals of Xdir have been designed using an object-oriented approach. At the highest level Xdir can be seen as being comprised of two objects: a visual interface object and a directory. These may then in turn be comprised of component objects (referred to as sub-classed objects). For example a list window is a sub-classed object of the visual interface.

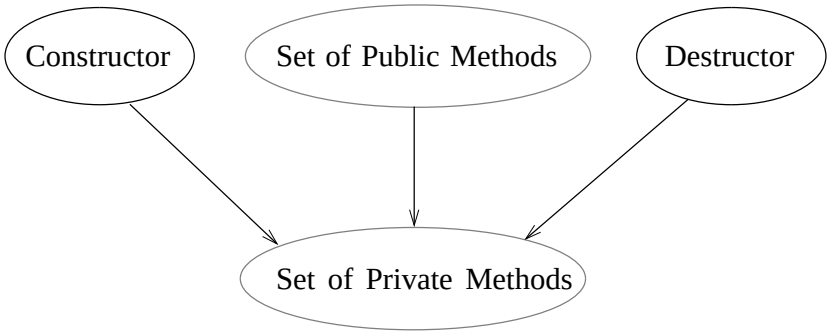
An ‘object’ is built out of four parts:

- Constructor function
  This initializes the object and creates any data structures or sub-objects.

- Destructor function
  Destroys the object.

- Private set
  Methods and data associated with an object but ‘private’ to that object.

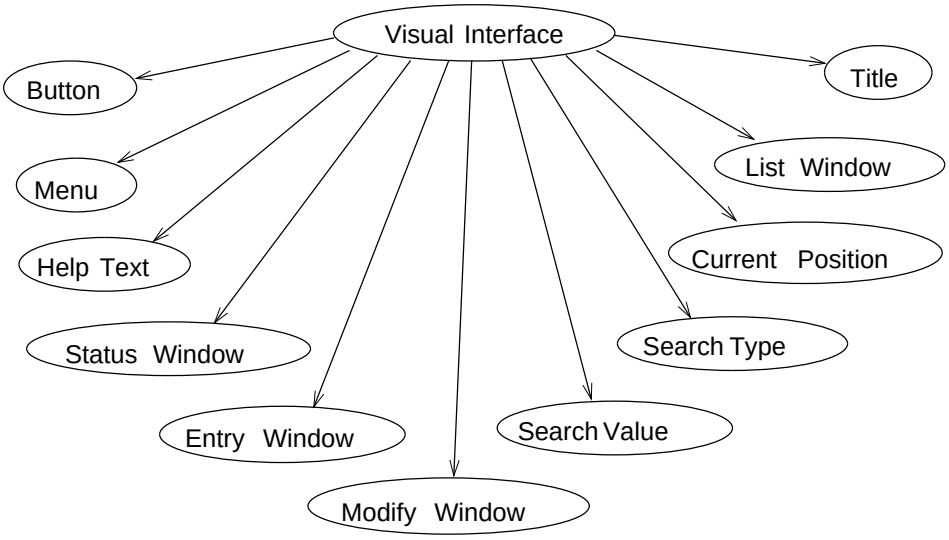
- Public set
  Methods and data that are ‘publicly’ available. This amounts to the interface to an object.



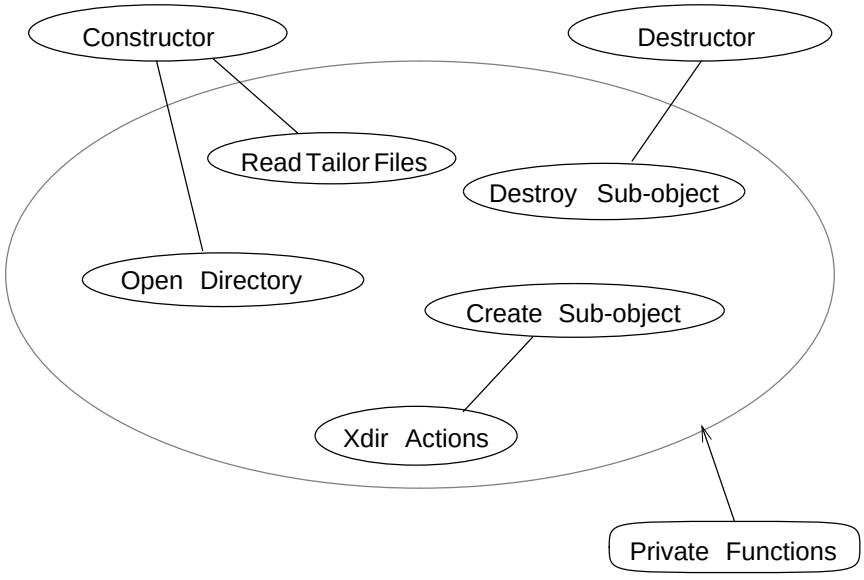
The diagram below shows the relationship between these parts.
The internal structure of Xdir will be specified in terms of the relationship between each contained object, followed by a description of the internals of each object.

