# Inside the Lizard
# A Look at the Mozilla Technology and Architecture

Mike Shaver
shaver@mozilla.org

Michael Ang
mang@subcarrier.org

May 29, 2000

The Mozilla "platform" is composed of a set of technologies and components that can be used to build cross-platform applications. This architecture was designed for the Mozilla browser and Netscape Communicator 6.0 but can also be used for other types of applications.

This document provides an overview of the key technologies used in the Mozilla architecture, including a discussion of the component object model, scripting components, creating cross-platform user interfaces, and accessing data sources. Porting the architecture to new platforms will aslo be discussed.

## 1 Introduction

After the initial release of the Mozilla code on March 31, 1998, there were many enthusiastic attempts to make that codebase (known as the "Classic" code) provide features such as incremental layout with reflow, fully compliant CSS and a Level-1 Document Object Model. The nature of the Classic code, though, made such tasks difficult and error-prone. Instead of continuing work on the original code base, the current architecture was created that incorporated the painfully learned lessons about cross-platform front-end development.

The new code base was designed to provide a framework of components upon which the web browser Mozilla could be built. The framework is flexible enough that it can now be used as the base for applications other than a browser. Creating a new cross-platform application can be as easy as putting together existing components using JavaScript and XUL. New components to extend Mozilla's functionality can be written in C++ or JavaScript.

While Mozilla lacks many features required of a full-blown application platform (such a remote procedure calls), there are still many interesting applications that can be created. Some novel uses of the Mozilla framework include XMLTerm, an X-Term replacement, and ChatZilla, a full-featured chat client.

## 2 Modular Code: XPCOM

The XPCOM system is a cross-platform variant of the DCE/Microsoft COM system for building binary components. We use it to provide clear "pluggablity" and modularity boundaries between sections of code, and also to manage object lifecycles through reference counting.

XPCOM strongly separates interface from implementation, down to the binary level. These two abstractions map to two C++ classes: an interface class and an implementation class. The client interacts directly only with the interface class, which contains no implementation details and has a well-defined binary format.

Since clients of XPCOM know only about the interface to an object, the implementation can be safely upgraded or replaced, compiler differences are hidden, and objects can even be implemented in another language (e.g. JavaScript).

### 2.1 XPCOM techniques

XPCOM objects can expose multiple interfaces (somewhat similar to multiple inheritance). `QueryInterface` is XPCOM's equivalent of C++'s dynamic cast. To access an object through a particular interface, you `QueryInterface` for that interface.

Since the same object may be accessed through several different interface pointers, reference counting is used to keep track of the lifetime of objects. When a client receives an interface pointer, the reference count on the underlying object is increased by one. Once the client is done with the interface pointer it calls Release on the pointer, and the reference count is decremented. Once the reference count reaches zero, the object is destroyed. `nsCOMPtr` is a smart pointer implementation that takes care of these details automagically.

XPIDL is the cross-platform interface definition language used for XPCOM. XPIDL files are processed to generate C++ headers for XPCOM objects, typelibs to allow calling objects dynamically from JavaScript (and potentially other languages), and generate online documentation.

### 2.2 XPCOM and JavaScript interoperation

Using XPConnect, it is possible for JavaScript to interoperate with XPCOM without needing any glue code. XPConnect uses the runtime type information previously built from the XPIDL.

Code using XPCOM interfaces should not generally know or care what language is used to implement underlying objects. XPConnect allows JavaScript to interoperate seamlessly with XPCOM. JavaScript code can manipulate XPCOM objects, and JavaScript objects can present XPCOM interfaces to be called by other XPCOM code.

XPConnect uses the runtime information generated from XPIDL interface definition files when JavaScript calls into XPCOM. No glue code needs to be written, and this results in significant memory and disk footprint savings.

Connection technologies for other languages are possible. (Sun engineers are currently investigating interoperation of XPCOM and Java, and ActiveState recently announced a project to provide connection technologies for Perl and Python.)

## 3 User Interface: XPToolkit

Historically, developing the user interface required writing C++ code for each platform of interest. Mozilla provides a set of tools collectively called the Cross-Platform Toolkit (XPToolkit) that can be used to define user interface appearance and application behaviour using W3C standards.

