The OpenOffice.org Source Project

Technical Overview

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
1 (800) 786.7638
1.512.434.1511

Copyrights and Trademarks

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Contents

Preface

The purpose of this paper is to provide a technical overview of the OpenOffice.org source project.

It is expected that as the OpenOffice.org source project matures, some of these details will evolve. This document will be updated periodically to reflect the nature of these changes to the OpenOffice.org source project. If you would like to contribute to any updates to this document, please join the OpenOffice.org general mailing list (www.openoffice.org).

The names used in this document for OpenOffice components, such as "OpenOffice.org wordprocessor application" are placeholders. The OpenOffice.org community will ultimately decide what these final names will be. This document will be updated when the component names have been finalized.

History

Version	Publication Date	Change Notes
Version 1.0	9/12/2000	First version of this paper.

Summary

Through the OpenOffice.org source project, Sun Microsystems is open-sourcing the technology that powers its StarOffice™ office productivity application suite. Sun recognizes that the open source community expects openness, interoperability, and adherence to standards and, now that the underlying technology of the StarOffice suite will be available to the community in the form of the OpenOffice.org sources and binaries, Sun presents in this document the OpenOffice.org suite's technological foundations and where they stand with respect to these expectations.

The OpenOffice.org suite's high level of interoperability derives from the standards it supports as well as its premier import/export interfaces with the various office productivity applications produced by Microsoft. The OpenOffice.org suite employs a component-based development system that exemplifies all the important characteristics of Component Ware - consistent interface allocation, support for important component standards, transparent localization components, batch job capability, and platform independence.

OpenOffice.org's component technology is open, object oriented, interface based, and independent of both platform and development system. The OpenOffice.org API is version independent, scalable, durable, and re-applicable. Because the component technology is used in its implementation, the OpenOffice.org API is programming language independent.

XML replaces binary as OpenOffice.org's file format and becomes the suite's new native file format. Sun and OpenOffice.org are positioning XML, with its extremely high standards profile, as the next standard for exchange of office documents.

OpenOffice.org Components

OpenOffice.org Suite

OpenOffice.org is a unified suite of productivity applications for all common office applications, including such functions as word processing, spreadsheets, drawings, presentations, data charting and formula editing. All components of the suite employ the same user interface concepts and underlying technology. They interoperate closely with one another, supporting features like inter-application copy-and paste and drag-and-drop for creating compound documents. It is straightforward to embed a spreadsheet in a text document, for example. They also interoperate well with other common desktop productivity application suites, including the various office productivity applications produced by Microsoft, for ease of document exchange. A scripting environment called OpenOffice.org Basic is supported in all OpenOffice.org components to automate work or build solutions.

OpenOffice.org wordprocessor application[1]

The OpenOffice.org wordprocessor application is a powerful tool for creating professional documents, reports, newsletters, and brochures. It is easy to integrate images and charts in documents, create business letters and extensive text documents with professional layouts, as well as create and publish Web content.

OpenOffice.org spreadsheet application

The OpenOffice.org spreadsheet application features decision making analysis tools for performing advanced spreadsheet functions and data analysis. Charting tools generate presentation applicationive, high-quality 2D and 3D charts.

[1] Component names are placeholders and will be replaced with actual names when said names are finalized; see the Preface for more information.

OpenOffice.org presentation application

The OpenOffice.org presentation application is a tool for creating multimedia presentations. Included are 2D and 3D clipart, special effects animation, and high-impact drawing tools.

OpenOffice.org drawing application

The OpenOffice.org drawing application is a vector-oriented drawing module that enables the creation of dynamic 3D illustrations and special effects.

OpenOffice.org data charting application

The OpenOffice.org data charting application presents complex data in visually presentation applicationive ways, from colorful 3D charts to simple pie, bar, and line diagrams.

System Integration

The overarching goal for the OpenOffice.org suite is to provide a comprehensive set of solutions for all office related functionality in an open world. The focus for all OpenOffice.org components is office functionality.

This is the reason why all office components should provide a perfect integration into already existing environments. Instead of competing with already accepted applications, the OpenOffice.org source project will provide the flexibility to use the office functionality in these applications as integrated parts. This will allow the use, for example, of office productivity files from a variety of vendors, including Microsoft, as mail attachments on every platform. In the future, it will also open up a way to build highly sophisticated applications and solutions with integrated office functionality.

Future releases of the OpenOffice.org applications should provide the flexibility to use different messaging components for Mail and News or let the user decide which tools he wants to use to explore the filesystem, while at the same time the office components can be an integrated part inside this application to deal with word-processing, spreadsheet, etc. related content.

Some technology used in the StarOffice product was licensed from other companies. Accordingly we are not able to provide the following technologies as Open Source under the OpenOffice.org source project:
✍Bristol XPrinter - printing on UNIX ® platforms

- L&H International CorrectSpell, Intl. Electronic Thesaurus - spell checking, international dictionaries & thesaurus
- Inso Word for Word - document filters for document formats other than MS Office
- Adabas D - database engine

Future releases of the OpenOffice.org components may provide open source replacement for these parts, which will provide similar functionality.

Interoperability

For exchanging information it is important that two individuals have a common language. For exchange electronic information today it is not hard to transport this information to different system, instead the problem is in most cases that users use different applications to do their work and to create content. So very often in the domain of office productivity you can only read the document with the same application you created the content, because the file format are proprietary. In some case it is also required to use the same release of the application, the same operation system release or run this on the same platform.

