OpenOffice.org
The Open Source Office Suite

Open Source Component Technology

Berry Xi-bei Jia
berry.jia@sun.com
SUN China Engineering and Research Institute
2003-11-14

Development in OpenSource Community

Introduction to OpenOffice.org Office Suite

The Open Source Office Suite

Everything you need in a world-class office productivity suite

- Writer - a word processor for creating dynamic documents
- Calc - a spreadsheet for analysing data
- Impress - for designing eye-catching presentations
- Draw - for producing dramatic illustrations
- Data Source tools - for opening up access to your databases
- works with MS-Office file formats

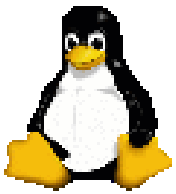
http://www.digitaldistribution.com/samples/openofficeintro11en.swf

OpenOffice.org
The Open Source Office Suite

System Requirements

Pentium-compatible PC
Microsoft Windows 95, 98, NT, ME, 2000, XP
32 Mb RAM (min) - 64Mb (recommended)
170Mb hard disk

SPARC Platform
Solaris 7 or 8
64Mb RAM minimum, 126Mb RAM recommended, 240Mb hard disk
OpenWindows or CDE
VGA or higher, at least 256 colours 800x600



Pentium-compatible PC or higher
Linux kernel 2.0.7 or higher; glibc 2.1.1 or higher
64Mb RAM, 170 Mb hard disk
X Server min 256 colours or greeyscale
VGA or higher, 256 colours 800x600 or better

OpenOffice.org
The Open Source Office Suite

Get It!

- Download from http://www.openoffice.org or local mirrors
- Local CD-Distributors (see web site)
- watch for magazine covers etc
- included free in many Linux distributions:
  - RedHat Linux version 8.0
  - SuSE Linux version 8.1
  - Mandrake Linux 9.0
  - Xandros

Development in OpenOffice.org Community

What is OOo?

- It is abbr. of OpenOffice.org, it is:
  - A website:
    - Home of the open source community and open source code development
  - A project:
    - the open source project through which Sun Microsystems has released the technology for the popular StarOffice Productivity Suite.
  - A product:
    - free full featured multi-lingual, multi-platform office suite developed by the open source source community.

Community Mission

" To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format".

OpenOffice.org
The Open Source Office Suite

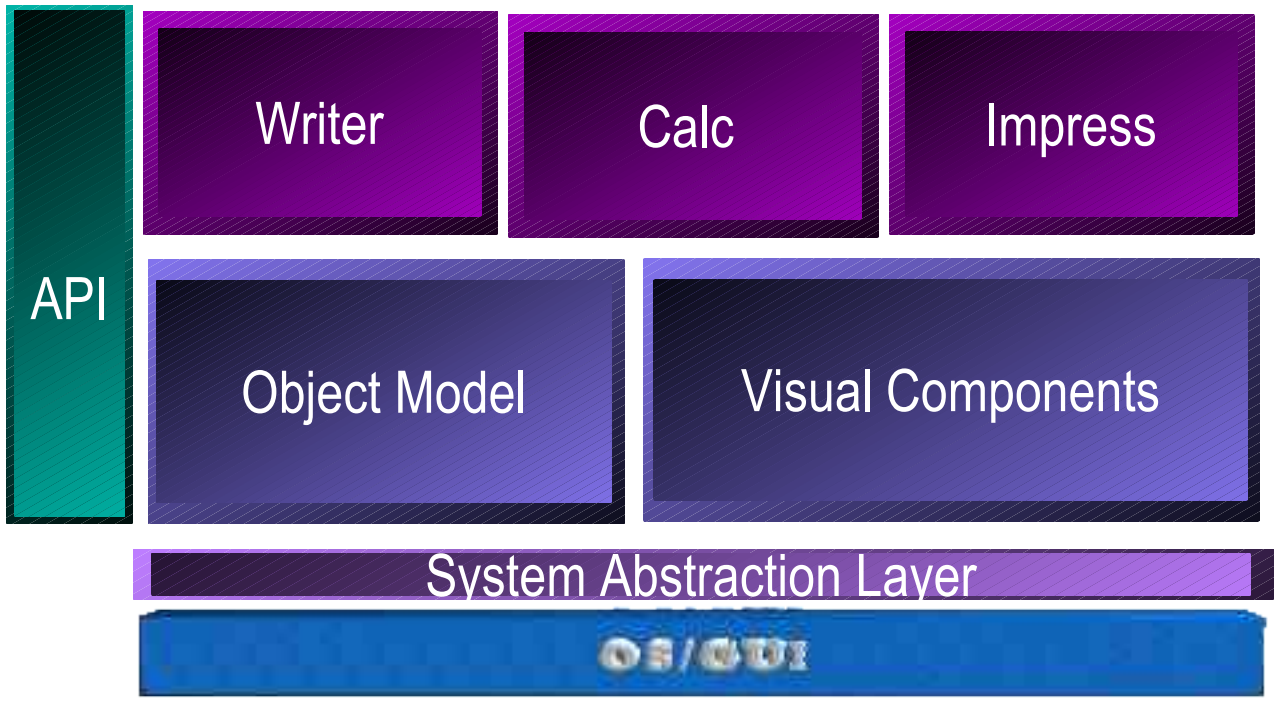
Infrastructure

- Hosted by Collab.Net
- Development Framework SourceCast
  - Web Content / Downloads
  - CVS
  - Mailing Lists
  - Issue Tracking System
- ~150 CVS Modules grouped in 30 Projects grouped in 3 Categories
- Bridging of ~80 Mailing List to Newsgroups

Milestones

- Aug. 1999 Sun buys StarDivision
- July 19th 2000 OpenOffice.org announced
- Oct. 13th 2000 OpenOffice.org live
- Jun 2001 ODK released
- Jul 2001 Marketing Project created
- May 2002 OpenOffice.org 1.0 Launch
- Sept. 28th 2003 OpenOffice.org 1.1 RC5 released
- Oct. 2003 OpenOffice.org 1.1.0 SDK released

Architecture



Contributions

- Patches received for code, config & docs
- Active development discussions on
  - Porting
  - Localizations
  - Marketing
  - Documentation
  - Native language projects

Accessibility and UI
API
XML
User and Developer Support
Distribution, CDs, Mirrors, OEMs

SUN Contributions

- Sponsoring for Hosting
- Community Management
- 2 Release Engineers
- Active Participation in Projects
- Large Number of Developers
- Issue Handling by QA Team

Other Company Contributions

- Ongoing
- Flash-Export (Verilogix)
- SVG-Import/Export (Batik)
- Native PostgreSQL/MS Access Drivers
- Debian, Mandrake, ...
- WordPerfect Filters (City of Largo)
- Potential
  - BiDi, ... (IBM)
  - Thai (NECTEC)

You can contribute...

- Even if you are not a software developer
  - just by using the product, documenting the bugs and issues encountered, and submitting these findings to the relevant project within OpenOffice.org.
  - the OpenOffice.org project leads want you to contribute your discoveries and suggestions to the project, which, if you contribute, is also your project.
- If you are a software developer
  - See the project page on OOo
- If you are interested in marketing
  - See http://marketing.openoffice.org/

Zones of Participation

- Mailing Lists
  - if you have some points to make, consider the "discuss" list, and if you have some questions about using the product, the "users" list.
  - Virtually every OpenOffice.org project also has its own discussion list, usually a "dev" mailing lists, and it is here where a lot of a project's work is done.
- Webmasters
  - Who are the webmasters? Louis Suarez-Potts, Zaheda Bhorat, and Max Lanfranconi; we are all community managers and each of us trades off on reading and moderating the discuss list
- IssueZilla

OpenOffice.org
The Open Source Office Suite

OOo By Numbers June 30

- 7.5 Million lines of code
- over 60,000 + members
- 7 M + Downloads
  - 600K source downloads
  - 6.4M binary downloads
  - Linux 39%, Windows 58%, Solaris 3%
- 20 + Languages available
- 30 Projects
- Ports
  - Windows, Linux x86, Solaris, Linux PPC
  - (FreeBSD, IRiX, Tru64, Mac OS X + in dev)
- 50 + Mirrors, 15 CD distros
- Native language projects
  - French, German, Italian, Spanish, Dutch +

Building tools

- Perl5
- glibc 2.1.x, Gcc 2.95.2, tcsh
- Jdk 1.2.2 and above
- Flex & Bison
- WIN NT
  - VC++ 6.0+
  - needs 4NT Command Shell
  - Cygnus Toolkit

Developing Tools...

- CVS
  - cvs -d :pserver:anoncvs@anoncvs.services.openoffice.org:/cvs login
- LXR
  - Browse and search the source repository.
- Bonsai
  - Find out who changed what in what file and when.
- Tinderbox
  - Shows tree status. Check this before checking out and checking in.

Developing Tools

- Bugzilla
  - Our bug tracking system. Use this to track and schedule bugs you're currently working on, as a scratch pad for development ideas and to manage code reviews.
- IssueZilla
  - OpenOffice.org uses a modified version of mozilla.org's BugZilla to track issues. CollabNet's BugZilla is called IssueZilla as it extends that bug-tracking system towards more generalized Issue tracking, which is a superset of bug tracking.

What Is IssueZilla?

- IssueZilla
  - a proprietary tool
  - based on Mozilla's open-source Bugzilla
  - but has been generalized by CollabNet to handle all kinds of issues, not just code-based issues.
  - On OpenOffice.org, the supported issue-types now include: DEFECT, ENHANCEMENT, FEATURE, TASK and PATCH.
- Reported issues will be assigned to an appropriate issue owner.

Why "Issues" And Not "Bugs"?

- Not all 'issues' are DEFECTS, which are more commonly known as "bugs." Some issues will be FEATURE Requests or ENHANCEMENT Requests, PATCH submissions, or even TASKS. For this reason, we feel the term "issue" is a more appropriate one than "bug".

- People often say 'bug' when they should really be referring to an issue. Enter the tasks you're planning to work on as enhancement requests and will help you track them and allow others to see what you plan to work on. If people can see your flight plan, they can avoid duplicating your work and can possibly help out or offer feedback

Archintecture of OpenOffice.org Source Codes

OpenOffice.org
The Open Source Office Suite

Ooo Architecture

System Abstraction Layer

- Abstracts all system specific APIs

Include:

- VCL:Visual Class Library
  - Encapsulates all access to the different underlying GUI systems
  - Provides completely platform independent and object oriented 2D graphics API and the whole widget set
- STL:Standard Template Library
  - as a generic container library
  - e.g. list,queues,stacks,maps,etc.
- OSL:Operating System Layer
  - encapsulates all the operating system specific functions
  - for accessing files,memory, sockets,pipes,etc
  - C-API instead of C++ API
- RTL:Runtime Library
  - semi platform independent functionality
  - e.g.memory management

Infrastructure Layer...

- Provides a complete object oriented platform

Include:

- UNO:Universal Network Objects
  - A component model
  - A inteface based object model
  - As efficient as COM with additioal features
  - All the components in framework layer and above are implemented as UNO components
- VOS:Virtual Operating System
  - Encapsulate all the functionality of OSL into C++ classes
- UCB:Universal Content Broker
  - Provides access to the content and associated meta infomation
  - e.g.HTTP,FTP,WebDAV,local FS
  - Synchronous and asynchronous

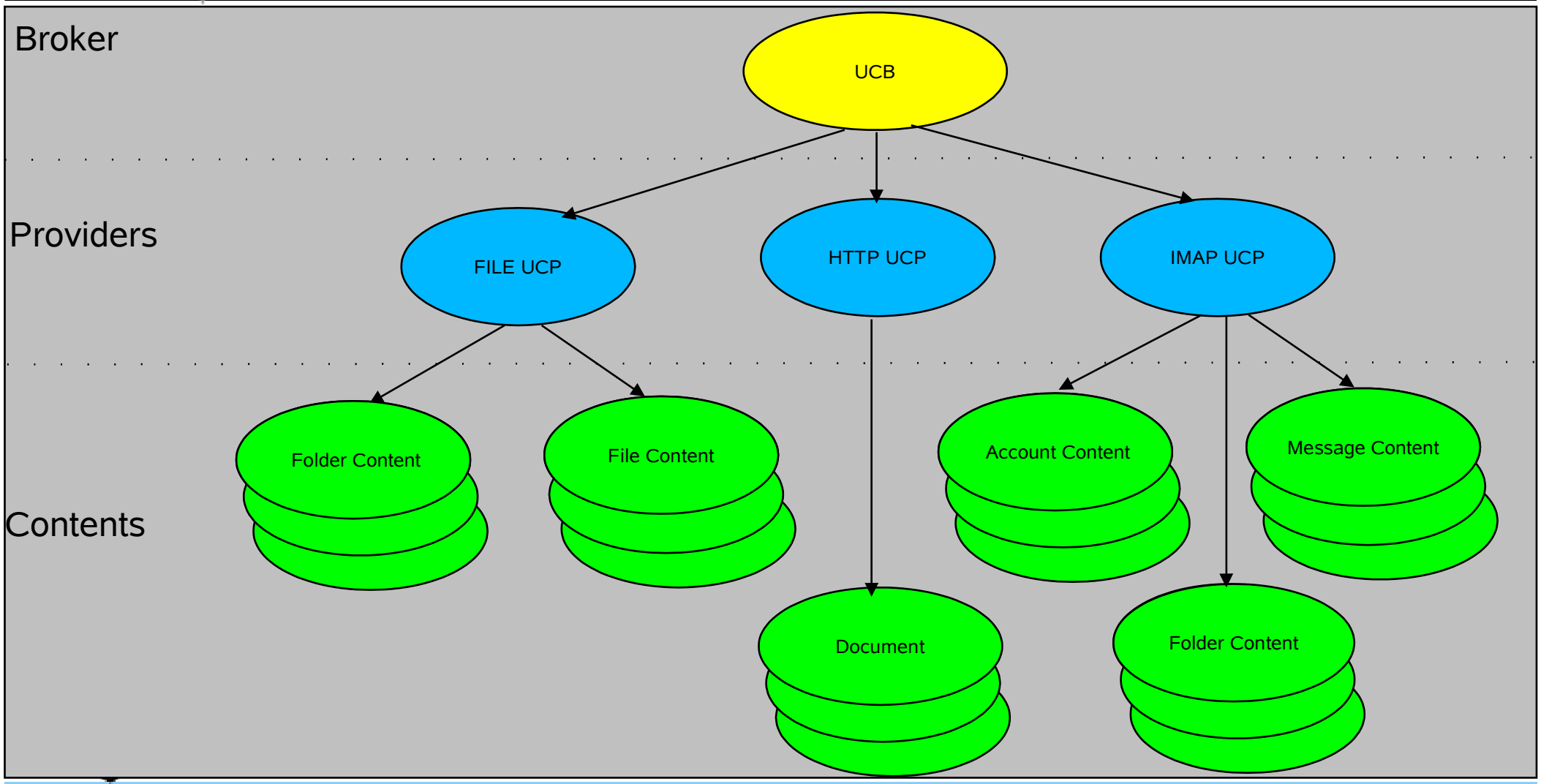
Infrastructure Layer

continue:

- SBL:Scripting and Basic Library
  - A BASIC dialect
  - Ensure new components using the component technology can be fully scriptable
  - Provides core reflection and introspection similar to that by Java
- SO:Compound Object
  - Provides the functionality to build compound documents
  - Compatible with OLE structure storage format
- Tools: tools libraries
  - Different small libraries such as:
  - Structured storage
  - Generic registry
  - Typesafe management
  - Persistent of property data

OpenOffice.org
The Open Source Office Suite

UCB Components



UNO

- Interface based object model
  - UNO defines a communication model among distributed objects
- UNO IDL: UNO Interface Definition Language
  - Defines interfaces with its own IDL
  - An UNO component can implement this interface in any programming language like C++ and Java
- UNO repository
  - The UNO component registers itself into a platform independent binary repository call UNO repository against a UNO service name

URE ...