XPToolkit minimizes the amount of platform specific code needed for building applications like browsers and mail clients. XPToolkit makes available a standard set of widgets and graphics services to render UI elements, allowing pixel level control of UI if necessary.

### 3.1 XUL

XUL, which stands for XML-based UI Language and is pronounced “zuul”, provides a way to specify UI appearance and application logic using an XML dialect, CSS, and JavaScript (or C++).

XUL is simply XML with specific meaning defined for a few element types, and into which JavaScript can be scattered. The elements are known as XUL “widgets” and include UI controls such as menus, trees, and drop-down boxes. XUL makes it easy for developers with a knowledge of an SGML (such as HTML) and JavaScript to design UI: this opens up UI development to a whole new set of programmers. The JavaScript application logic can access XPCOM (and thus most libraries) using XPConnect, making it possible to build complete applications.

The combination of appearance and behaviour is called “chrome”. Chrome is loaded from `.xul` files and associated JavaScript and CSS file. The script code in these files have full power to change and extend the behaviour of the browser, so this is allowed only from trusted scripts. By contrast, “skins” are combinations of graphics and CSS that only change the appearance of the browser, not behaviour.

Most of the UI for the browser is written in XUL, and is thus open for hacking. Adding a button on the toolbar that opens a tree control and populates it from a remote data source is simply a matter of writing some XML and JavaScript.

XUL makes it possible to develop the UI using familiar web technologies. One of the goals of XUL is to enable random hackers to create new features easily and share them with others. This will allow the UI to enjoy the same massively parallel development that has benefited other Open Source projects.

The recently introduced eXtensible Bindings Language (XBL) provides even more flexibility. For example, XBL allows for the creation of new widgets which can be accessed from XUL, automatically picking up methods and properties from XPCOM components, and extending event handlers including capturing keypresses.

## 3.2 XBL

XUL provides many existing widgets such as text boxes and scroll bars. The eXtensible Bindings Language (XBL) makes it possible modify existing XUL elements and create new ones.

XBL files contain a set of bindings, each of which describes the behaviour of a XUL element. Bindings may specify the following: child content, such as other XUL elements methods and properties, where the implementation and data can be either inline JavaScript or point to an XPCOM interface events that the element will handle, such as key and mouse events custom style properties

For example, a new widget for a labelled image could be created by extending the existing box widget by adding an image and a label:

```
<binding name="captionbinding" extends="xul:box">

<content>
  <xul:box
      xmlns="there.is.only.xul">
    <xul:image inherits="image:src"/>
    <xul:text inherits="caption:value"/>
  <xul:box/>
</content>

</binding>
```

We then attach the binding using the following code, which is found in the corresponding CSS file.

```
captionimg
  behavior: url("mybindings.xbl#captionbinding");
}
```

Code snippet starts with the name of the new element being defined (`captionimg`). The URL specifies the filename followed by the name of the binding (`captionbinding`) that we defined earlier.

The resulting compound element would be used from XUL like this:

```
<captionimg
  caption="My image caption" image="myimage.png"/>
```

## 3.3 XPFE

The Cross-Platform Front End (XPFE) is the name given the Mozilla browser front end (chrome). The definition and use of this term is a bit loose, but the trend now is to use “XPFE” when referring to the specific Mozilla browser user interface built on top of the generic XPToolkit.

## 4 RDF

The Resource Description Framework[1] is a graph-based model for describing Internet resources (like web pages and email messages), and how these resources relate to each other. Mozilla’s use of RDF stretches the common usage of “Internet resources”, though, and adds resources like bookmarks, local file system information and address book content to the mix.

### 4.1 RDF in Brief

RDF allows arbitrary hierarchical content to be manipulated through a common API, and serialized in a standard syntax. This serialization syntax, known as RDF/XML, allows RDF-using applications to communicate their graphs to each other, and store those graphs persistently.

[1] http://www.w3.org/RDF

Within Mozilla, there are APIs provided to manipulate RDF nodes (resources) and assertions (links between nodes, like graph arcs or objects holding "pointers" to other resources). This allows Mozilla components to act as RDF "datasources" and perform operations on those datasources, without having to go through the RDF/XML serialization syntax. The XPCOM interfaces allow you to create resources; add, remove and query assertions between resources; and access registered datasources.