To give the use the freedom to use application and the system which is appropriate for his work, it is important that the system is able to handle a widespread set of file formats. Especially for word-processing, spreadsheet, presentation it is often necessary to be able to read the different binary formats used by the various office productivity applications produced by Microsoft.

On the other hand interoperability is not only necessary for content. It is also necessary that the components which are able to process the different file formats can be used in an heterogeneous environment to build up solutions. Otherwise you would be locked into another application space. Today's software technology is moving forward from having libraries to component technology, which will not only allow using different programming languages to build solutions, instead it will in most cases support using a scripting language to provide the logical glue between high-level component build rapidly easy adaptable software solutions.

File Formats

One of our primary objectives is to provide interoperability with existing solutions in the same application domain. We support a wide range of standard file formats such as

HTML, RTF, GIF, and JPEG as well as Microsoft's proprietary file formats because of their widespread use. OpenOffice.org provides the highest quality document import and export functionality for the various office productivity applications produced by Microsoft.

Documents formats

ASCII CSV
ASCII Text
dBase
DIF
Encoded Text
HTML
Lotus 1-2-3 1.0 DOS
Lotus 1-2-3 1.0 Windows
Lotus Freelance
Microsoft Excel for Windows 5.0
Microsoft Excel for Windows 95
Microsoft Excel for Windows 97/2000
Microsoft Word for Windows 6.0
Microsoft Word for Windows 95
Microsoft Word for Windows 97/2000
PowerPoint 97/2000
Rich Text Format (RTF)
SYLK
Text DOS
Text OS/2
Text Unix
Text Win

Graphic formats

Adobe Photoshop (psd)
AutoCAD (dxf)
CompuServe Graphics (gif)
Computer Graphics Metafile (cgm)
Encapsulated PostScript (eps)
JPEG Bitmaps (jpg)
Kodak Photo-CD (pcd)
Macintosh PICT (pct)
MS Windows Metafile (EMF)
OS/2 Metafile (met)
Paint Brush (pcx)
PBM
PGM
Portable Network Graphics (png)
PPM
SGF
SGV
SUN Raster-Format
TGA
TIFF-Bitmap
Truevision TARGA (tga)
Windows Bitmap (Bmp)
Windows Metafile (wmf)
XBM
XPM

Component Technology

Despite the rise of the graphical user interface and a constant flow of new innovations in the software industry, software development has become only more complicated over the years. Creating high quality applications increases demands on the user, which in turn lead to more and more expendable programs. Component based program development systems promise a welcome change in this trend. They are based on the JavaBeans architecture or the Component Object Model (COM), which are also called Component Ware. The programmer can access previously prepared components (building blocks) with these products, and use them in applications. The components are ready for immediate use once they have been adapted to the operational area.

The authors of the StarOffice product experienced this situation three years ago when developing a new software architecture, which for the first time made an office suite available on different platforms for use as an important building block. In the subsequent planning and development period, a need for the following attributes in this new architecture was realized:

✍**Consistent Interface Allocation**

An important aspect in Component Ware development is the way the overall application is dissected into individual units. The programmer must be offered exactly the needed number of single components. The more complex the application, the more difficult it is to distribute these components reasonably. The simpler and more logical the application, the higher its acceptability.

- **Support for Various Component Standards**

  While the COM standard in Windows software is the most widely known component standard, the JavaBeans components and CORBA interfaces are among the most common in the professional business field and are accepted as standards in the open world. Since there is not yet a general established industry standard in this area for all platforms, the component model has to provide bridges between these different technologies.

- **Localization-Transparent Components**

  Current Component Ware extensions should take advantage of the network for component communication. Pure client components are inapplicable for modern network use. Modern Component Ware must allow for delegation of a portion of an application to a central server. On the other hand, components running locally on a desktop system should not lose performance because of network communication protocols when all components are running on one machine.

- **Batch Job Ability**

  Previous Component Ware products employed a visual and user fixed basic approach for batch operations. User queries with hard coded components can bring the complete server operation to a halt. For batch operations within the server it should be possible to process content without requiring a visual representation.

- **Platform Independence**

  Especially in these times of network computers and operating system upheaval, it is important that modern Component Ware be able to select a platform independent extension. The interface should be defined in a platform independent way so that program code can be easily ported.

The OpenOffice.org Component Technology

The OpenOffice.org suite provides a component technology named Universal Network Objects, which adheres to all these requirements of modern Component Ware and it is formed on the Object Technology level, which is the basis upon which the OpenOffice.org API is set up.

This component technology is:

✍**Open**

It supports all the popular component standard communication protocols such as CORBA, JavaBeans, OLE Automation (Windows Scripting Host, Visual Basic, Delphi, and so forth), JavaScript, Phyton, Pearl, etc. scripting languages, as well as native integration in the C++ and C programming languages.

✍**Object Oriented**

It is object oriented and therefore supports concepts such as aggregation, inheritance, exception handling and polymorphism.

✍**Interface Based**

Its functions are integrated into various interfaces. Function areas of similar structures have access to the same interfaces, so that the programmer can easily feel at home in the component world.

✍**Platform Independen**t

It is specified to be platform independent and is available on all platforms that run OpenOffice.org.

✍**Exception Able**

It offers exception ability, which means that it can allow itself to be mapped onto the inserted development system mechanism - for example onto C++ Exceptions and Java Exceptions.