- URE: UNO Runtime Environment
  - URE locates,instantiates and controls the life cycle of an UNO component requested by the UNO service name
  - URE uses the UNO repository to locate and instantiate the UNO components
  - In Process model:
    - If URE finds the target component is developed with same language and running under same process, the interaction between them is same as function call
    - If all StarOffice components are run in single process, there is no overhead in making function call from one UNO component to another.

URE

continue:

- Remote model:
  - URE can also locate and communicate with UNO objects instantiated in a remote host
  - URE can automaticaly marshal parameters if the target component is developed using different language or running on remote host
  - UNO components can make use of other componnets developed with CORBA/ COMd and vice versa using URE remote bridges for CORBA and OLE.
  - UNO object model is network aware so that the different StarOffice components can be running in different machines and interact seamlessly without any special modification

Framework Layer

- Provides the framework to share the reusable components to all app.

Include:

- SFX:Application Framework Library
  - Shell and views
  - Content detection and aggregation
  - Template management
  - Configuration management
  - Merging or switching menu and toolbars
  - Customization of app
- SVX: SVX library
  - Shared functionality for all app which is not related to a framework
  - Object oriented drawing
  - 3D rendering systems
  - Common dialogs for font selection, search and replace,transliteration etc.
  - Database connectivity
- I18N: Internationalization

Application Layer

- All applications are realized as shared libraries,which are loaded by the application framework at runtime.

Include:

- Star Writer:
  - The word processor app
- Star Calc:
  - The spreadsheet app
- Star Impress:
  - The presentation app
- Star Draw:
  - The drawing app
- Star Chart:
  - The charting app

UNO
Component Model
Technology

UNO Framework Overview

Universal Network Objects

- **UNO is**
  - **the interface-based component model of OpenOffice.org.**
- **UNO offers interoperability between**
  - **different programming languages**
  - **different object models**
  - **different machine architectures**
  - **different processes**
  - **either in LAN**
  - **or even via the Internet**

The Goal of UNO

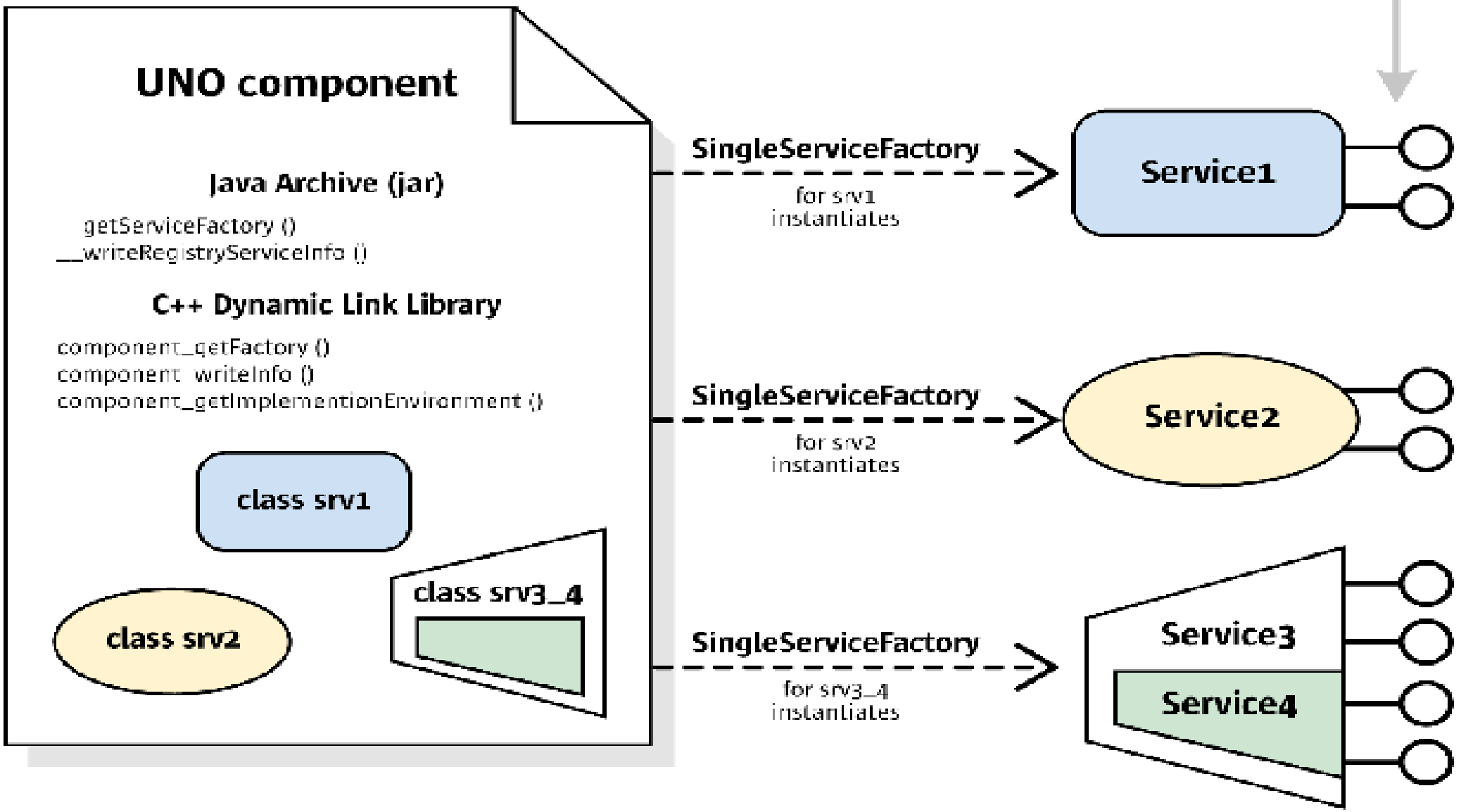
- provide an environment for
  - network objects
- across
  - programming language
- and
  - platform boundaries.
- UNO objects run and communicate everywhere.
- UNO reaches this goal by providing the following fundamental framework...

UNO components

- provide component operations to be called by the service manager and the component loader
- implement one or several UNO objects

Objects implement

- core UNO interfaces
- one or more services exporting additional interfaces



UNO Framework ...

- UNO IDL
  - UNO objects are specified in an abstract meta language, called UNO IDL (UNO Interface Definition Language), which is similar to CORBA IDL or MIDL.
- UNO Component
  - UNO objects in the form of compiled and bound libraries are called components. Components must support certain base interfaces to be able to run in the UNO environment.

UNO Framework ...

UNO Service Manager

- To instantiate components in a target environment UNO uses a factory concept. This factory is called the service manager.
- It maintains a database of registered components which are known by their name and can be created by name.
- The service manager might ask Linux to load and instantiate a shared object written in C++ or it might call upon the local Java VM to instantiate a Java class.
- This is transparent for the developer, there is no need to care about a component's implementation language.
- Communication takes place exclusively over interface calls as specified in UNO IDL.

UNO Framework ...

- UNO Bridge
  - UNO provides bridges to send method calls and receive return values between processes and between objects written in different implementation languages.
- UNO Remote Protocol
  - The bridges use a special UNO remote protocol (urp) for this purpose which is supported for sockets and pipes.
- UNO Runtime
  - Both ends of the bridge must be UNO environments, therefore a language-specific UNO runtime environment to connect to another UNO process in any of the supported languages is required.

UNO Framework ...

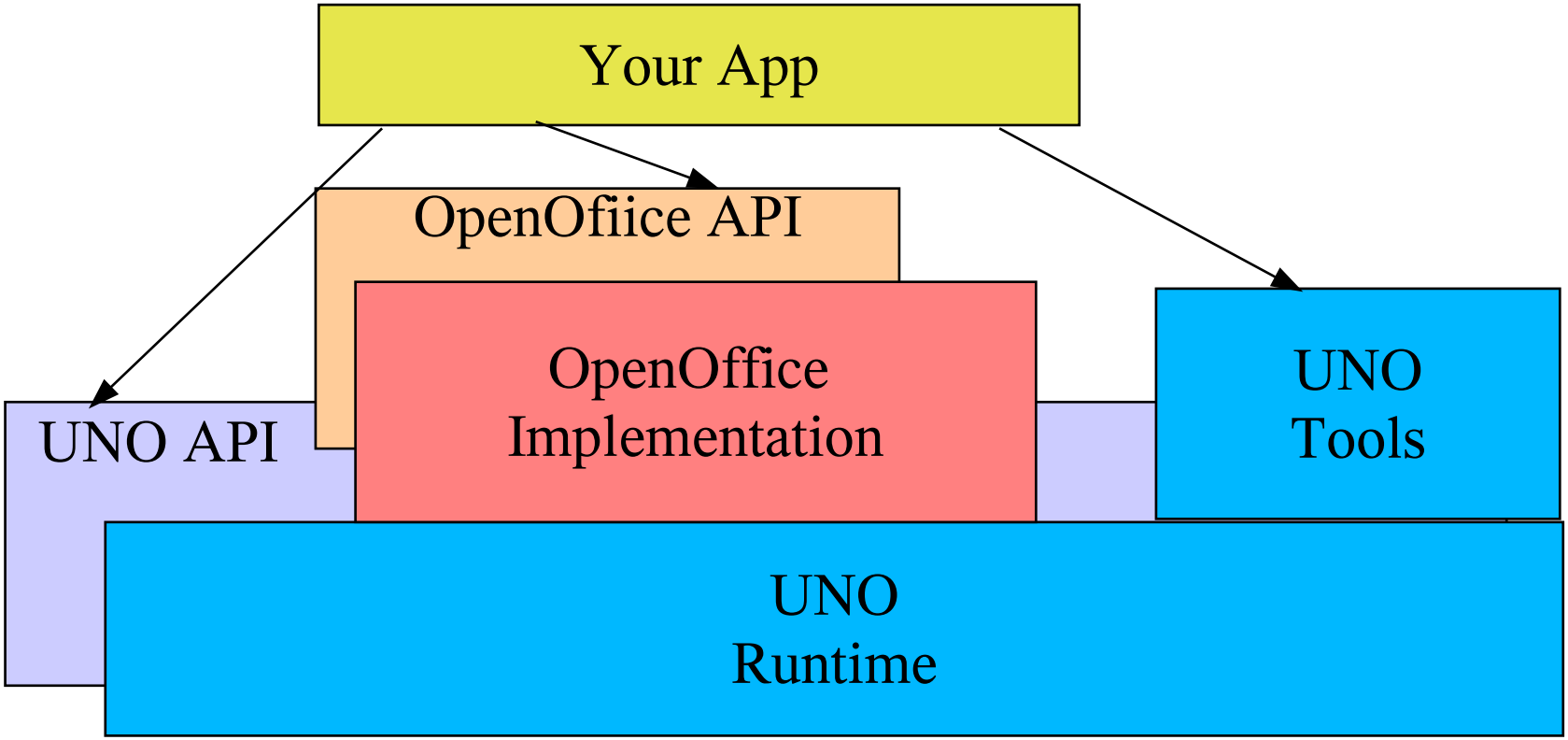
- UNO Language Binding
  - These runtime environments are provided as language bindings.
- OOo API
  - Most objects of OpenOffice.org are able to communicate in a UNO environment.
  - The specification for the programmable features of OpenOffice.org is called the OpenOffice.org API.

UNO Language Binding

- Currently there are complete UNO language bindings for:
  - C
  - C++ (compiler dependent, please see http://porting.openoffice.org for a list of supported platforms)
  - Java
  - Python
- Additionally, there are bindings that allow to access UNO components, but do not support the development of new UNO components:
  - OpenOffice.org Basic
  - Object Linking and Embedding (OLE) Automation
  - Common Language Infrastructure (CLI). Note: The Microsoft .NET framework is an implementation of this standard.