## 4.2 Uses of RDF in Mozilla

RDF is used in Mozilla to represent many kinds of interesting data, from bookmarks and address books, to mail folders and remote search engines. Examples of RDF usage in Mozilla include:

- Bookmarks
- Mail folders
- Newsgroups
- Remote site maps
- "Sidebar" or "Flash panel" content
- Browser history and browsing profile
- Address book
- "Related Links"
- Local filesystems
- Remote FTP sites
- Web search engines

## 4.3 RDF and XUL

To display RDF data in Mozilla, the XUL tree widget is usually employed:

```
<tree datasources="rdf:bookmarks"
    ref="NC:BookMarksRoot">

<template>
  <treechildren>

  <treeitem uri="rdf:*">
    <treerow>
      <treecell
        value="rdf:#Name"/>
      <treecell
        value="rdf:#URL"/>
    </treerow>
  </treeitem>

  </treechildren>
</template>

</tree>
```

The bookmarks source is specified by `rdf:bookmarks` and the resource used to "root" the tree is `NC:BookmarksRoot`.

The items in the tree are generated using a XUL template. The `uri` attribute is used to designate that a given element will be repeated in the template. In this example one `treeitem` with a contained `treerow` and two `treecell`s will be generated for each resource found.

The assertions on the bookmarks graph named by `Name` and `URL` are used to populate the cells of the tree grid, and adding or removing new assertions to the RDF graph used will cause the tree widget to automatically update with new data. (Typesetting concerns required a small liberty to be taken with the example: URIs are used for property naming, and the Mozilla datasources would have used `rdf:http://home.netscape.com/NC-rdf#Name` and `rdf:http://home.netscape.com/NC-rdf#URL` as the canonical `rdf:#Name` and `rdf:#URL` assertion names, respectively.)

With XUL templates it is possible to create a set of rules to create different types of elements based on the RDF data. For example, rules are used in the bookmark management XUL template to display actual bookmark entries differently from bookmark separators, even though both are contained in the same RDF resource.

## 4.4 Creating New Datasources

To reflect another kind of resource into the RDF universe, a new RDF datasource must be implemented. The `nsIRDFDataSource` interface must be implemented directly if the datasource wishes to react to RDF operations in a specialized manner, such as adding a file to the filesystem when a new assertion is made, and is described in `mozilla/rdf/base/idl/nsIRDFDataSource.idl`. If the datasource does not require special handling of the `nsIRDFDataSource` API operations, XPCOM aggregation can be used to let the `nsIRDFInMemoryDataSource` handle most of the RDF operations. This approach is often used when the datasource is parsing data in a non-RDF format, and perhaps writes changes out to the original storage when the RDF engine has finished whatever modifications are performed.

When a datasource generates a graph to represent its chosen data, the selection of "vocabulary" is key: merging, sorting and other useful RDF features require that the participating datasources use the same assertion names. In addition to the `NC` vocabulary used for many of the Mozilla datasources, the RDF Schema Specification[2] and Dublin Core[3] provide other standard vocabularies for describing electronic resources.

Once your datasource is implemented, you must use the component manager to register your datasource with the RDF infrastructure. Detailed discussion of this process is beyond the scope of this document, but a good example can be seen in `mozilla/rdf/build/nsRDFFactory.cpp`.

A more detailed treatment of issues involved in RDF datasource implementation is available in the "datasource howto" [4].

## 5 Porting Issues

In order to port Mozilla to a new platform, three major components require attention. (It is assumed that there is an appropriately capable C/C++ toolchain available — gcc 2.9.x and egcs are suitable. If the toolchain is not sufficiently capable, additional effort may be required: as an example, the OpenVMS port had issues with a linker that did not distinguish symbols on the basis of case.)

The foundation of Mozilla's portability is the Netscape Portable Runtime (NSPR). Applications built on NSPR can use system facilities such as threads, thread synchronization, I/O, interval timing, atomic operations, and several other low-level services in a platform-independent manner. Much of the work in porting Mozilla to a new platform is in porting NSPR.