✍**Development System Independent**

It can be used with all current development environments and programming languages, including C++, C, Visual Basic, Windows Scripting Host and all systems which support COM, CORBA, JavaBeans componenets, and OLE Automation.

✍**Network Able**

Components based on the component technology can communicate on a network and can also delegate functions on a remote server, for example, to offer access to the complete text processing functions on Internet appliances.

Openness

Opening up all specification, file formats, technologies and last but not least the source code, will help ensure that no one can be locked into an application, platform or environment space. But just providing all this information to everyone is not enough. Openness will also required that already existing open standards and technologies were used when ever appropriate and also the project over time will evolve newer technologies and adapt other open standards.

XML File Format

We adopted XML to replace the old binary file format and become the OpenOffice.org suite's new native file format. Our goals were twofold: to have a comp lete specification encompassing all components, and to provide an open standard for office documents. One single XML format applies to different types of documents - e.g., the same definition applies for tables in texts and in spreadsheets. XML is ideal as an open standard because of the free availability of XML specifications and DTDs, and XML's support for XSL, XSLT, Xlink, SVG, MathML, and many other important and emerging standards.

Beside replacing the binary file format with XML, the OpenOffice.org suite will use XML internal for exchanging any type of content between the different applications. OpenOffice.org provides today an infrastructure for using different XML components. The XML-Parser and the XML-Printer are all implemented as components. Every of these component support the Simple API for XML (SAX). This infrastructure will allow in the future to dynamically configure a pipelines of different XML components, like XML-Parser, XSLT-Processor, etc. to process XML-Input and Output. This will allow transformation of XML-Data into different formats on the fly, without storing intermediate files and parse them again for every transformation step.

Application Programming Interfaces

The OpenOffice.org API is based on the OpenOffice.org component technology and consists of a wide range of interfaces defined in a CORBA-like IDL.

While the component technology determines *how* the components or applications communicate with each other, the OpenOffice.org API defines the interface for accessing office functionality from different programming languages. This interface structure is very important in determining the degree to which re-application of a development is possible.

The interfaces defined by the OpenOffice.org API are characterized as follows:

- They are completely defined component **interfaces** with the environment.
- They are **version independent** and **scalable**.
- They are **durable.**
- They are **re-applicable**.

Unlike other office suite APIs, the OpenOffice.org API does not simply reflect the features of preexisting implementations. Rather, it has been designed from the viewpoint of application and component developers. It offers programming interfaces for nearly all OpenOffice.org components and makes it possible to integrate new components.

Application Areas

There are multiple ways to use OpenOffice.org APIs. First, there is the typical macro programming for running certain tasks automatically. Secondly, parts of OpenOffice.org can be run as components of other programs; e.g., OpenOffice.org components are accessible as JavaBeans components.

A more advanced application is to modify OpenOffice.org components by wrapping them into replacement components or integrating completely new components with OpenOffice.org.

A very interesting application area is to replace the user interface of OpenOffice.org and build a completely different application domain.

Design Principles

Some principles that are important in all our designs are:

- **orthogonality**

  The API consists of interfaces which can easily be combined to serve special needs of certain objects.

- **scalability**
  We distinguish between functionality that is commonly needed and that which is required by specialized versions. Developers can always start with a minimum set of interfaces and add more step by step to embody more features.

- **reusability**
  We avoid creating specialized interfaces when a generic version is possible.

- **Remote usability**
  Services can be used efficiently from different processes or even different machines.

- **multithread enabled**
  Services can be used from multiple threads.

Architectural Paradigm

Our architectural decision is **Interfaces and Support-Classes** instead of **Implementation Inheritance**.

Interfaces and Support-Classes means that objects communicate only by interfaces. Support classes are used for recurring implementations. This was the choice for our design because components are highly independent of environment, language and version

Implementation Inheritance means partly implemented base classes from which specialized classes derive interface and implementation. This paradigm was not our choice, because in larger systems this leads to fat interfaces or deep inheritance hierarchy. In addition, components depend on the environment, mostly via their base class, and are programming language dependent and highly version dependent.

Object Model

The OpenOffice.org API is designed for and implemented using the OpenOffice.org Component Technology. Therefore the OpenOffice.org API is programming language independent and can be used from C/C++, Java, and several scripting languages. For other languages, only a language binding needs to be provided to access the whole OpenOffice.org API. The API is made up by following stereotypes:

- **implementation classes**
  These are not actually part of the API, but are mentioned here for better understanding. Implementation classes are implementations of services using a real programming language. Normally developers who use services do not have to deal with the implementation itself on the API level. Objects are usually not translated into language concepts for an application that uses them, but rather for the implementer of the class. A good example is an implementation class similar to a concrete Java-class.

- **services**
  Specifications of objects are called services. One can think of a service as a contract which is beneficial to both sides: the implementation class that supports certain

services and the application that uses this component for those services. Services usually describe the interfaces which they implement and a set of their properties. Although services are normally not translated into language concepts, a service may be considered to be similar to an abstract Java class.

- **interfaces**

  Specifications of a single aspect on an API level are called interfaces. A service can be considered to be a legally proven text module for contracts. Services can be combined to create contracts. Interfaces are very much like Java technology interfaces.

- **structs**

  Plain data blocks are specified as structs . Structs, therefore, do not have methods. The advantage of structs is that they can transferred as they are to a different process or even a different machine, which increases efficiency of interprocess and remote calls. With Java technology, structs are represented as a class which consist only of data members and get and set methods.