UNO Development Model

API: Use Ooo


Your App
OpenOfiice API
OpenOffice
Implementation
UNO API
UNO
Tools
UNO
Runtime

API: Use Ooo

Areas of application

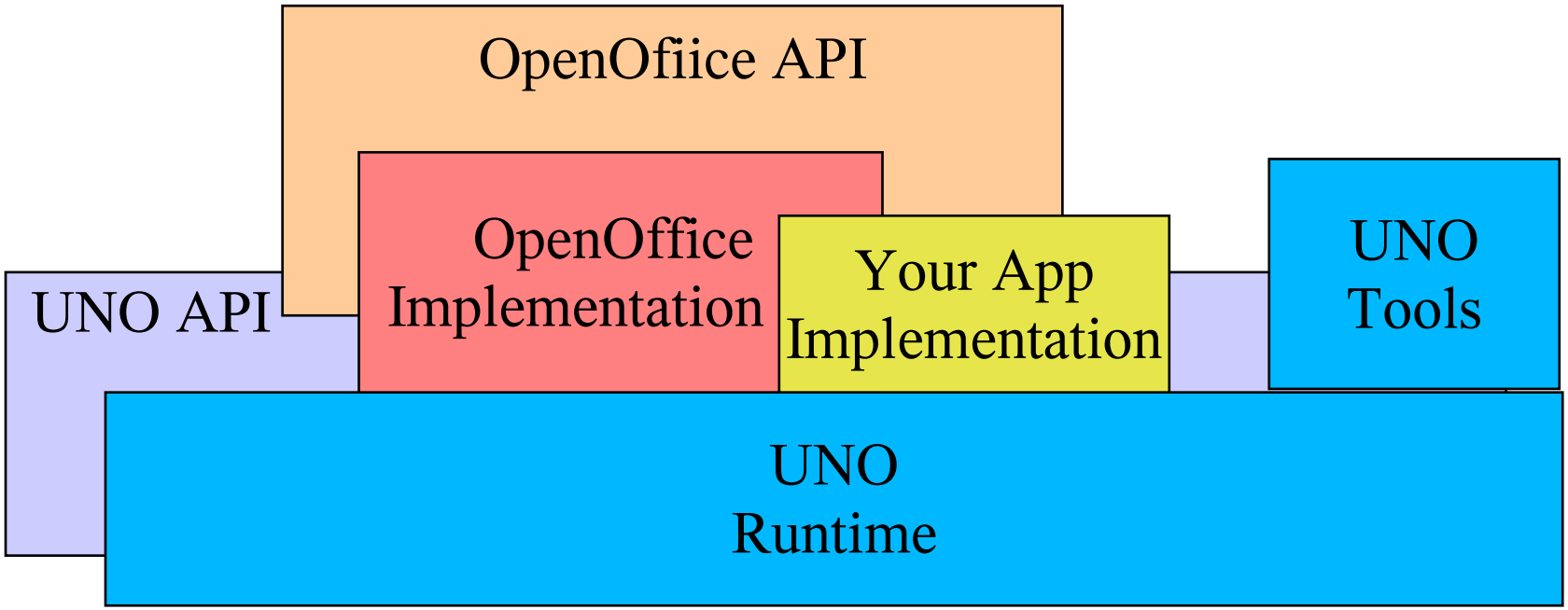
Languages:

- StarBasic, VB, Jscript, OLE Automation, Java, C++, Python

Scenarios:

- Generating and processing documents
- Extracting information out of documents
- Converting documents into different document formats(such as HTML)
- Embed OpenOffice Bean/ActiveX into 3rd party App

API: Extends OOo



API: Extends Ooo

Areas of application

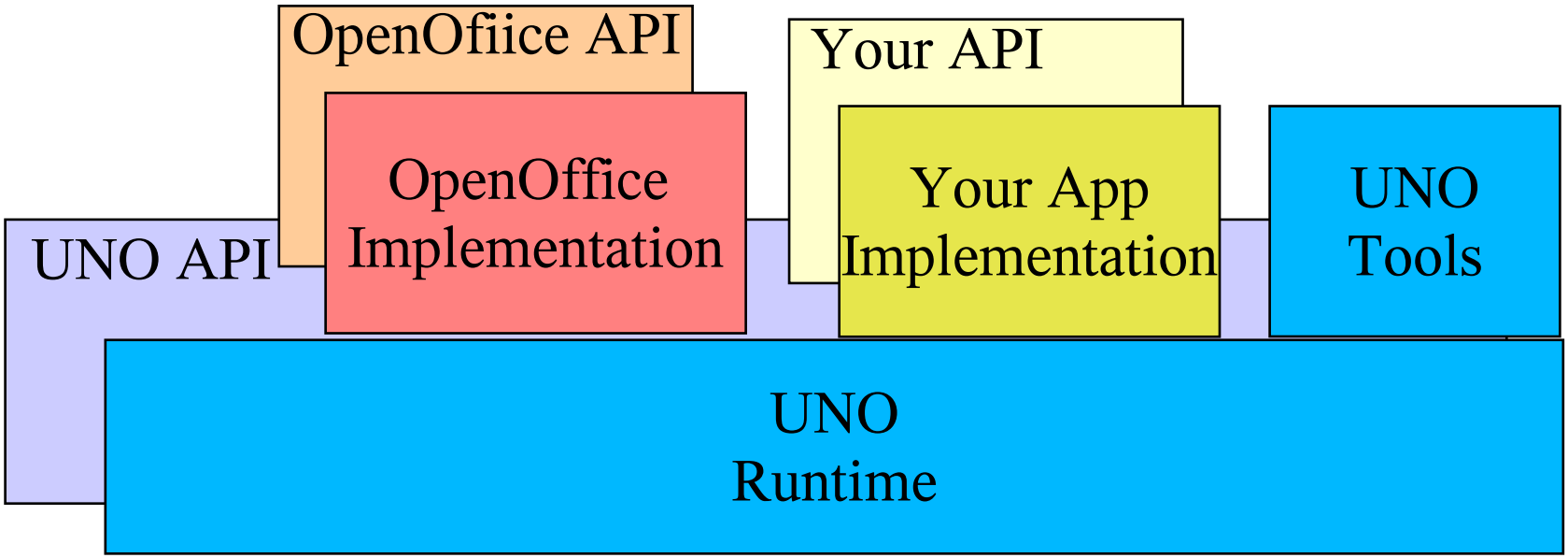
Languages:

- Java, C++

Components:

- Calc Add-in
- Chart Add-in
- Import/Export filters
- Linguistic extensions
- Database drivers

API: Do Everything



API: Do Everything

Areas of application

Languages:

- Java, C++

Adventagement:

- Cross platform
- Language extensible
- Scripted
- Component based ...

Do Evetything!

UNO
Interface Definition
and
Data Type Mapping

API Concept ...

- The OpenOffice.org API is a language independent approach to specify the functionality of OpenOffice.org.
- It is actually more like a specification than an API of a preexisting implementation.
- On the other hand we tried to be as similar to Java as possible.
- A long term target on the OpenOffice.org roadmap is to split the existing OpenOffice.org into small components which are combined to provide the complete OpenOffice.org functionality. Such components are manageable, they interact with each other to provide high level features and they are exchangeable with other implementations providing the same functionality, even if these new implementations are implemented in a different programming language.

UNO IDL

- UNO IDL is based on CORBA IDL, but additionally it supports
  - inheritance for exceptions and structures,
  - assigned values for enums,
  - a new stereotype "service"(combines interfaces and properties).
- And currently it does not support:
  - arrays as defined types
  - Unions

API
Simple Data Types

UNO	Type description	Java	C++	Basic
char	16-bit unicode character type	char	sal_Unicode	-
boolean	boolean type; true and false	boolean	sal_Bool	Boolean
byte	8-bit ordinal type	byte	sal_Int8	Integer
short	signed 16-bit ordinal type	short	sal_Int16	Integer
unsigned short	unsigned 16-bit ordinal type	-	sal_uInt16	-
long	signed 32-bit ordinal type	int	sal_Int32	Long

API
Data Types: String

- UNO considers strings to be simple types, but since they need special treatment in some environments:
  - In Java, use UNO strings as if they were native java.lang.String objects.
  - In C++, strings must be converted to UNO unicode strings by means of SAL conversion functions, usually the function createFromAscii() in the ::rtl::OUString class:
    - static OUString createFromAscii(const sal_Char * value) throw();
  - In Basic, Basic strings are mapped to UNO strings transparently.

UNO	Description	Java	C++	Basic
string	string of 16-bit unicode characters	java.lang String	::rtl: OUString	String

API
Data Types: Any

- The OpenOffice.org API frequently uses an any type, which is the counterpart of the Variant type known from other environments.
- The any type holds one arbitrary UNO type. The any type is especially used in generic UNO interfaces.

The any type holds one arbitrary UNO type. The any type is especially used in generic UNO interfaces.

IDL type	Size [byte]	C++ type	Java type	Description
any	sizeof (uno_Any)	com::sun::star::uno::Any	java.lang.Object/ com.sun.star.uno.Any	universal type

UNO IDL: Interface

```
// module com::sun::star::uno  ; base interface for all UNO interfaces
interface XInterface
{
    any queryInterface( [in] type aType );
    [oneway] void acquire();
    [oneway] void release();
};
//The [oneway]  flag indicates that an operation will be executed asynchronously.
```

Interface Mapping

IDL type	Size [byte]	C++ type	Java type	Description
Interface	4	com::sun::star:: uno::Reference< interfacetype >	Java interface with the same name	Pointer to a refcounted interface

In Java

If an IDL interface inherits another interface, the Java interface extends the appropriate Java interface.

UNO IDL: Service

- UNO uses services to specify complete objects which can have many aspects.
  - A service describes an object by combining interfaces and properties into an abstract object specification.
  - Do not get confused by the meanings the word service has in other contexts. In UNO, a service is precisely this: a composition of interfaces and properties.
- An interface
  - Is a set of methods that together define one single aspect of a service.

UNO IDL: Service

A property

- is a feature of a service which is not considered an integral or structural part of the service and therefore is handled through generic getPropertyValue()/setPropertyValue() methods instead of specialized get methods, such as getPrinter().
- An object containing properties only has to support the com.sun.star.beans.XPropertySet interface to be prepared to handle all kinds of properties.
  - Typical examples are properties for character or paragraph formatting. With properties, you can set multiple features of an object through a single call to setPropertyValues(), which greatly improves the remote performance.
  - For instance, paragraphs support the setPropertyValues() method through their com.sun.star.beans.XMultiPropertySet interface.

Why Service?

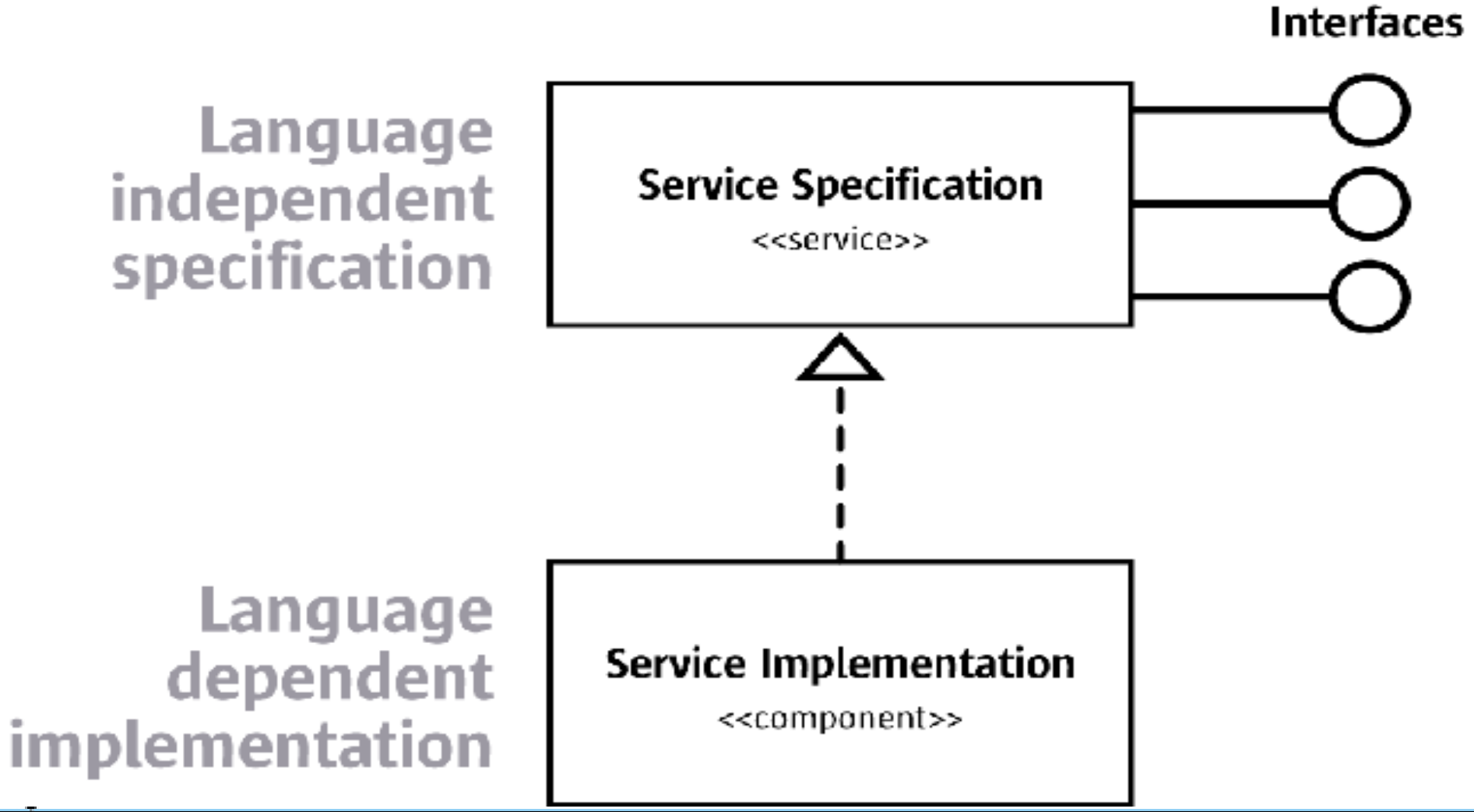
- Services separate specification from implementation
- Services make fine-grained interfaces manageable
  - Abstract interfaces are more reusable, if they are fine-grained, i.e. if they are small and describe only one aspect of an object, not several aspects.
  - But then you need many of them to describe a useful object. Services allow to have fine-grained interfaces on the one hand and to manage them easily by forging them into a service.
  - Since it is quite probable that objects in an office environment will share many aspects, this fine granularity allows the interfaces to be reused and thus to get objects that behave consistently.