5.1.1 The Visual Interface

The diagram below shows the hierarchy of objects comprising the visual interface.

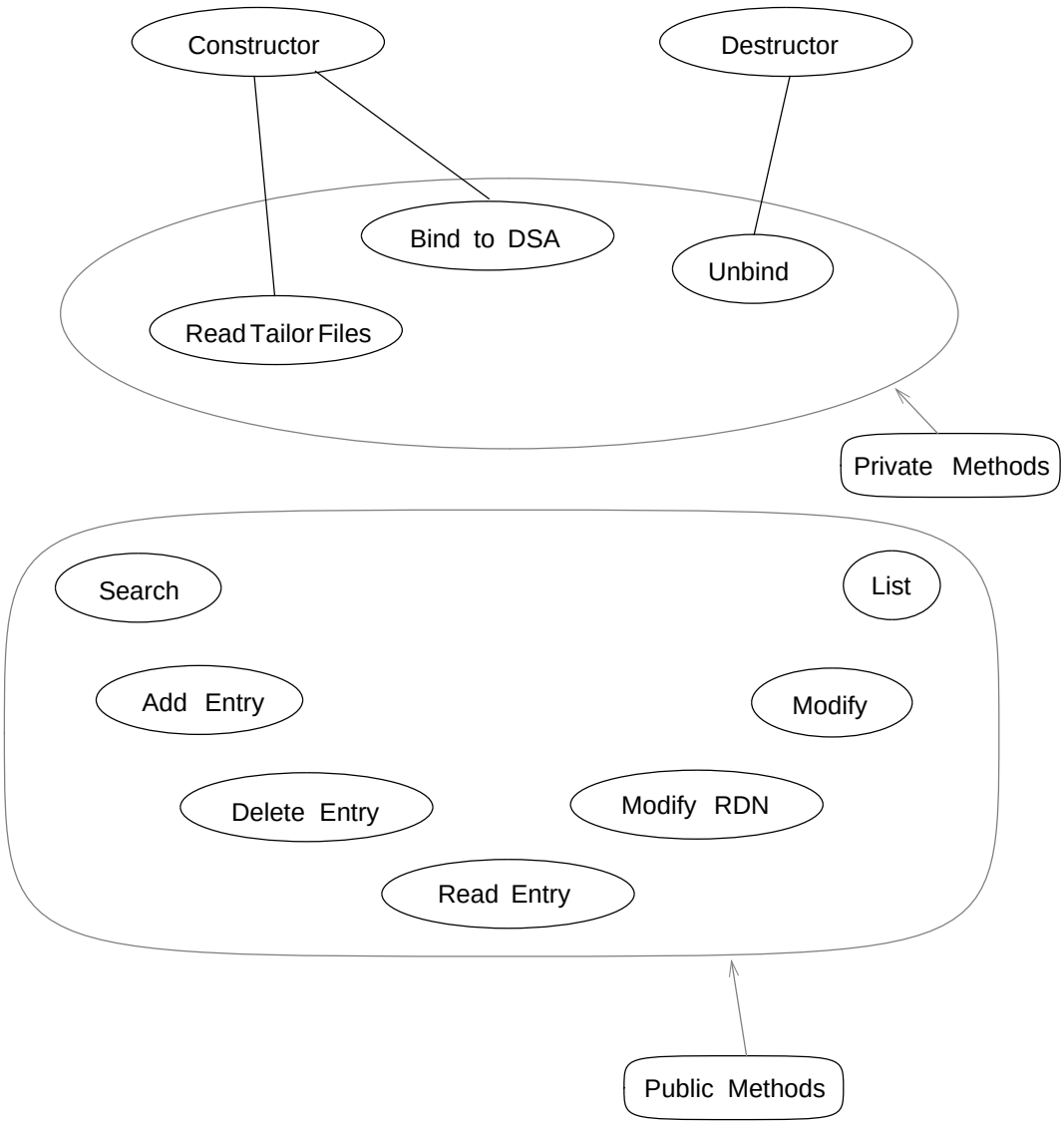


The internal structure of the visual interface object is depicted below. The set of Xdir actions, implemented as X toolkit callbacks, are selectively inherited by the sub-classed window objects.



5.1.2 The Query Engine

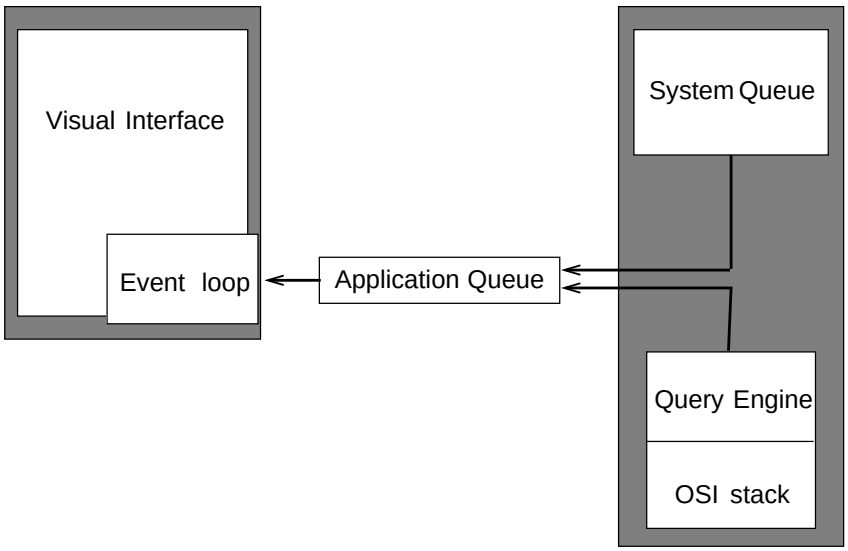
The structure of the query engine object is depicted below. The public functions defined in the visual interface are specified in Appendix A.



5.2 Merging Two Asynchronous Event Loops

Xdir faces the problem of managing two separate asynchronous event loops. The windows event loop and the notification events from the OSI stack. In the case of the MS-windows implementation this is trivial as stack messages are passed using the MS-windows message loop. For the X-windows implementation, the two event loops will be merged at a point between the query engine and the visual interface. It is intended that the visual interface will supply X event generating procedures to the query engine, which the query engine will use to pass results back to the visual interface.

The diagram below shows the flow of information between the various parts of the interface.



Appendix 1: Abstract Programming Interface

Data types and associated functions:

status
- status check ok
- status check fatal
- set status
- print status

entry name
- read entry name
- set entry name
- print entry name

attribute
- read attribute value
- read attribute name
- set attribute value
- set attribute name
- convert attribute value for display
- convert attribute name for display

search value
- set search value
- read search value

search type
- set search type
- read search type
- search options
- read search options
- set search options

Composite data types:

name list
- entry name
- entry name
- .
- .

attribute list
    attribute
    attribute
    .
    .

directory entry
    status
    entry name
    attribute list

directory list
    status
    entry name (parent)
    name list

directory position
    entry name

search specification
    search value
    search type
    search options

read specification
    entry name
    read result
    directory entry

Functions provided by query engine for use by visual interface:

search request(search specification) = directory list
read request(read specification) = directory entry
list request (entry name) = directory list
add entry request(directory entry) = status
delete entry request (entry name) = status
modify request (entry name, attribute list, attribute list) = directory entry
modify rdn request(entry name, attribute) = status