- **exceptions**

  Exceptions are extraordinary results from method calls. Exceptions are used for error handling just as in the Java technology.

- **constants/constant groups/enums**

  Constant values are split into two categories: constants - which can be grouped and can have numeric or character string values, and enums - which contain a fixed set of numeric values. In the Java technology, both are represented as classes with constant data members.

Common Design Patterns

The OpenOffice.org API uses heavily designed patterns, which provide a very consistent overall design. Some examples of application domain unspecific design patterns are:

- **Factory environment/container**

  New instances of services are created using factories. Factories of contents can emanate from the container object or from the environment. For efficiency, factories for iterators of all kinds, including cursors, are provided by the container.

- **Property Sets/-Access etc.**

  A set of interfaces makes the properties specified in the services accessible. Properties can be viewed as non-structural data members of objects, such as color or font. The variety of access interfaces covers the spectrum from convenient local access to fast remote access.

- **Collections/Containers**

  A collection in our terminology gives access to a set of similar sub-objects. A container allows replacement, insertion and removal of these sub-objects. A wide variety of predefined interfaces is available for this design pattern.

- **Enumerators/Iterators/Cursors**
  Enumerators, iterators and cursors are used for efficiently listing contained objects. Cursors in text play an especially major role, because indexing of text is not reasonable.
- **X...Supplier**
  To access structural but optional data members, we frequently offer supplier interfaces which in many cases only have one get method.
- **Events**
  Where it is of interest to get notification about certain status changes, there are services that offer methods to register and unregister listener interfaces. When the event occurs, a method at the listener interface is called and the event is given as an argument. This is the same as in the event concept in JavaBeans.
- **Exceptions for Error Handling**
  Exceptions constitute our principal error handling concept. For asynchronous calls, exceptions are transferred in callback events.

Module Categories

The OpenOffice.org API is organized in a hierarchical module concept which follows Java technology package or CORBA conventions.

- **office specific interfaces**
  e.g. For text documents, spreadsheet documents, drawing and presentation documents
- **integration framework**
  These interfaces make it possible to integrate new components into OpenOffice.org, e.g.: configuration management , Universal Content Broker
- **application domain independent**
  This very important category contains interfaces for property access, collections and contains, or streaming operations, as well as for attaching scripting engines and many more interfaces.
- **component system**
  The base of the suite is the handful of interfaces which are necessary to deal with object model topics such as lifetime control, querying interfaces, building bridges and instantiating remote objects.

Summary

The results and benefits of the OpenOffice.org source projects principles and activities to date are manyfold. One can build new applications from components without modifying source code. One can use a familiar programming language. The entire API is documented in a reference manual – `http://soldc.sun.com/staroffice`. A first release of a Development Kit, containing support of the OpenOffice.org Basic and also Java technology, is available.

Architecture

The OpenOffice.org source project is based on an architecture that can provide comprehensive personal productivity to different UNIX-based systems and may be ported many other platforms as well. This is because the whole technology is based on a platform-independent approach. Less than 10% of the code is platform dependent – this acts as an abstraction layer for the upper software components. Because of the availability of C++-Compilers on every major platform, C++ is used as an implementation language. This allows to port the OpenOffice.org technology to a wide range of different platforms. The decision for an object oriented language gives the OpenOffice.org source project the opportunity deliver a fully object oriented architecture.

The following information will give just a rough overview over the overall architecture. Some components of the OpenOffice.org source project like the help-system or the setup application are not covered here. Many parts of the OpenOffice.org source project consists of more than one CVS module. In many cases one block in the architecture is covered by more then five CVS modules in the source tree.

Layered architecture

The whole architecture is based on a layered approach. There are four defined layers where each covers a special area of the functionality.

- **System Abstraction Laye**r
  This layer encapsulate all system specific APIs and provide a consistent object oriented API to access system resources in a platform independent manner.

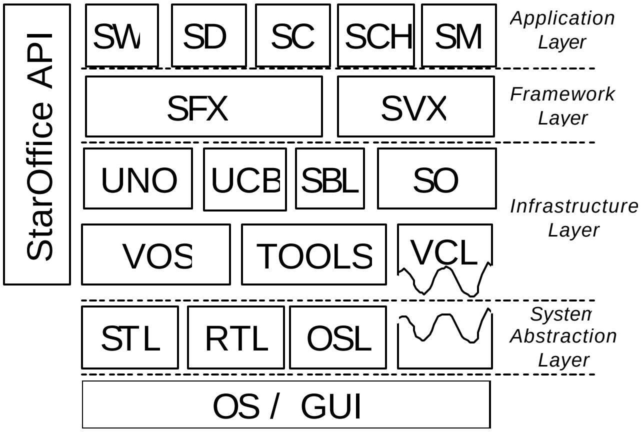
- **Infrastructure Layer**
  A platform independent environment for building application, components and services is provided by this layer. It covers many aspects of an object oriented API for a complete object oriented platform including a component model, scripting, compound documents, etc.