OpenOffice.org
The Open Source Office Suite

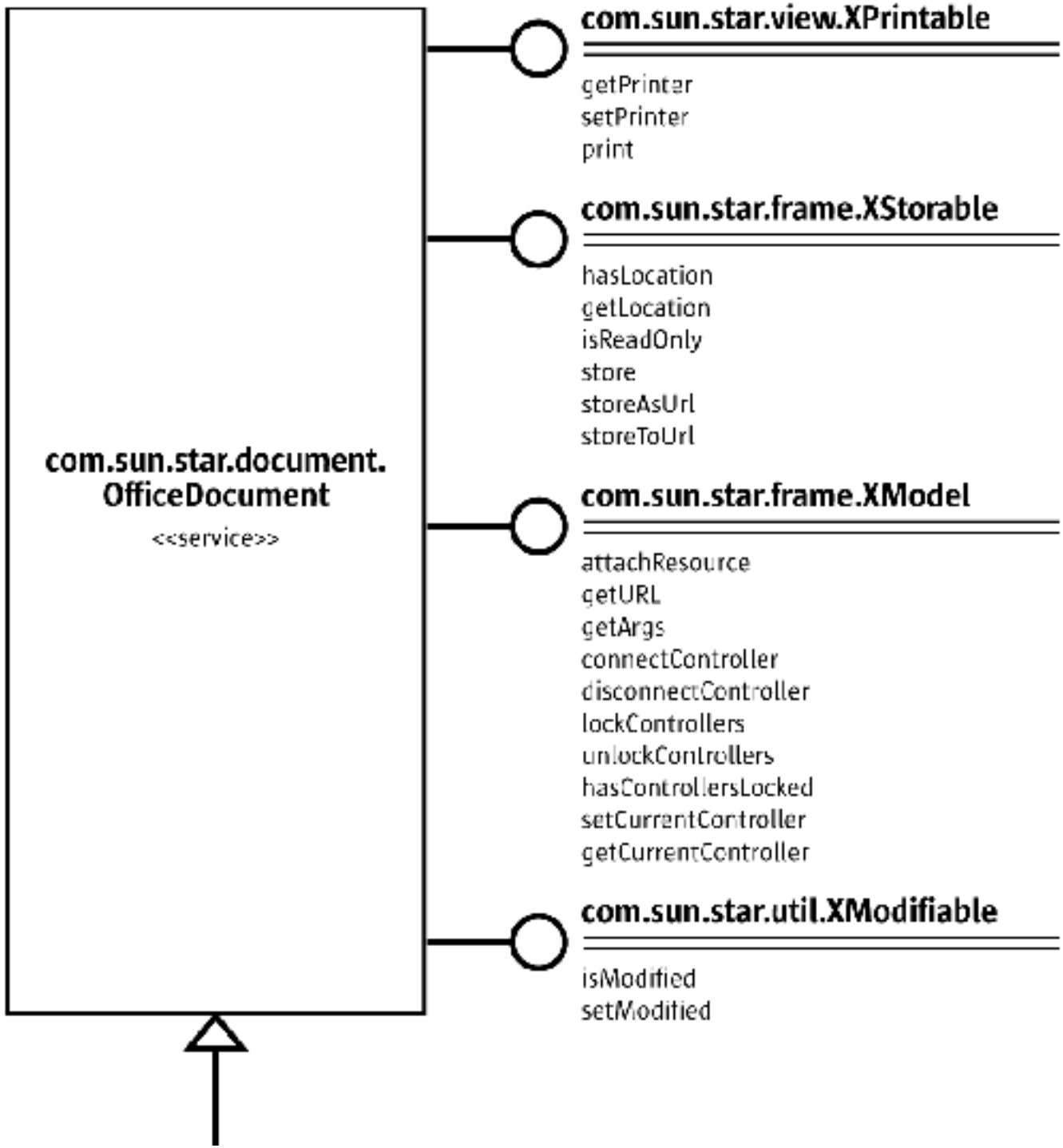
Why Service?

- Service names allow to create instances by specification name, not by class names
- Services handle a large number of non-structural properties
  - If you have only interfaces to specify objects, you need many get and set methods to handle all the qualities of office documents.
  - Moreover, once you define them, they become a hard part of the structure of an object, which makes it difficult to reuse the specification elsewhere.
  - With properties, a multitude of qualities can be specified that are no structural parts of the objects, and instead of calling many get and set methods, properties in a UNO service can be manipulated at once by a single method call, if necessary.

Interface/Service/component

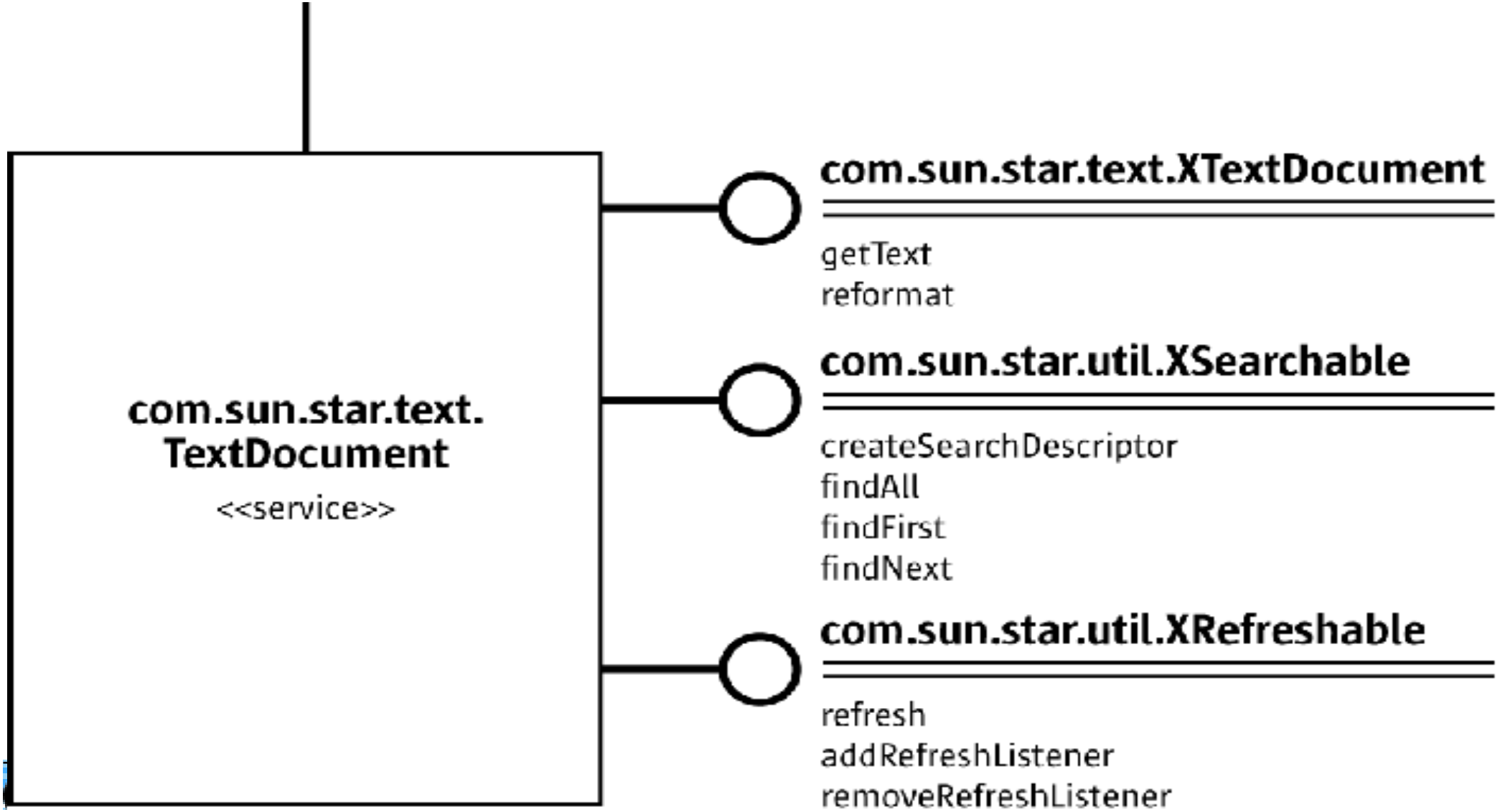


Service Exam ...



OpenOffice.org
The Open Source Office Suite

Service Exam



UNO IDL: Referencing Interfaces

```
// com.sun.star.text.TextDocument
service TextDocument
{
    ...
    interface com::sun::star::text::XTextDocument;
    interface com::sun::star::util::XSearchable;
    interface com::sun::star::util::XRefreshable;
    [optional] interface com::sun::star::text::XFootnotesSupplier;
    [optional] interface com::sun::star::text::XEndnotesSupplier;
    ...
};
```

UNO IDL: Including Properties

```
// com.sun.star.text.TextContent
service TextContent
{
    interface com::sun::star::text::XTextContent;
    [optional, property] com::sun::star::text::TextContentAnchorType
AnchorType;
    [optional, readonly, property] sequence<com::sun::star::text::
TextContentAnchorType> AnchorTypes;
    [optional, property] com::sun::star::text::WrapTextMode
TextWrap;
};
```

UNO IDL: property flags

- Optional
- Readonly
- Removable
  - The property is removable, this is used for dynamic properties.
- Transient
  - The property will not be stored if the object is serialized
- Bound
  - Changes of property values are broadcast to com.sun.star.beans.XPropertyChangeListeners
- Constrained
  - The property broadcasts an event before its value changes. Listeners have the right to veto the change.

UNO IDL: property flags

- Maybeambiguous
  - Possibly the property value cannot be determined in some cases, for example, in multiple selections with different values.
- Maybedefault
  - The value might be stored in a style sheet or in the environment instead of the object itself.
- Maybevoid
  - In addition to the range of the property type, the value can be void. It is similar to a null value in databases.

UNO IDL: Referencing other Services

```
// com.sun.star.text.Paragraph
service Paragraph
{
    service com::sun::star::text::TextContent;
    [optional] service com::sun::star::style::ParagraphProperties;
    [optional] service com::sun::star::style::CharacterProperties;
     ...
};
```

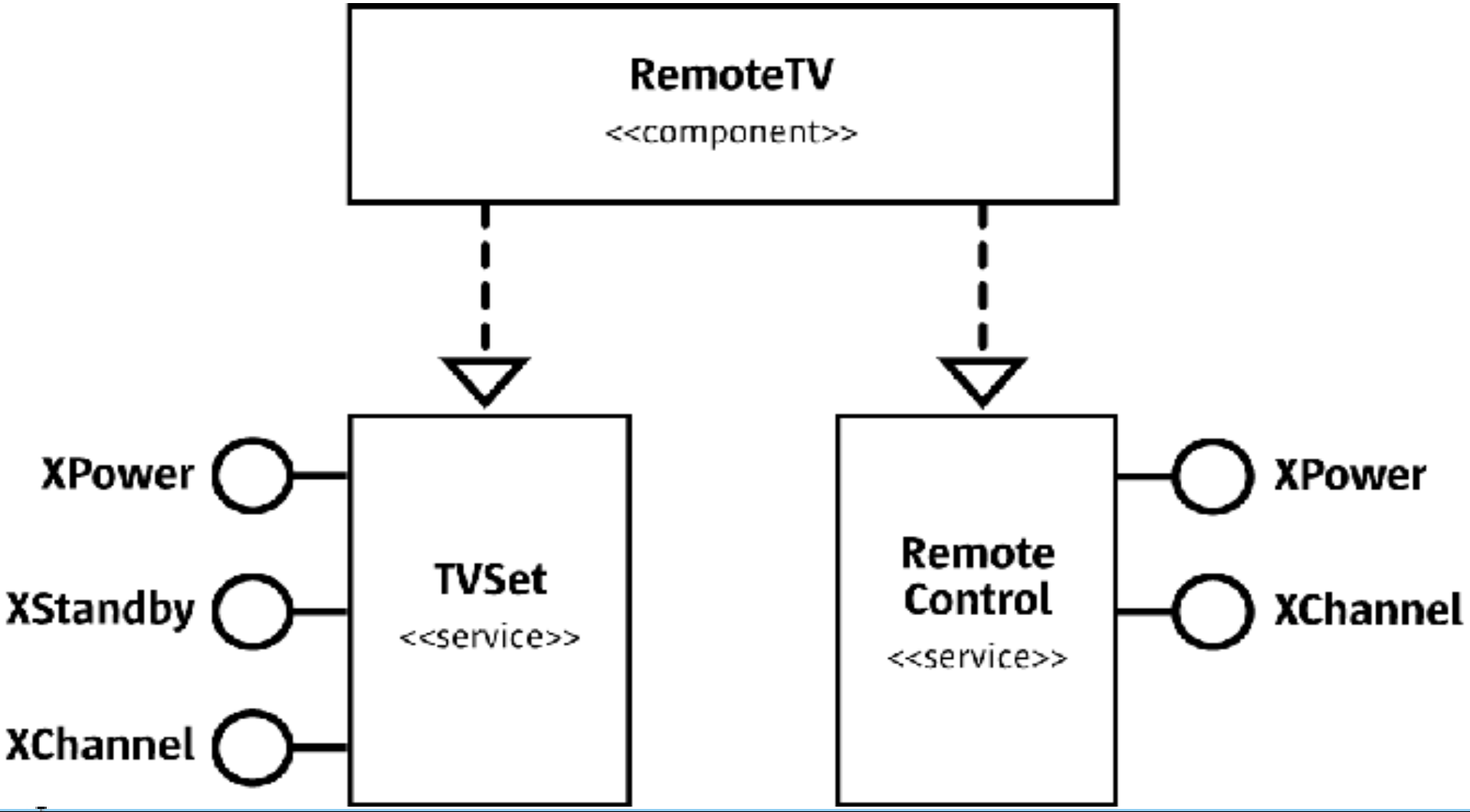
- That a service is included by another service has nothing to do with implementation inheritance, only the specifications are combined. It is up to the implementer if he inherits or delegates the necessary functionality, or if he implements it from scratch.

OpenOffice.org
The Open Source Office Suite

Service Implementations in Components

- A component is a shared library or Java archive containing implementations of one or more services in one of the target programming languages supported by UNO.
  - All specified interfaces and properties must be implemented.
- Components must be registered in the UNO runtime system.
  - After the registration all implemented services can be used by ordering an instance of the service at the appropriate service factory and accessing the functionality over interfaces.