The `gfx` library provides primitive graphics and rendering capabilities, including line drawing, image rendering and text display. The Mozilla source contains implementations of `gfx` widgets (such as menus, buttons, scrollbars, text entries and checkboxes) which are rendered from primitives without the assistance of the platform's native widget library. This permits Mozilla to fully support HTML4 rendering requirements, including partial opacity for form elements and use of animated images in backgrounds. However, in order to integrate Mozilla into a desktop environment such as KDE, it is necessary to port at least message passing, timers, and menus from the underlying toolkit (Qt in this case).

Before the existance of `gfx`, it was necessary to port the `widget` library to use the system's native widget set (such as GTK+, Qt, Windows, or Macintosh). As of May 2000, the only remaining dependency on the native widget library is for scrollbars in HTML list and combo boxes. This dependency is in the process of being removed.

`xptcall` is used by XPConnect and is required on all platforms. Porting `xptcall` requires knowledge of the platform's calling conventions at the assembly language level (this isn't as bad as it may sound). As of May 2000, there are a handful of platforms (including some Linux platforms) that do not have a working `xptcall`. Significant detail on the topic of porting the `xptcall` library, as well as a list of completed and developing ports, can be found in `mozilla/xpcom/libxpt/xptcall/porting.html`.

## 6 Breaking News

An incomplete list of current projects in the Mozilla codebase at the time of this writing:

- The newly formed Architecture Group will work on footprint and performance issues, API design and review, and contribute to making Mozilla a stable platform for developers.

[2]http://www.w3.org/TR/PR-rdf-schema/
[3]http://purl.oclc.org/dc/
[4]http://www.mozilla.org/rdf/doc/datasource-howto.html

- A "Mozilla 1.0" architecture freeze is brewing for later this year. The freeze will involve a review of the public APIs and blessing a core set of XPCOM interfaces, moving to new style progids, and using the new string APIs, and a freeze of the DOM. After the freeze developers should have a stable base upon which to create applications with the Mozilla framework.

## 7 Acknowledgements

The authors would like to specifically thank the following for their documentation and instruction:

- Chris Waterson (*waterson@netscape.com*)
- David Hyatt (*hyatt@netscape.com*)
- Scott Collins (*scc@netscape.com*)
- John Bandhauer (*jband@netscape.com*)
- Warren Harris (*warren@netscape.com*)
- Christopher Blizzard (*blizzard@mozilla.org*)
- Brendan Eich (*brendan@mozilla.org*)

In addition, all the Mozilla developers, testers, documenters, kibbitzers and cheerleaders are deserving of heartfelt thanks for making Mozilla such an exciting project on which to work.

This document was originally published in July 1999 and was updated in May 2000 by Michael Ang. Any existing inaccuracies are mine alone.

## 8 Additional Information

- Useful URLs.
  - Roadmap:
    `http://www.mozilla.org/roadmap.html`
  - XPCOM:
    `http://www.mozilla.org/projects/xpcom/`
  - XPConnect:
    `http://www.mozilla.org/scriptable/`
  - XPToolkit (XUL/XBL):
    `http://www.mozilla.org/xpfe/`
  - RDF:
    `http://www.mozilla.org/rdf/doc/`
  - Weekly status:
    `http://www.mozilla.org/status/`
  - "SeaMonkey" (Mozilla 1.0):
    `http://www.mozilla.org/projects/seamonkey/`
  - ChatZilla
    `http://www.mozilla.org/projects/rt-messaging/chatzilla/`
  - XMLTerm
    `http://xmlterm.org`
- Source browsing. The complete Mozilla source tree is browsable via the LXR web interface, and the hyperlinked version is updated approximately every 15 minutes. To view a source file through LXR, remove the leading `mozilla/` prefix and replace with `http://lxr.mozilla.org/mozilla/source`. For example, `mozilla/js/src/jsapi.c` becomes `http://lxr.mozilla.org/mozilla/source/js/src/jsapi.c` (modulo typesetting damage).
- Mailing lists and newsgroups. Mozilla hosts several dozen mailing list/newsgroup pairs related to a variety of topics. `http://www.mozilla.org/community.html` lists many of them, and the general forms `mozilla-`*`topic`*`@mozilla.org` and `netscape.public.mozilla.`*`topic`* hold for all. Messages sent to the mailing list or posted to the newsgroup appear in both places. New fora are announced on the Mozilla web site periodically, as they are created.