✍**Framework Layer**

To allow the reuse of implementations in different applications the layer provides the framework or environment for each application and all shared functionality like common dialogs, file access or the configuration management

✍**Application Layer**

All OpenOffice.org applications are part of this layer. The way these applications interact is based on the lower layers The chart shown below was created to depict the architecture of the StarOffice suite but it is the same for the OpenOffice.org suite:



System abstraction layer

The layered approach of the system architecture is one of the important facts to allow the easy porting of the technology to wide range of different system platforms. For this the architecture defines a virtual layer witch is called the System Abstraction Layer (SAL). All platform depended implementation take place below this layer or are part of some optional modules. In an ideal world an implementation of the SAL specific functionality and recompiling the upper layer module will allow you to run the applications. To provide the whole set of functionality the optional platform specific modules, like telephony support or speech recognition, have to be ported, too. To reduce the porting effort the set of functionality provide by the SAL is reduced to a minima set available on every platform. Also for some system the layer includes some implementations to emulate some functionality or behavior. For example on systems where no native multi threading is supported, the layer can support so called “user land” threads.

At this time the implementation of the platform dependent and independent parts of the graphical library is linked into one dynamically loaded shared library. So there is no well defined set of libraries which build up the SAL.

Operating system layer

The operating system layer (OSL) encapsulate all the operating system specific functionality for using and accessing system specific resources like files, memory, sockets, pipes, etc. The OSL is a very thin layer with an object oriented API. In contrast to the upper layer this object oriented API is a C-API. This will allow to easily port this layer to different platform using different implementation languages. For embedded systems or internet appliances for examples an assembler language can be used to realize the implementation.

Runtime library

The runtime library provides all semi platform independent functionality. There is an implementation for string classes provide. Routines for conversion of strings to different character sets are implemented. The memory management functionality resides in this module.

Standard Template library

As a generic container library the standard template library is used. It supplies implementations for list, queues, stacks, maps, etc.

Visual Class library

The visual class library is one of the core libraries of the OpenOffice.org technology. The VCL encapsulate all access to the different underlying GUI systems. The implementation is separated into two major parts. One is completely platform independent an includes an object oriented 2D graphics API with metafiles, fonts, raster operations and the whole widget set use by the OpenOffice.org suite. This approach virtually guarantees that all widgets have the same behavior independently of the used GUI system on the different platforms. Also the look&feel and the functionality of the widgets are on all platforms the same.

Because of this design VCL doesn't encapsulate the native widgets or controls of the underlying GUI system. The platform dependent part implements a 2D-graphic drawing canvas which is used by the platform independent parts. This canvas redirect every functionality directly to the underlying GUI system. Currently there exists implementation for the Win32, X-Windows, OS/2 and Mac. The access to the printing functionality, clipboard and Drag&Drop is also realized inside the VCL.

Infrastructure layer

Virtual Operating System layer

To make the usage of system resources like files, threads, sockets,etc. more convenient the virtual operating system encapsulate all the functionality of the operating system layer into C++ classes. The C++ classes here offer an easy to use access to all system resource in a object oriented way.

Tools libraries

There are different small libraries building up a set of tool functionality. This includes a common implementation for handling date and time related data. There is in implementation for structured storages available. Other implementation provide a generic registry, typesafe management and persistence of property data.

Universal Network Objects

The so called Universal Network Objects are the component technology used inside the OpenOffice.org products. The component technology does not depend on any graphical subsystem, but is heavily based on multithreading and network communication capabilities.

The system consist of several pieces. An IDL-Compiler, which generates out of the specified definition of an interface a binary representation and the associated C-Header or Java technology files. The binary representation is platform and language independent and is at runtime used to marshall argument for remote function calls or to generate code on the fly for a specific language to access the implementation provided by the interface. This technique reduced the amount of generated code for the different language binding tremendously. The drawback is that not only for every language binding a specific backend for the code generation is needed, it is that for every specific compiler a bridging module is needed at runtime.

Many parts of the UNO technology are implemented as UNO components. This helps to create a very flexible system and also the extension of the system at runtime. For example by providing new bridges or communication protocols. UNO provides transparent access to components over the network or locally. For the communication over the network IIOP can be used. If the component are realized as shared libraries the component can be loaded by UNO into to the process memory of the application and every access of the component is just like a function call without any marshalling of arguments which is required for remote function call.

Universal Content Broker

The Universal Content Broker allow all upper layers to access different kind of structure content transparently. The UCB consists of a core and several Universal Content Providers which are used to integrate different access protocols. The current implementations provides content provides for the HTTP protocol, FTP protocol, WebDAV protocol and access to the local file system.

The UCB does not only provide access to the content, it also provides the associated meta information to the content. Actually there is synchronous and asynchronous mode for operations supported.

OpenOffice.org Compound Objects

The Compound Object implementation provide the functionality to build compound documents, where for examples a spreadsheet is embedded in a word-processing document.

The current implement provides a platform independent implementation of all this functionality for compound documents and for embedding of visual controls like multi media players or different kind of viewers. All content of compound document is stored in a structured storage. The current implementation is compatible to the OLE structure storage format. This allows access OLE compound documents on every platform where OpenOffice.org is available. On the Windows platform the implementation interact with the OLE services and will so allow a tight integration of all OLE capable applications.

OpenOffice.org Scripting and Basic library

The scripting functionality coming with the OpenOffice.org suite is a BASIC dialect featuring an interpreter that parses the source statements and generates meta instructions. These instructions can be executed directly by the supplied meta instructions processor or can be made persistent in modules or libraries for later access. All functionality supplied by the upper level application components is accessed via a scripting interface in the component technology. This will help ensure that new components using the OpenOffice.org component technology can be fully scriptable without spending a huge amount of effort.