Component Example



UNO IDL: Structs

```
// com.sun.star.lang.EventObject
struct EventObject
{
        com::sun::star::uno::XInterface Source;
};
// com.sun.star.beans.PropertyChangeEvent
struct PropertyChangeEvent : com::sun::star::lang::EventObject {
    string PropertyName;
    boolean Further;
    long PropertyHandle;
    any OldValue;
    any NewValue;
};
```

UNO IDL: Predefined Values

- Const

```
const short ID = 23;
const boolean ERROR =
  true;
```

- Constants
  - defines a named group of const values.

```
constants ImageAlign {
    const short LEFT = 0;
    const short TOP = 1;
    const short RIGHT = 2;
    const short BOTTOM =
  3;
};
```

UNO IDL: Predefined Values

```
// com.sun.star.uno.TypeClass
enum TypeClass {
    VOID,
    CHAR,
    BOOLEAN,
    SHORT,
    ...
};
```

```
enum Error {
    SYSTEM = 10, // value 10
    RUNTIME,     // value 11
    FATAL,       // value 12
    USER = 30,   // value 30
    SOFT         // value 31
};
```

UNO IDL: sequence

- A sequence type is a set of elements of the same type, that has a variable number of elements.

```
sequence< com::sun::star::uno::XInterface >

sequence< string > getNamesOfIndex( sequence< long > indexes );
```

- In C++ an IDL sequence is mapped to:
  - template< class t > com::sun::star::uno::Sequence< t >
  - e.g.: Sequence< sal_Int32 > seqInt(3);

In Java an IDL sequence is mapped to::

- An IDL sequence<long> is mapped to int[ ]
- An IDL sequence< sequence <long> > is mapped to int[ ][ ]

UNO IDL: module

- Modules are namespaces, similar to namespaces in C++ or packages in Java.
- They group services, interfaces, structs, exceptions, enums, typedefs, constant groups and submodules with related functional content or behavior.

```
Module com {
    module sun {
        module star {
            module uno {
                interface XInterface { ...};};};};};
```

UNO IDL: exception

```
// com.sun.star.uno.Exception is the base exception for all exceptions
exception Exception {
    string Message;
    Xinterface Context;
};
// com.sun.star.uno.RuntimeException is the base exception for serious
problems occuring at runtime, usually programming errors or problems in
the runtime environment
exception RuntimeException : com::sun::star::uno::Exception {
};
```

UNO IDL: singleton

- Singletons are used to specify named objects where exactly one instance can exist in the life of a UNO component context.
  - A singleton references one service and specifies that the only existing instance of this service can be reached over the component context using the name of the singleton.
  - If no instance of the service exists, the component context will instantiate a new one.

```
singleton theServiceManager {

    service com::sun::star::lang::ServiceManager

};
```

IDL comparison

feature	UNOIDL	CORBA-IDL	MIDL
multiple inheritance of interfaces	no	yes	no
exceptions	yes	yes	no
exceptions with inheritance	yes	no	no
mandatory superinterface	yes	no	yes
C-style arrays	no	yes	?
Life cycle concept	yes (refcounting)	no	yes (refcounting)

UNO IDL: tools

- Idlc: .idl→.urd
  - idlc -C -I../idl XimageShrink.idl
- Regmerge: .urd→.rdb
  - regmerge thumbs.rdb /UCR XimageShrink.urd
- ***maker: .rdb→.h or .java
  - Javamaker
    - javamaker -BUCR -Torg.openoffice.test.XImageShrink -nD <OFFICE_PROGRAM_PATH>/applicat.rdb thumbs.rdb
  - Cppumaker
    - cppumaker -BUCR -Torg.openoffice.test.XImageShrink <OFFICE_PROGRAM_PATH>/applicat.rdb thumbs.rdb

Registry: tools

- Rdbmaker
  - Create a new registry by extracting given types (including dependent types) from another registry.
- Regcomp
- Regmerge
- Regview
  - regview thumbs.rdb
- Xml2cmp

UNO
InterProcess and Remote Mechanism

UNO Interprocess Connections

- UNO objects in different environments connect via the interprocess bridge.
  - by converting the method name and the arguments into a byte stream representation, and sending this package to the remote process, for example, through a socket connection.
- Starting OpenOffice.org in Listening Mode
  - soffice -accept=socket,host=0,port=2002;urp;
- Importing a UNO Object
  - The correct way to do this is using the com.sun.star.bridge.UnoUrlResolver service.

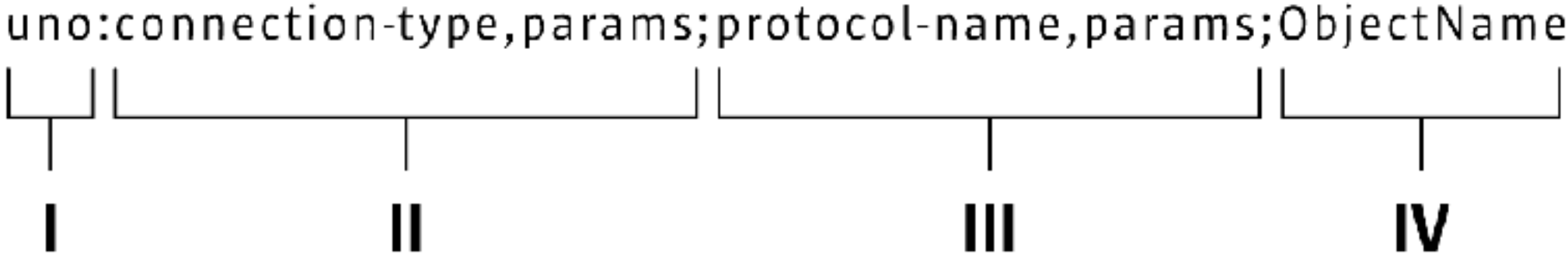
XunoUrlResolver

```
interface XUnoUrlResolver: com::sun::star::uno::XInterface
{
   /** resolves an object on the UNO URL */
   com::sun::star::uno::XInterface resolve( [in] string sUnoUrl )
      raises (com::sun::star::connection::NoConnectException,
            com::sun::star::connection::ConnectionSetupException,
            com::sun::star::lang::IllegalArgumentException);
};
```

- The string passed to the resolve() is called a UNO URL

UNO URL

UNO-Url



Uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager

URL schema uno

This identifies the URL as UNO URL and distinguishes it from others, such as http: or ftp: URLs.

Connection Type Socket

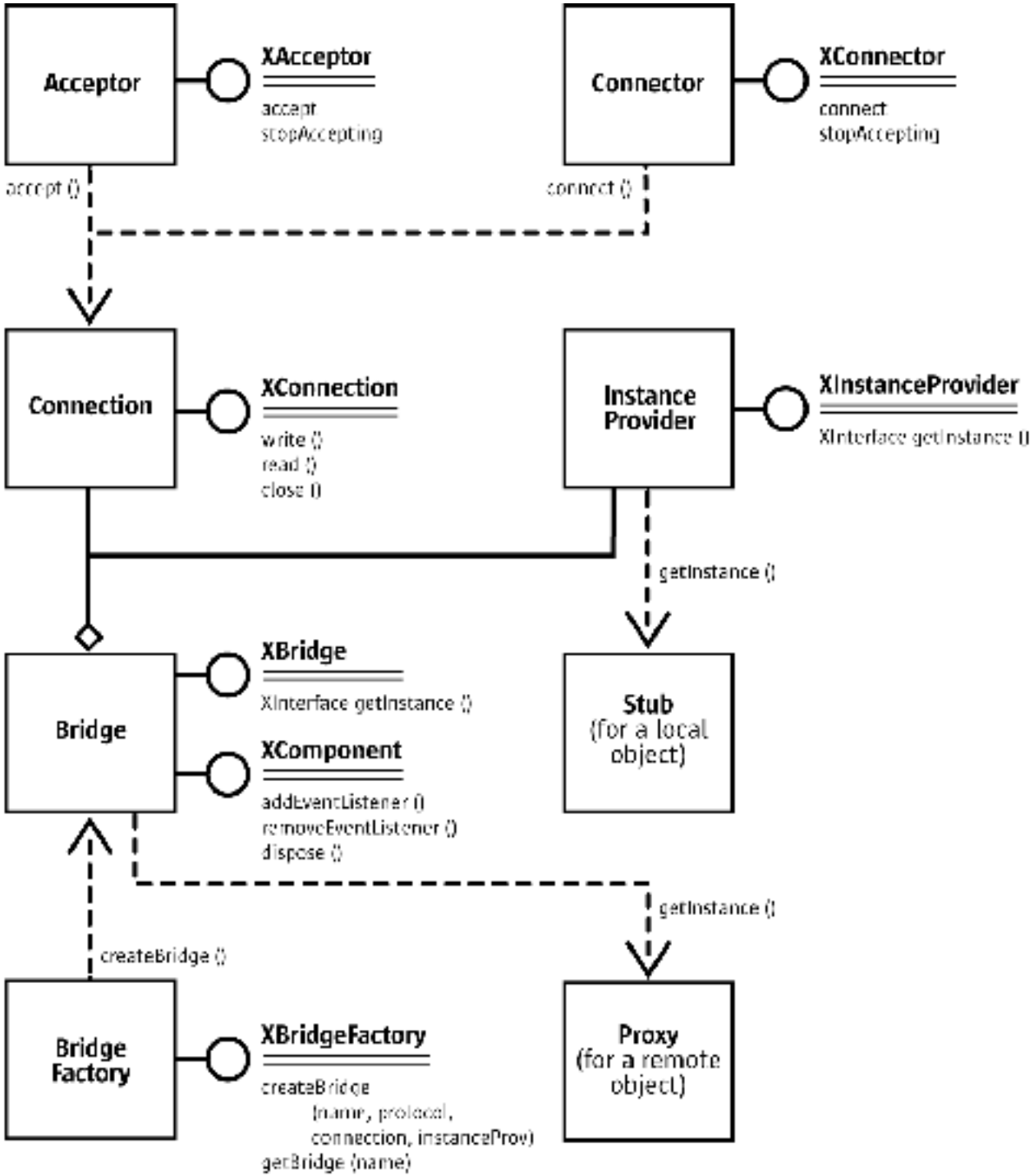
- Reliable TCP/IP socket connection
- Param:
  - Host
    - Hostname or IP number
    - In an acceptor string, this may be 0 ('host=0'), which means, that it accepts on all available network interfaces.
  - Port
  - TcpNoDelay
    - Corresponds to the socket option tcpNoDelay. For a UNO connection, this parameter should be set to 1 (this is NOT the default — it must be added explicitly). If the default is used (0), it may come to 200 ms delays at certain call combinations.

Connection Type Pipe

- A named pipe (uses shared memory).
  - This type of interprocess connection is marginally faster than socket connections
  - works only if both processes are located on the same machine.
  - It does not work on Java by default, because Java does not support named pipes directly
- Param:
  - name
    - Name of the named pipe.
    - Can only accept one process on name on one machine at a time.

Connection Type: add more

- You can add more kinds of interprocess connections
  - by implementing connector and acceptor services
  - and choosing the service name by the scheme com.sun.star.connection.Connector.<connection-type>, where <connection-type> is the name of the new connection type.
- If you implemented the service com.sun.star.connection.Connector.mytype
- use the UnoUrlResolver with the URL 'uno:mytype, param1=foo;urp;StarOffice.ServiceManager' to establish the interprocess connection to the office.



InterProcess

Key Service:
Connector
acceptor

UNO
Instantiation

Root Object

- service manager is the root object for connections to OpenOffice.org (and to any UNO application)
  - The root object serves as the entry point for every UNO application and is passed to every UNO component during instantiation.

Two different concepts to get the root object:

- Previous concept: service manager
- Newer concept: component context

Service Manager

- The com.sun.star.lang.ServiceManager is the main factory in every UNO application.
- It instantiates services by their service name,
  - to enumerate all implementations of a certain service,
  - and to add or remove factories for a certain service at runtime.
- The service manager is passed to every UNO component during instantiation.

Interfaces in Service Manager

- com.sun.star.lang.XMultiServiceFactory interface

```
interface XMultiServiceFactory: com::sun::star::uno::XInterface
{
    com::sun::star::uno::XInterface createInstance( [in] string aServiceSpecifier )
        raises( com::sun::star::uno::Exception );
    com::sun::star::uno::XInterface createInstanceWithArguments(
            [in] string ServiceSpecifier,
            [in] sequence<any> Arguments )
        raises( com::sun::star::uno::Exception );
    sequence<string> getAvailableServiceNames();
};
```

CreateInstance()

- createInstance() returns a default constructed service instance.
  - The returned service is guaranteed to support at least all interfaces, which were specified for the requested servicename.
  - The returned XInterface reference can now be queried for the interfaces specified at the service description.
- In case the service manager does not provide an implementation for a request, a null reference is returned, so it is mandatory to check.

CreateInstance()

- When using the service name, the caller does not have any influence on which concrete implementation is instantiated.
  - If multiple implementations for a service exist, the service manager is free to decide which one to employ.
  - This in general does not make a difference to the caller because every implementation does fulfill the service contract.
  - Performance or other details may make a difference.
- So it is also possible to pass the implementation name instead of the service name, but it is not advised to do so as the implementation name may change.

Component Context

- Often a component needs more functionality or information that must be exchangeable after deployment of an application.
  - In this context, the service manager approach is limited.
- It is basically a read-only container offering named values. One of the named values is the service manager.
- The component context is passed to a component during its instantiation.
- This can be understood as an environment where components live (the relationship is similar to shell environment variables and an executable program).