The scripting interfaces are also implemented as components which will allow an easy integration of other scripting languages. The interfaces provide functionality like core reflection and introspection similar to the functionality by the Java platform.

Framework layer

OpenOffice.org Application framework library

The Application framework library provides an environment for all applications. All functionality shared by all application and not provided by any other layer is realized here. For the framework every visual application has to provide a shell and can provide several views. The library provides all basics functionality so only the application specific features have to be added.

The Framework is also responsible for content detection and aggregation. The template management is provided here and the configuration management too. The Framework is in some areas related to the compound documents, because of the functionality for merging or switching menu- and toolbars. Also the capability for customization of all applications is provided by the library.

SVX Library

The SVX library provides shared functionality for all applications which is not related to a framework. So part of the library is a complete object oriented drawing layer which is used by several applications for graphic editing and output. Also a complete 3D-rendering systems is part of the drawing functionality.

The common dialogs for font selection, color chooser, etc. are all part of this library. Also the whole database connectivity is realized here.

Application layer

All applications like the wordprocessor application, spreadsheet application, presentation application, charting application, etc. build up this layer. All this application are realized as shared libraries, which are loaded by the application framework at runtime. The framework provides the environment for all this application and also provides the functionality how these application can interact.

Build Environment

The following information provides an overview of the proposed Build Experience for OpenOffice.org and how we hope it will evolve. This information is intended for developers and engineering managers.

Open Source projects

Open Source projects follow by now a very familiar pattern when it comes to providing a Build Experience. This translates as follows:

```
1. cvs checkout OpenOffice
2. ./configure
3. make
4. make install
```

The build tools are well known to the Open Source community - i.e., gmake, autconf, etc. This is the Build Experience that we want OpenOffice.org source project developers to work within. Not only should this be the sum total of the commands involved, but more importantly milestone builds should build cleanly on all supported platforms.

The OpenOffice.org technology in its first release under the OpenOffice.org source project will not support the mode described above. The mode initially supported will not be far from this model, and it will contain the foundation for evolving to this model. The only implication is that build tools may not be the ones most familiar to the open source community. Sun is committed to providing a product that builds. This document will describe this Build Experience.

It is important to understand that the Build Experience will be an evolving model, one that will grow and gain from the experience of the Open Source Developer Community.

The task facing us is a challenging one. The OpenOffice.org suite is a very large application from any standpoint. It is a complex application consisting mainly of C++ code employing templates and exception handling and supporting independent language binding for a distributed component based architecture.

The Build Experience

The Build Experience will cover the following areas:

- Build Requirements
- Downloading the Source
- Build Prerequisites
- Build and Install Instructions
- Build Tools & Makefiles
- Build Environment
- Build Troubleshooting
- Porting to New Platforms
- Build Documentation and Infrastructure

Build Requirements

The OpenOffice.org sources will build on the Solaris(TM) operating environment, Linux, and WIN32 platform. Work is in progress on the Macintosh platform. Each system will describe the software and hardware requirements to build on their respective systems. Typical of these will the version of gcc, the Java Technology JDK , hard disk and RAM sizes.

It is crucial that these requirements be understood at the outset. Building the OpenOffice.org sources is not a task to be taken lightly. It may turn out to be the largest open source project ever in terms of source size and time to build. The source and build environment size is in the region of 328MB and consists of approximately 20,000 files. It can take up to 18 hours to fully build from scratch, but this is rarely needed. The build environment is prepared to allow working with milestone builds.

Downloading the Source

The source may be downloaded using CVS, or *ftp* can be employed to download gzipped tarballs of the source. In addition, milestone completed builds will be made available in gzipped tarball format plus daily snapshots of the source.

Build Prerequisites

The OpenOffice.org technology relies on a number of external sources to be built. Instructions on where to locate and download these will be available.

Build and Install Instructions

The OpenOffice.org technology will use autoconf to check the integrity of the build environment and to set up the correct build environment on the supported platforms.

After setting up the build environment, the build and install instructions will support the make and make install concepts. Details on building all the source as well as build components will be available.

Build Tools & Makefiles

The OpenOffice.org technology is built using dmake from the http://dmake.wticorp.com . This is itself an open sourced project. The syntax is a make like syntax. The dmake options will support a dmake all, dmake <component> and dmake install concepts.

The OpenOffice.org source project team has also developed a number of development tools including support for Interface Definition Language formats, resource pre-processors and bitmap creators. All these will be fully explained.

Build Environment

The OpenOffice.org technology relies on a large set of environment variables as well as compiler pre-processor options and flags. These will be documented to support greater understanding of the build environment among developers

The OpenOffice.org technology is a CVS module based source tree. These CVS modules will have an overview explanation plus an explanation of the order in which they are built.

Build Troubleshooting

A troubleshooting guide and Build FAQ will be published to support developers.

Porting to Other Systems

The OpenOffice.org technology for its supported platforms will rely on a bootstrap process that is in place in order to build it. To port the OpenOffice.org technology, developers will need to know how to create this bootstrap in the first place.

Build Documentation & Infrastructure

An online Build manual will exist to cover all the topics described above. There will also be a support infrastructure in place to deal with specific build issues. Bugzilla will be employed to store all bugs and issues arising from the build.