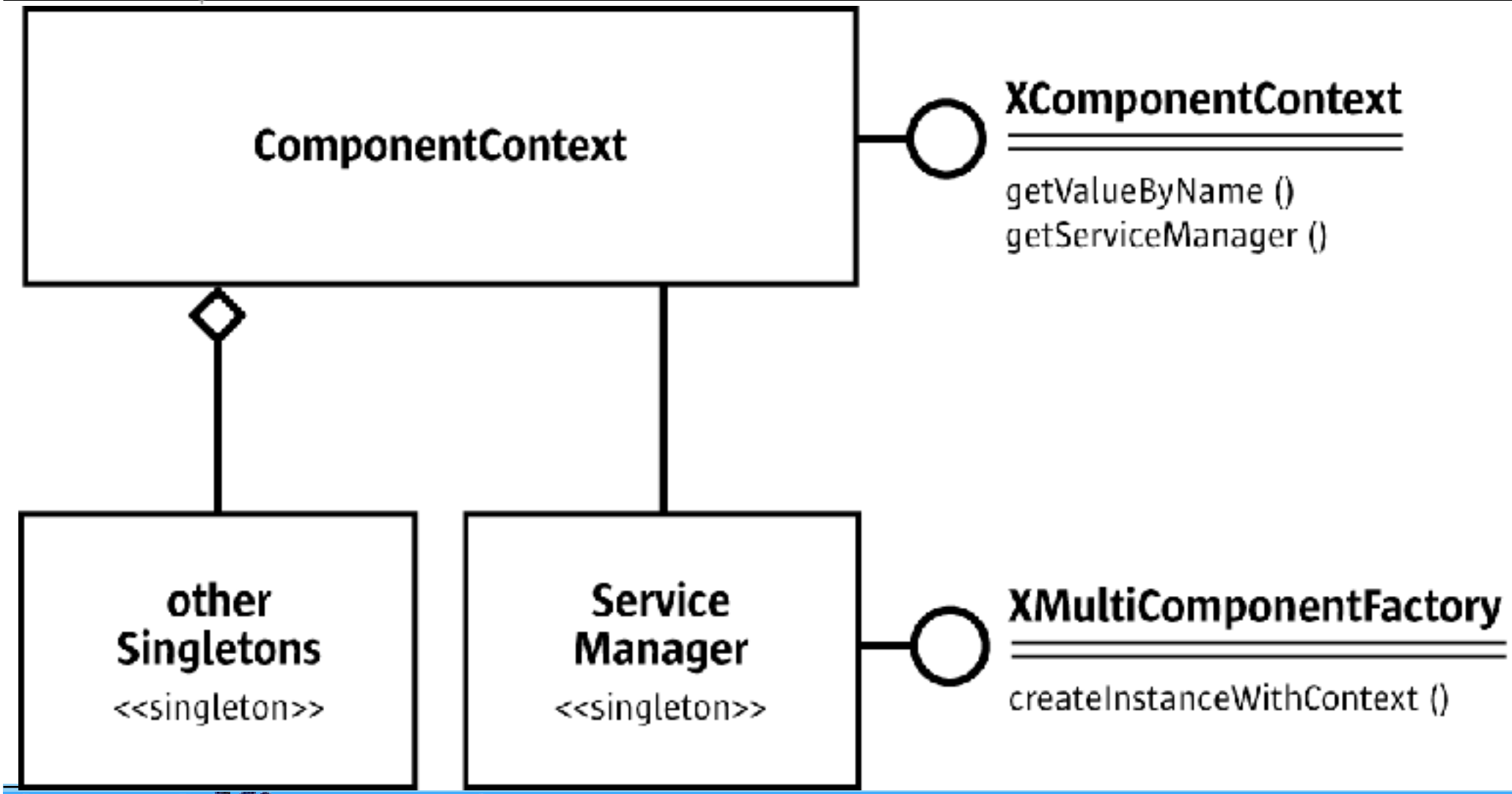
Interfaces in Component Context

- com.sun.star.uno.XComponentContext interface

```
// module com::sun::star::uno
interface XComponentContext : XInterface
{    any getValueByName( [in] string Name );
   com::sun::star::lang::XMultiComponentFactory getServiceManager();};
```

- It offers at least three kinds of named values:
  - Singletons (/singletons/...)
  - Implementation properties (not yet defined)
  - Service properties (not yet defined)
    - Note that service context properties are different from service properties.

Component Context



Using UNO Interfaces

```
// module com::sun::star::uno

interface XInterface

{    any queryInterface( [in] type aType );

    [oneway] void acquire();

    [oneway] void release();   };
```

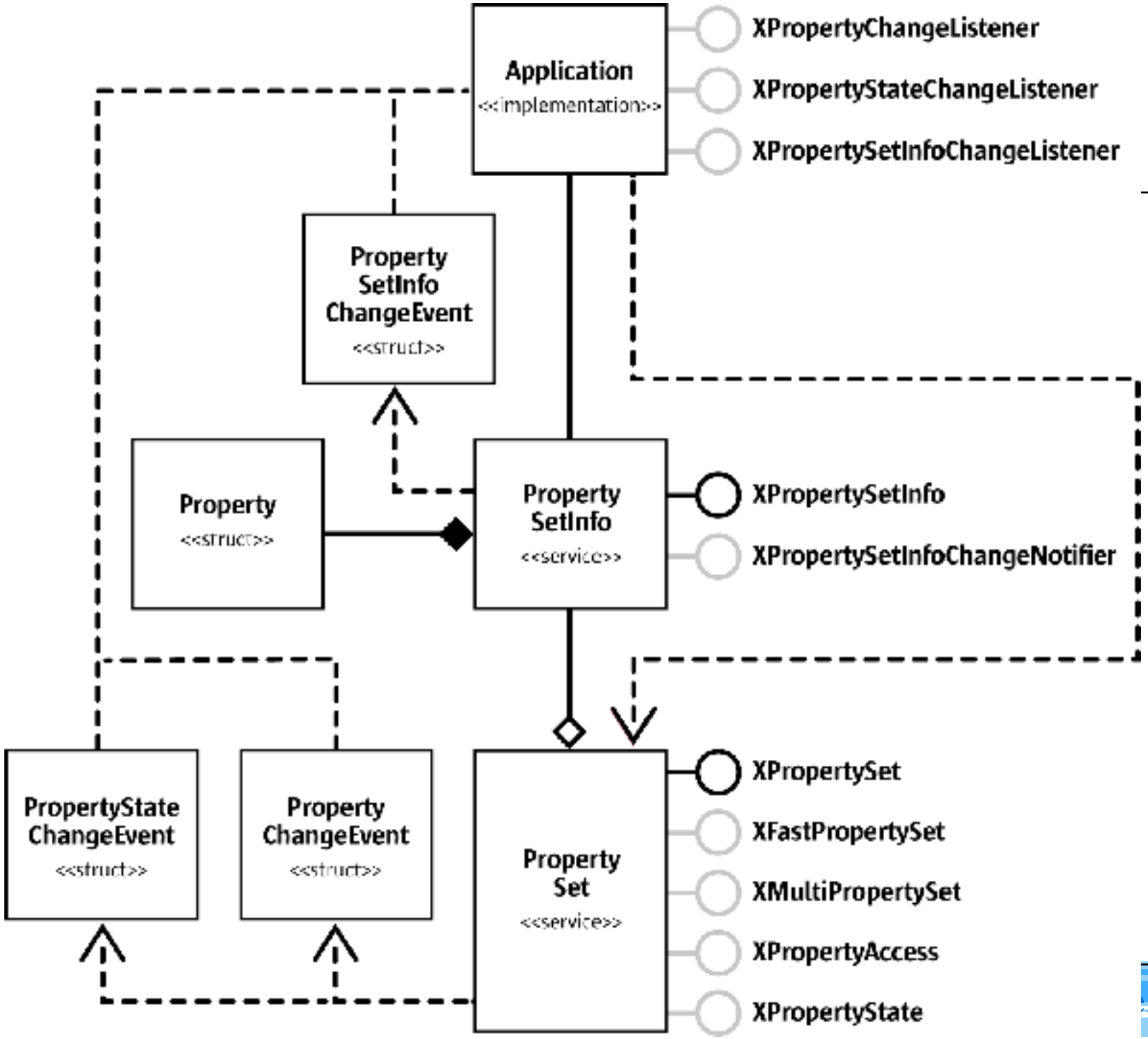
- The type parameter is an UNO IDL base type, and generally stores the name of a type and its com.sun.star.uno.TypeClass.
- The call may return with an interface reference of the requested type or with a void any.
- In C++ or Java simply test if the result is not equal null.

Using UNO Interfaces

```
XComponentContext xLocalContext = com.sun.star.comp.
helper.Bootstrap.createInitialComponentContext(null);

// initial serviceManager

XMultiComponentFactory xLocalServiceManager =
xLocalContext.getServiceManager();

// create a urlresolver

Object urlResolver  = xLocalServiceManager.
createInstanceWithContext("com.sun.star.bridge.
UnoUrlResolver", xLocalContext);
```

Using UNO Interfaces

```
XunoUrlResolver xUrlResolver = (XUnoUrlResolver)
    UnoRuntime.queryInterface(UnoUrlResolver.class, urlResolver);
if (null == xUrlResolver) {
    throw new java.lang.Exception(
        "Error: UrlResolver service does not export XUnoUrlResolver
interface");
}
Object remoteObject = xUrlResolver.resolve(
    "uno:socket,host=0,port=2002;urp;StarOffice.ServiceManager");
```



Using UNO Properties

UNO
Lifetime Control

Lifetime of UNO Objects

- UNO uses the same mechanism as Microsoft COM by handling the lifetime of objects by reference counting.
- While in C++ UNO, each object maintains its own reference count.
- Once acquire() is called on the UNO object, there is a reference or a hard reference to the object, as opposed to a weak reference.
- Calling release() on the object is often called releasing or clearing the reference.

Lifetime of UNO Objects in Java

- Java UNO uses the normal Java garbage collector mechanism to handling the lifetime of objects.
- The UNO Java binding encapsulates acquire() and release() in the UnoRuntime.queryInterface() call.
- The same applies to the Reference<> template in C++.
- As long as the interface references are obtained through these mechanisms, acquire() and release() do not have to be called in your programs.

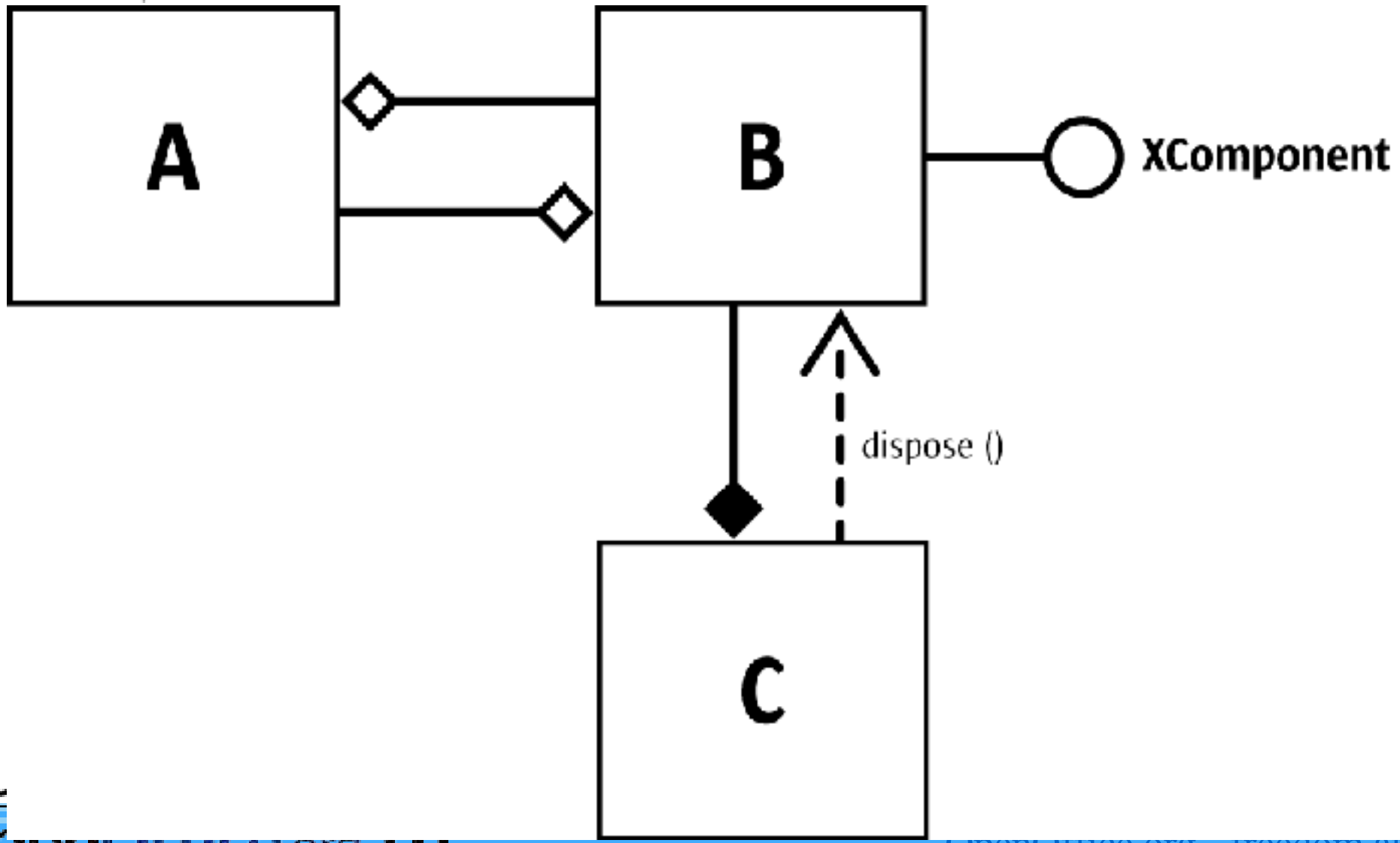
cyclic references

- A central problem of reference counting systems is cyclic references.
  - Assume Object A keeps a reference on object B and B keeps a direct or indirect reference on object A.
  - Even if all the external references to A and B are released, the objects are not destroyed, which results in a resource leak.
- In UNO, the developer must explicitly decide when to the break cyclic references.
- To support this concept, the interface com.sun.star.lang.XComponent exists.
  - When an Xcomponent is disposed of, it can inform other objects that have expressed interest to be notified.