Outlook

Providing a clean Build Experience which is familiar and consistent to the Open Source Community is not only our aim but we see it as critical to the success of this project. Although this experience has not yet reached full fruition, Sun has determined not to delay the release of the source for this reason. It is more important that we engage with the community than to wait until we have replicated the same build experience. To this end we are committed to providing as clean a build as possible now while putting an infrastructure in place to allow us to evolve to a more standard model.

Current Source Tree

The OpenOffice.org source project will be among the largest open source projects. There are about 70 CVS modules with more than 20,000 files providing about 7,600,000 lines of code.

Appendix A gives a first overview over all CVS modules with a short description of the purpose. Additional information will be made available at the website http://www.openoffice.org .

Future Steps

An Open World Component Technology

Current Situation

Nowadays there is no Component Technology in the Open world which is accepted by everyone. There exist different open source projects which are based on a component model, but each of them uses its own concept or technology.

In the Desktop domain there is KDE and GNOME. Both of them are using CORBA as a low-level communication layer but the component models on top are completely different in each project. So today it is essentially impossible to write a component which can be used in both desktop environments. Also the current CORBA specification doesn't include a model for compound documents or other desktop software related functionality. So there are different efforts underway to overcome these problems, but they lack interoperability. Another issue with most CORBA implementations is that these were designed to run distributed network based applications using remote services. So using such CORBA implementation with all the network communication overhead in an infrastructure for a desktop component model, which in most cases will run on one local machine, will be in most cases the wrong approach.

When the Mozilla open source project started, there was a requirement to have a lightweight component model which could be easily ported to different platforms. So XPCOM was created.

The reasons why the OpenOffice.org suite currently employs its own component model are all based on these issues. Additionally, there is the requirement to provide a similar set of office suite functionality and technology to the widely used Microsoft products in this domain.

A Short Term Solution

As discussed above, there is no component technology available which is well accepted, portable and used prevalently. So to to provide office functionality in different environments, it is necessary that different component technologies be supported. Supporting different component technologies is difficult but possible, but supporting different concepts or philosophies is quite impossible.

Because OpenOffice.org's component technology is able to use software bridges to access other components' worlds, it is possible to integrate the office components into different environments, such as GNOME and Mozilla. The bridging can be easily done in an environment where the basic concept for the components are quite similar. In any other case in can be very hard to provide this bridging functionality hiding all conceptual differences. In most cases developers and users will have to deal with the different concepts and philosophies. One of the annoying problems for example will be the different naming convention. Also if a bridge needs to deal with a complex conversion of arguments and calling conventions, performance issues could arise.

So in the near future we will provide bridges from OpenOffice.org component technology to XPCOM and Bonobo. OpenOffice.org components can than be used in these environment as integrated components.

Vision

The past has shown the importance of a homogenous component technology and an object model which integrates all software components into one easy-to-use building set. The usage existing building blocks in different development environment like C/C++, Java technology, Phyton, Pearl, VB, etc. should be transparently provided by a component technology.

The object model should define the reuse of existing interfaces and components and the matching of concepts over a wide array of applications. It should define common base concepts that will be used no matter what component is being reused or what application is being built.

One challenge for the Open World will be to provide one efficient component technology and a homogenous, superior object model for the desktop, the office productivity suite, and the browser.

Unified Component Ware

A unified Component Technology should fulfill all requirements gave rise to today's different solutions. This will not guarantee that the technology will fulfill all future requirements, but it will allow the replacement of the existing solutions over time and it will increase acceptance by the Open World. To ensure a high level of acceptance, open standards should be used whenever appropriate.

CORBA is an open standard that is widely accepted for building heterogeneous distributed systems. On the other hand the CORBA specification falls short in the areas of compound documents and small components for personal productivity. This has changed to some extent with the CORBA 3.0 specification, but there are still open issues. For a desktop environment there is a need for support of in-process components and for local communication features, which are today still open to implementation in the CORBA environment. To the extent that these features are transparent to all components, there is no need for a clear specification, but in most cases today the developer of the component and the developer of the application using these components has to deal with this situation. Another problem for an open component system is the lack of a technology and a concept for writing cross-platform graphical components.

ORBit is the CORBA implementation used by most of the GNOME components. It is fast and lean, allowing the use of CORBA in areas that would not normally seem practical. It supports much of the CORBA 2.2 standard, and has hooks that allow easy integration with GNOME programs. ORBit provides language binding for Perl, C++, Tcl, Python, ADA and Eiffel, but currently not for Java technology. It allows C and C++ objects in the same address space to short-circuit calls (i.e. no on-the-wire marshaling) for maximum speed. The ORBit is a good candidate starting point for an open component technology. It could provide the functionality for all communication infrastructure and will allow interoperability with CORBA. While CORBA is an accepted standard, the ice Component Model provides the ability to bridge easily from one component environment to another, which will for example allow the usage of components in the Microsoft world. Also, the Mozilla XPCOM technology is a very lightweight and lean system that it could provide some advantages for an overall system.

On top of this communication and low level object infrastructure there should be a clear definition of interfaces for commonly used functionality, such as factories and reference-counted objects. There is also a requirement for a clear concept for lifetime management of components, activation, security and the integration into a graphical subsystems. All these interfaces should be designed for use in many different application domains.

Especially for personal productivity tools and desktop environments, we need a clear concept and definition of interfaces to build up an infrastructure for:

- Compound Documents and Component Reuse
- Document Models
- Content Storage
- Monikers
- Control Components
- Component activation (Shared library components, process-based components and distributed components)
- Printing

It is important that these specifications be useful in many application domains, and that standards that cover existing areas be used and supported.

Not every developer wants to deal with all the details of a component model. Therefore, to achieve acceptance, it is important to provide helper libraries for different languages that make the usage and the implementation of components as easy as possible.

Outlook

We have started an initiative for a new homogenous component technology by talking to several developers in different open source projects. We will work together with different groups to provide a leading edge component technology for a desktop environment and distributed applications. To the extent that standards are available and meet the requirements, they will be used. We would like to see the best pieces of today's existing technologies (e.g. Bonobo, XPCOM, UNO ...) be integrated into this technology.

Also we would like to start an effort with several open source projects to define the concept, the naming conventions and philosophy for an Open World component mode. By the time the component model is available it will be used by all OpenOffice.org components, and the OpenOffice.org API will be adapted over time to meet the specification of the new component model. In the future this will allow writing highly sophisticated applications and solutions by using existing components exclusively.

Appendix A

Project	Modules	Description
Xml file formats		
	xmloff	*Import/Exportfilter for XML*
	sax	*SAX UNO -components for xml-parsing and writing.*
L10N		
	i18n	*Internationalization Functionality.*
	transex3	*L10N Tools*
openoffice.org wordprocessor application		
	sw	*OpenOffice.org wordprocessor application*
	starmath	*OpenOffice.org math application*
	lingu	*Linguistics stub*
Openoffice.org spreadsheet application		
	sc	*openoffice.org spreadsheet application*
	scaddins	*openoffice.org spreadsheet application addins*
Graphic applications		
	sd	*OpenOffice.org drawing application*
	sch	*OpenOffice.org charting application*
	goodies	*Collection of graphic filters and 2d and 3d drawing*
	Svx	*Collection of graphic layers and Collection of API used by all OpenOffice.org Applications*
Database Access		
	sdb	*Database driver layer.*
	dbaccess	*Database access layer*
	connectivity	*OpenOffice.org Base Connectivity*
Porting		
	sal system abstraction layer	*Low level API for the integration of all supported platforms.*
Buildtools and Buildenvironment		
	solenv	*Buildenvironment tools*
	dmake	*Make application*
	rscpp	*Preprocessor for the Resource Compiler.*
	xml2cmp	*Processor for Uno-ComponentDescriptions*
	jtools	*Java technology helper applications*
	config_office	*Helper for buildenvironment configuration*
Graphic System Layer		
	vcl	*Visual Class Library*
	rsc	*Resource compiler*

Project	**Modules**	**Description**
	toolkit	*VCL Implementation of the UNO Toolkit and the UNO Controls*
	UnoControls	*UNO controls*
	forms	*Forms implementation*
Printing		
	xprinter	*Stubs for the Bristol X-Printer*
Scripting Engines		
	basic	*Basic interpreter and the basic runtime library*
	basctrl	*Basic IDE*
Utilities		
	tools	*Common used base classes (string, date, time, streams, ...)*
	svtools	*Collection of Patterns and help classes*
	std2	*STL port - a derivative from the SGI/STL.*
	io	*UNO I/O services*
	eventattacher	*Component based event handling*
	unzip	*Compression library*
	unotools	*UNO helper classes*
	extensions	*Additional components*
	external	*External references*
	configmgr	*Configuration management*
	sot	*Compatible storage implementation.*
Installation		
	setup2	*Setup Application*
	scp	*Packaging scripts*
	Scptools	*Packaging tools*
	Instsetoo	*Installation set creation*
	Readlicense	*Readme and license texts*
	Extras	*Demo documents, help files, resources...*
	Wizards	*Wizards*
UCB (*Universal Content Broker)*		
	chaos	*Universal Content Broker*
	inet	*Internet transport protocols (FTP, HTTP, LDAP, IMAP, NNTP, POP3, SMTP).*
	uui	*UCB Graphical User Interface Components*
	ucbhelper	*Helper classes for UCB users and Content Provider implementers*
	store	*Reliable, recoverable storage filesystem*
	ldapber	
StarOffice API		
	api	*IDL definitions of all interfaces of the OpenOffice.org API*
	offuh	*Generates UNO headers.*
UDK (Uno Development Kit) / Componenet Technology		
	cppu	*C++ UNO core, C++ bridges*
	unoidl	*Interface Definition La nguage compiler.*
	cppuhelper	*C++ UNO implementation helpers*
	javaunohelper	*Java technology Uno helper implementations*
	jurt	*Java technology UNO Runtime*
	bridges	*UNO bridges from any language*
	remotebridges	*UNO interprocess bridges*
	stoc	*OpenOffice.org components - basic UNO services*
	cpputools	*Collection of UNO utility and runtime programs*
	registry	*Registry*
	codemaker	*IDL compiler backend*
	rdbmaker	
	vos	*Object orientated Framework above sal. This module encapsulates sal in C++ classes for ease of use.*
Object Integration		
	So3	*Compound document implementation*
	Sj2	*Integration of embedded objects*
	Ie	*Integration of internet explorer*
Application Framework		
	Sfx2	*Application framework*

Project	Modules	Description
	offmgr	*Database components*
	res	*Bitmap resources*
	idl	*IDL compiler for the resources.*
	framework	*Provides Frames Hierarchy for logical managing of components/documents.*
	desktop	*OpenOffice.org (desktop) application*
	DocumentProperties	*Document properties handling*