Xcomponent

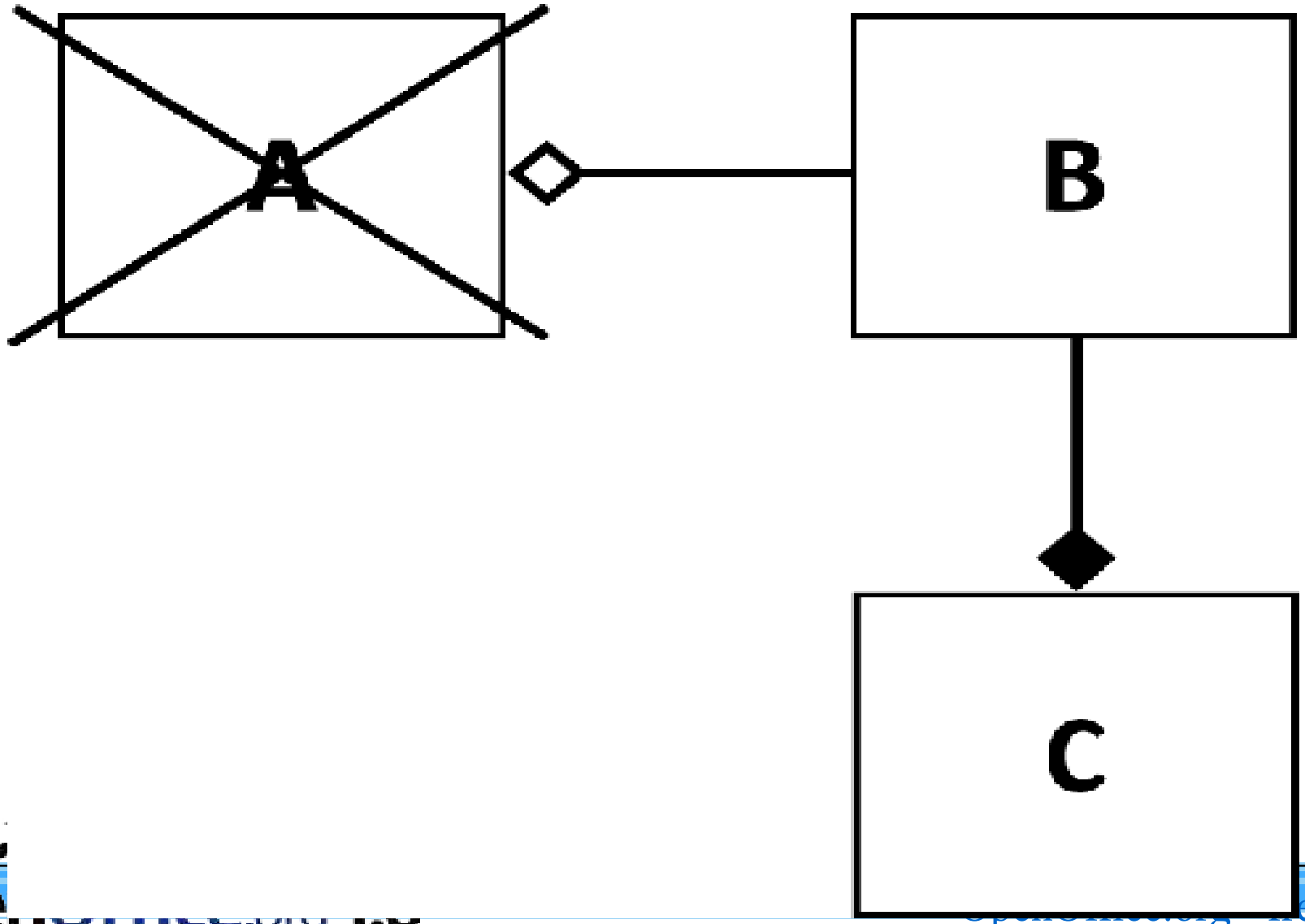
```
// within the module com::sun::star::lang; when dispose() is called, previously
added XEventListeners are notified
interface XComponent: com::sun::star::uno::XInterface
{     void dispose();
      void addEventListener( [in] XEventListener xListener );
      void removeEventListener( [in] XEventListener aListener );   };
// An XEventListener is notified by calling its disposing() method
interface XEventListener: com::sun::star::uno::XInterface
{   void disposing( [in] com::sun::star::lang::EventObject Source );  };
```

Xcomponent



OpenOffice.org
The Open Source Office Suite

Xcomponent



Weak Objects and References

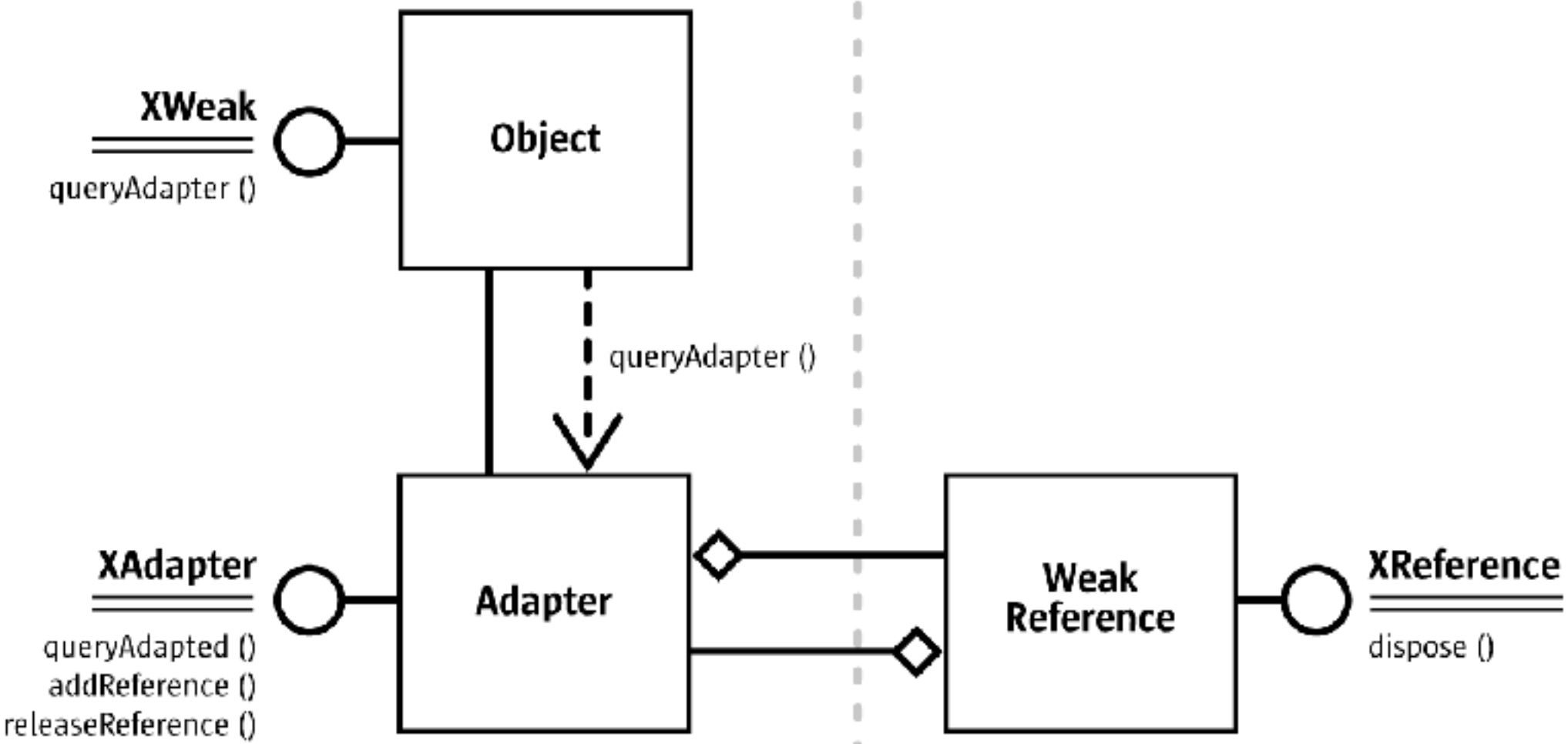
- A strategy to avoid cyclic references is to use weak references.
- Having a weak reference to an object means that you can reestablish a hard reference to the object again if the object still exists, and there is another hard reference to it.
  - object B could be specified to hold a hard reference on object A, but object A only keeps a weak reference to B. If object A needs to invoke a method on B, it temporarily tries to make the reference hard. If this succeeds, it invokes the method and releases the hard reference afterwards.

Weak Objects and References

- The sense of weak references is to hold a reference to an object without affecting the lifetime of the object.
  - That means that a weak reference may become invalid, at any time, if the referenced object dies.
- When a hard reference is established from the weak reference, it calls the queryAdapted() method at the com.sun.star.uno.XAdapter interface of the adapter object.
  - When the original object is still alive, it gets a reference for it, otherwise a null reference is returned.
- The adapter notifies the destruction of the original object to all weak references which breaks the cyclic reference between the adapter and weak reference.

OpenOffice.org
The Open Source Office Suite

Weak Objects and References



Weak demo in C++

```
// forward declaration of a function that
Reference< XFoo > getFoo();
int main()
{
    // default construct a weak reference.  This reference is empty
     WeakReference < XFoo > weakFoo;
```

Weak demo in C++

```
{
    // obtain a hard reference to an XFoo object
    Reference< XFoo > hardFoo = getFoo();
    assert( hardFoo.is() );
    // assign the hard reference to weak referencecount
    weakFoo = hardFoo;
  // the hardFoo reference goes out of scope. The object itself
  // is now destroyed, if no one else keeps a reference to it.
  // Nothing happens, if someone else still keeps a reference to it
}
```

Weak demo in C++

```
// now make the reference hard again
Reference< XFoo > hardFoo2 = weakFoo;
// check, if this was successful
if( hardFoo2.is() )
{
    // the object is still alive, you can invoke calls on it again
    hardFoo2->foo();
}   else
{       // the objects has died, you can't do anything with it anymore.  }
}
```

UNO
Object Identity

Object Identity

- UNO guarantees if two object references are identical, that a check is performed and it always leads to a correct result, whether it be true or false.
  - This is different from CORBA, where a return of false does not necessarily mean that the objects are different.
- In Java UNO, there is a static areSame() function at the com.sun.star.uno.UnoRuntime class.
- In C++, the check is performed with the Reference<>:: operator == () function that queries both references for XInterface and compares the resulting XInterface pointers.
  - CORBA interfaces are not designed in this manner. They need an object ID, because object identity is not guaranteed.

UNO
StarBasic Binding

UNO Language Bindings

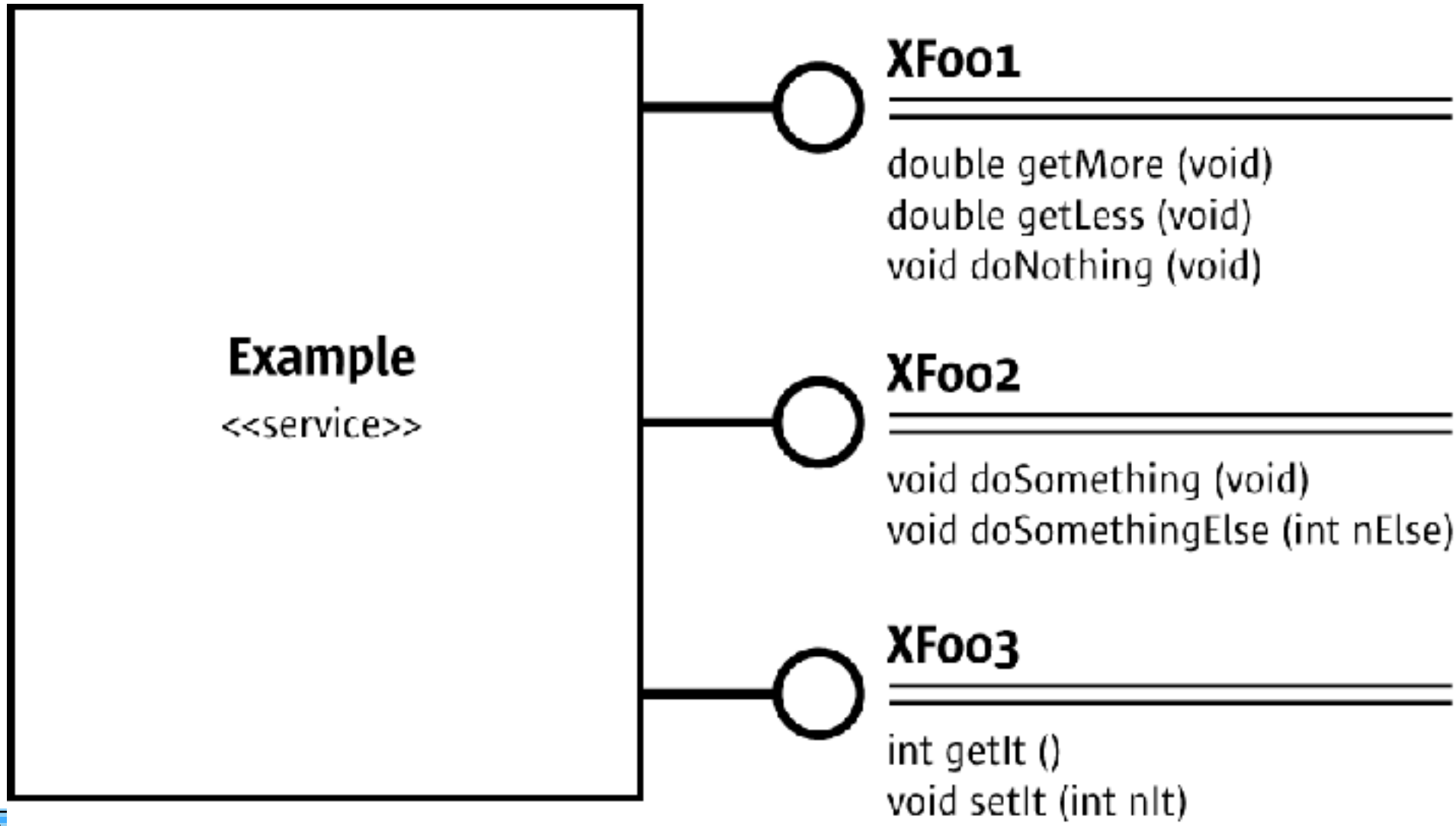
- sometimes called UNO Runtime Environment (URE)
- Language binding topics:
  - Mapping of all UNO types to the programming language types.
  - Mapping of the UNO exception handling to the programming language.
  - Mapping of the XInterface features (querying interfaces, object lifetime, object identity).
  - Bootstrapping of a service manager.

UNO
StarBasic Binding

- In Basic it is not necessary to distinguish between the different interfaces an object supports when calling a method.
  - In Java and C++, it is necessary to obtain a reference to each interface before calling one of its methods.
  - In Basic, every method of every supported interface can be called directly at the object without querying for the appropriate interface in advance.

```
' Basic
oExample = getExampleObjectFromSomewhere()
oExample.doNothing()              ' Calls method doNothing of XFoo1
oExample.doSomething()            ' Calls method doSomething of Xfoo2
oExample.doSomethingElse(42)      ' Calls method doSomethingElse of Xfoo2
```

UNO
StarBasic Binding



StarBasic: Instantiating UNO Services

- In Basic, instantiate services using the Basic Runtime Library (RTL) function createUnoService().

```
oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
```

- Instead of above, it is also possible to get the global UNO com.sun.star.lang.ServiceManager of the OpenOffice.org process by calling GetProcessServiceManager().

```
    oServiceMgr = GetProcessServiceManager()
    oSimpleFileAccess = oServiceMgr.createInstance( "com.sun.star.ucb.
SimpleFileAccess" )
    ' is the same as
    oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.
SimpleFileAccess" )
```

demo in StarBasic

```
Sub Main
  Dim mBritishWords(5) As String
  Dim mUSWords(5) As String
  Dim n As Long
  Dim oDocument As Object
  Dim oReplace As Object

  mBritishWords() = Array("colour", "neighbour", "centre", _
        "behaviour", "metre", "through")
  mUSWords() = Array("color", "neighbor", "center", _
        "behavior", "meter", "thru")
```

demo in StarBasic

```
  oDocument = ThisComponent
  oReplace = oDocument.createReplaceDescriptor
  For n = lbound(mBritishWords()) To ubound(mBritishWords())
    oReplace.SearchString = mBritishWords(n)
    oReplace.ReplaceString = mUSWords(n)
    oDocument.replaceAll(oReplace)
  Next n
End Sub
```

UNO
Java Binding

UNO Java Binding

Libs:

- jurt.jar,
- jut.jar,
- javaunohelper.jar,
- ridl.jar,
- classes.jar
- sandbox.jar .

The client obtains a reference to the global service manager of the office (the server) using a local com.sun.star.bridge.UnoUrlResolver.

The global service manager of the office is used to get objects from the other side of the bridge.

UNO in Java

Prototype of queryInterface

- java.lang.Object UnoRuntime.queryInterface(java.lang.Class targetInterface, Object sourceObject)
- java.lang.Object UnoRuntime.queryInterface(com.sun.star.uno.Type targetInterface, Object sourceObject)

UNO client in Java

```
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.beans.XPropertySet;
public class FirstConnection extends java.lang.Object {
    private XComponentContext xRemoteContext = null;
    private XmultiComponentFactory xRemoteServiceManager = null;
```

UNO client in Java

```
public static void main(String[] args) {
    FirstConnection firstConnection1 = new FirstConnection();
    try {
        firstConnection1.useConnection();
    }
    catch (java.lang.Exception e) {
        e.printStackTrace();
    }
    finally {
        System.exit(0);
    }
}
```

UNO client in Java

```
protected void useConnection() throws java.lang.Exception {
    try {
        xRemoteServiceManager = this.getRemoteServiceManager(
                "uno:socket,host=localhost,port=8100;urp;StarOffice.
ServiceManager");
        String available = (null != xRemoteServiceManager ?
"available" : "not available");
        System.out.println("remote ServiceManager is " + available);
         // do something with the service manager...
    }
    catch (com.sun.star.connection.NoConnectException e) {...}
}
```

UNO client in Java

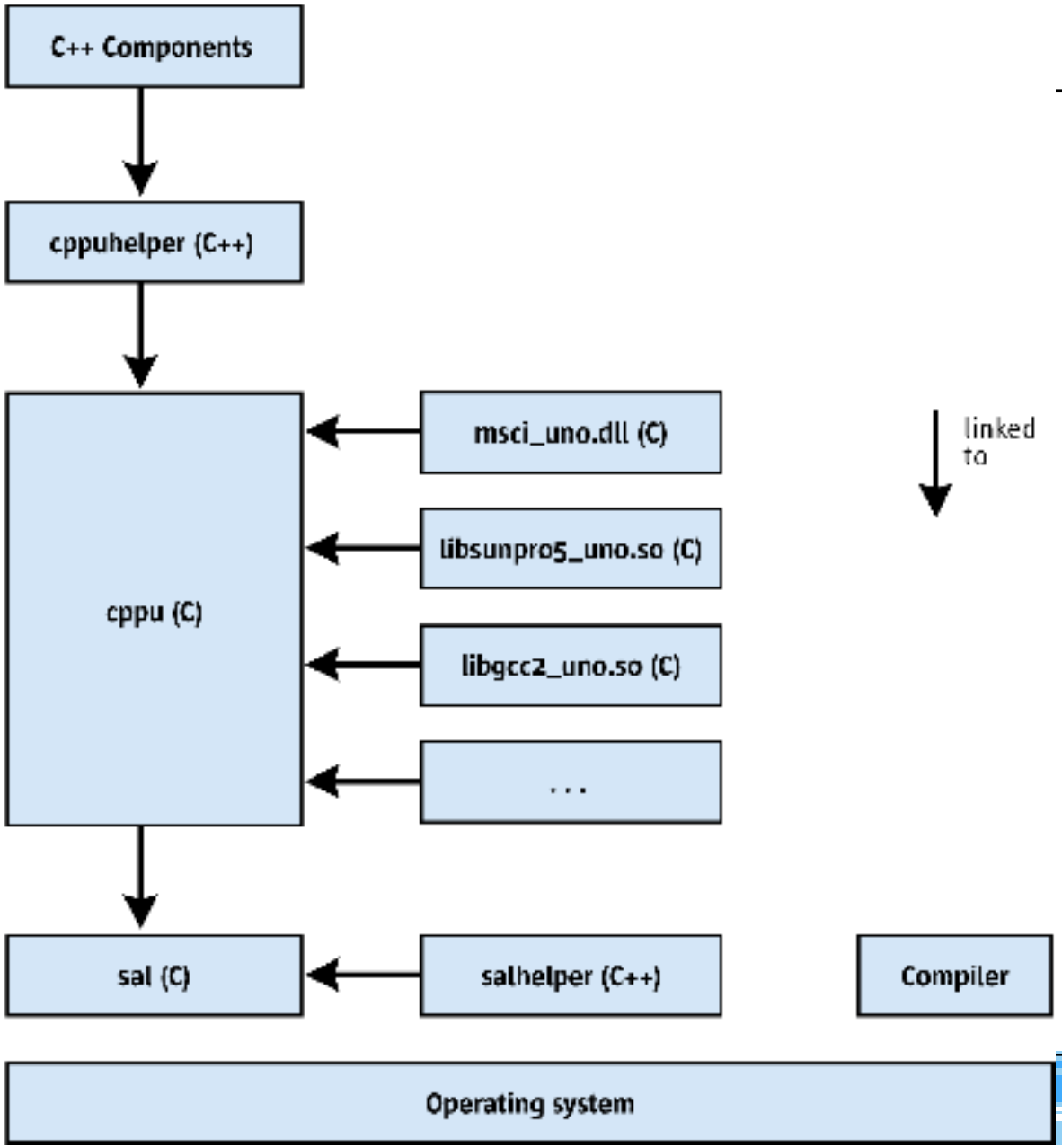
```
    protected XMultiComponentFactory getRemoteServiceManager
(String unoUrl) throws java.lang.Exception {
        if (xRemoteContext == null) {
         // First step: create local component context, get local servicemanager and
         // ask it to create a UnoUrlResolver object with an XUnoUrlResolver interface
            XComponentContext xLocalContext =com.sun.star.comp.
helper.Bootstrap.createInitialComponentContext(null);
            XMultiComponentFactory xLocalServiceManager =
xLocalContext.getServiceManager();
            Object urlResolver  = xLocalServiceManager.
createInstanceWithContext("com.sun.star.bridge.UnoUrlResolver",
xLocalContext );
          // query XUnoUrlResolver interface from urlResolver object
            XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver)
UnoRuntime.queryInterface(XUnoUrlResolver.class, urlResolver);
```

UNO client in Java

```
        // Second step: use xUrlResolver interface to import the remote StarOffice.
ServiceManager,
        // retrieve its property DefaultContext and get the remote servicemanager
            Object initialObject = xUnoUrlResolver.resolve(unoUrl);
            XPropertySet xPropertySet = (XPropertySet)UnoRuntime.
queryInterface(XPropertySet.class, initialObject);
            Object context = xPropertySet.getPropertyValue
("DefaultContext");
            xRemoteContext = (XcomponentContext)UnoRuntime.
queryInterface(XComponentContext.class, context);
        }
        return xRemoteContext.getServiceManager();
    }
}
```

UNO
C++ Binding

Overview of the base shared libraries (C++)



UNO C++ Binding

- File Access
  - osl::FileBase
  - osl::VolumeInfo
  - osl::FileStatus
  - osl::File
  - osl::DirectoryItem
  - osl::Directory

Threads and Thread Synchronization
- osl::Socket
- osl::Mutex
- osl::Condition
- osl::Semaphore

Socket and Pipe
- osl::Socket
- osl::Pipe

UNO Client in C++

```
Reference< XComponentContext > xcomponentcontext;

  xcomponentcontext =

  defaultBootstrap_InitialComponentContext();

  mxLocalMSF = Reference< XMultiServiceFactory >::query
    (xcomponentcontext->getServiceManager());

  const OUString sDesktopLoaderName
    (RTL_CONSTASCII_USTRINGPARAM("com.sun.star.frame.Desktop"));

  rRemoteDesktop =

  Reference< XComponentLoader >::query(mxRemoteMSF-
    >createInstance(sDesktopLoaderName));

  RemoteDocument= rRemoteDesktop->loadComponentFromURL(...)
```

UNO
Automation Bridge

Automation Bridge

- The OpenOffice.org software supports Microsoft's Automation technology.
- Automation is language independent. The respective compilers or interpreters must, however, support Automation.
- The compilers transform the source code into Automation compatible computing instructions.
- UNO was not designed to be compatible with Automation and COM, although there are similarities.

Automation Bridge: Instantiation

- The service manager is the starting point for all Automation clients. The service manager requires to be created before obtaining any UNO object.
- Since the service manager is a COM component, it has a CLSID and a programmatic identifier which is com.sun.star.ServiceManager.
- It is instantiated like any ActiveX component, depending on the language used:
- The only automation specific call is WScript.CreateObject() in the first line, the remaining are OpenOffice.org API calls.

Automation Bridge: Instantiation

- JScript:

```
var objServiceManager= new ActiveXObject("com.sun.star.ServiceManager");
```

- Visual Basic:

```
Dim objManager As Object
Set objManager= CreateObject("com.sun.star.ServiceManager")
```

- VBScript with WSH::

```
Set objServiceManager= WScript.CreateObject("com.sun.star.ServiceManager")
```

- JScript with WSH

```
var objServiceManager= Wscript.CreateObject("com.sun.star.ServiceManager");
```

Automation Bridge: Instantiation

```
//C++

IDispatch* pdispFactory= NULL;

CLSID clsFactory= {0x82154420,0x0FBF,0x11d4,{0x83,
0x13,0x00,0x50,0x04,0x52,0x6A,0xB4}};

hr= CoCreateInstance( clsFactory, NULL, CLSCTX_ALL, __uuidof(IDispatch), (void**)
&pdispFactory);
```

- In Visual C++, use classes which facilitate the usage of COM pointers. If you use the Active Template Library (ATL), then the following example looks like this:

```
CComPtr<IDispatch> spDisp;

if( SUCCEEDED( spDisp.CoCreateInstance("com.sun.star.ServiceManager")))

{   // do something  }
```

Automation Bridge: Registry

The value of the key CLSID\{82154420-0FBF-11d4-8313-005004526AB4}\LocalServer32 reflects the path of the office executable.

Key	Value
CLSID\{82154420-0FBF-11d4-8313-005004526AB4}	"StarOffice Service Manager (Ver 1.0)"
Sub Keys	
LocalServer32	"<OfficePath>\program\soffice.exe"
NotInsertable	
ProgID\com.sun.star.ServiceManager.1	"com.sun.star.ServiceManager.1"
Programmable	
VersionIndependentProgID	"com.sun.star.ServiceManager"

Automation Bridge: Registry

Key	Value
com.sun.star.ServiceManager	"StarOffice Service Manager"
Sub Keys	
CLSID	"{82154420-0FBF-11d4-8313-005004526AB4}"
CurVer	"com.sun.star.ServiceManager.1"

There are three different keys under HKEY_CLASSES_ROOT that have the following values and subkeys:

Automation Bridge: Registry

Key	Value
com.sun.star.ServiceManager.1	"StarOffice Service Manager (Ver 1.0)"
SubKeys	
CLSID	"{82154420-0FBF-11d4-8313-005004526AB4}"

All keys have duplicates under HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.

DCOM

- The Automation bridge maps all UNO objects to automation objects. That is, all those objects implement the IDispatch interface.
- To access a remote interface, the client and server must be able to marshal that interface. The marshaling for IDispatch is already provided by Windows, therefore all objects which originate from the bridge can be used remotely.
- To make DCOM work, apply proper security settings for client and server.

Unsupported COM Features

- The Automation objects provided by the bridge do not provide type information.
  - That is, IDispatch::GetTypeInfoCount and IDispatch::GetTypeInfo return E_NOTIMPL.
- Also, there are no COM type libraries available and the objects do not implement the IProvideClassInfo[2] interface.
- GetIDsOfName processes only one name at a time. If an array of names is passed, then a DISPID is returned for the first name.
- IDispatch::Invoke does not support named arguments and the pExcepInfo and puArgErr parameter.

Learn UNO

	UNO	CORBA	mapping possible
multiple inheritance of interfaces	no	yes	yes
inheritance of structures	yes	no	yes
inheritance of exceptions	yes	no	yes
mandatory base interface for all interfaces	yes	no	yes
mandatory base exception for all exceptions	yes	no	yes
context for methods	no	yes	no
char	no	yes	yes
8 bit string	no	yes	yes
union	no	yes	yes
array	no	yes	yes
assigned values for enum	yes	no	yes
metatype 'type'	yes	no	yes
object identity	yes	no	no
life time mechanism	yes	no	no
succession of one-way calls	yes	no	no
in process communication	yes	no	no
thread identity	yes	no	no
customized calls	no	yes	yes
less code generation	yes	no	no

- StarOffice/OpenOffice.org Software Development Kit (SDK)
  - Developer's Guide(more than 1000 pages)
  - IDL reference with cross references to the Developer's Guide and vice versa.
  - C/C++ And Java UNO reference documentation.
  - development tools
  - examples in Java, C++, OpenOffice.org Basic
  - minimal build environment for the examples
  - run targets in the makefiles for easy running the examples
  - use of the new deployment tool to integrate new components or Basic libraries in an existing office installation.

- UNO Development Kit
  - http://udk.openoffice.org/
  - http://api.openoffice.org/

Questions?