www.hyperworks.de

www.escalon.de

Überblick über die StarOffice und OpenOffice.org API

Ralf Kuhnert
Hyperworks Internet Solutions
Dietrich Schulten
Escalon System-Entwicklung

Linuxtag 2003
Karlsruhe

Einleitung

Dieser Vortrag richtet sich an Entwickler, die MSO Makros reimplementieren, Office Lösungen erstellen oder neue Funktionalität zu OpenOffice.org hinzufügen wollen.

Es soll ein Einblick in die Grundlagen und Möglichkeiten der API vermittelt und die ersten Schritte bei der Verwendung der API mit Basic und Java gezeigt werden.

Die OpenOffice.org API

Die Anwendung OpenOffice.org bietet über die *API* plattformübergreifenden Zugriff auf ihre Funktionsbereiche. Die wichtigsten Funktionsbereiche sind derzeit die vier Dokumenttypen Writer, Calc, Draw und Impress und die Datenbankschnittstelle Base.

Die OpenOffice.org API liefert die Basis für Scripting (Automatisierung) und eigene Erweiterungen zu OpenOffice.org. Derzeit ist die Verwendung der OpenOffice.org API möglich unter Windows, Solaris, Linux, MacOS X und mit Java, C++, StarOffice Basic, Python und allen Sprachen, die MS COM unterstützen.

> Kurze Demo:
> Dokument öffnen, bearbeiten, speichern, drucken.

Die OpenOffice.org API basiert auf *UNO*, einer eigenen Komponententechnologie, die für OpenOffice.org geschaffen wurde. UNO benutzt eine Reihe objektorientierter Techniken dazu, die Austauschbarkeit und Kommunikation von Software-Komponenten über Netzwerk- und Plattformgrenzen hinweg zu ermöglichen. Entwickler müssen sich bei Verwendung der API mit UNO auseinandersetzen. Man begegnet den Konzepten von UNO in der API Referenz und sie schlagen sich natürlich auch in der Art und Weise nieder, wie man mit der OpenOffice.org API programmiert. Da die Konzepte unter anderem eine eigene Begrifflichkeit mitbringen, wird dies oft als Einstiegshürde empfunden.

Die OpenOffice.org API wird nicht nur benutzt, um per Scripting auf das Office zuzugreifen. Das Erlernen der OpenOffice.org API bildet zugleich eine wichtige Grundlage dafür, das Office durch eigene Komponenten zu erweitern und am existierenden Quellcode von OpenOffice.org zu arbeiten. Die Beschäftigung mit der API verleiht Office Entwicklern darum weitreichende Möglichkeiten.

Das Software Development Kit

Das OpenOffice.org SDK ist auf www.openoffice.org erhältlich und enthält wichtige Tools zur Programmierung mit OpenOffice.org.

- *Developer's Guide / Dietrich Schulten und Ralf Kuhnert*
  führt in die Programmierung der OpenOffice.org API ein. Schwerpunkt ist Java, es sind aber auch ausführliche Kapitel zu Basic, C++ und Automation via MS COM enthalten.
- *Referenzen*
  - *API Referenz*: beschreibt die OpenOffice.org Funktionalität
  - *Java UNO Referenz*: beschreibt Werkzeuge für den Zugriff auf die API unter Java
  - *C++ UNO Referenz*: beschreibt Werkzeuge für den Zugriff auf die API mit C++
- *Tools zur Erstellung eigener Office Komponenten*
  Für die Entwicklung eigener Office Komponenten werden diverse Tools benötigt. C++ braucht ausserdem gewissen Libraries und Header, die nicht im Lieferumfang von OpenOffice.org enthalten sind.
- *Wizards für IDEs*
  Das SDK enthält Wizards, die Entwickler bei der Entwicklung eigener Komponenten unterstützen, z.B. für die NetBeans Java IDE.
- *Beispiele*
  Die Beispiele des SDK sind sämtlich für GNU make aufbereitet.

Für die Programmierung unter Basic und Java sind der Developer's Guide und die API Referenz unverzichtbar. Die Nutzung der anderen Teile des SDK ist unter Basic und Java optional.

Aufbau des Vortrags

Im Folgenden geben wir eine kurze Einführung in die wichtigsten Bestandteile von UNO, und zwar soweit sie Entwicklern in der Programmierung mit OpenOffice.org regelmäßig begegnen. Darauf aufbauend werden wir zeigen, wie man mit Hilfe der OpenOffice.org API Dokumente in OpenOffice.org bearbeitet, und einen Ausblick geben, was man über Scripting hinaus mit der OpenOffice.org API bewerkstelligen kann.

UNO - die Komponententechnologie der OpenOffice.org API

Als Komponententechnologie steht UNO neben Corba oder MS COM. Gründe für den Aufbau einer eigenen Komponententechnologie waren vor allem: Einfachheit der Programmierung, Schnelligkeit, Plattformunabhängigkeit.

UNO hat das Ziel, austauschbare Komponenten zu liefern, die über Netzwerk- und Plattformgrenzen hinweg kommunizieren können. Im Folgenden betrachten wir die Mechanismen, mit denen UNO dieses Ziel erreicht.

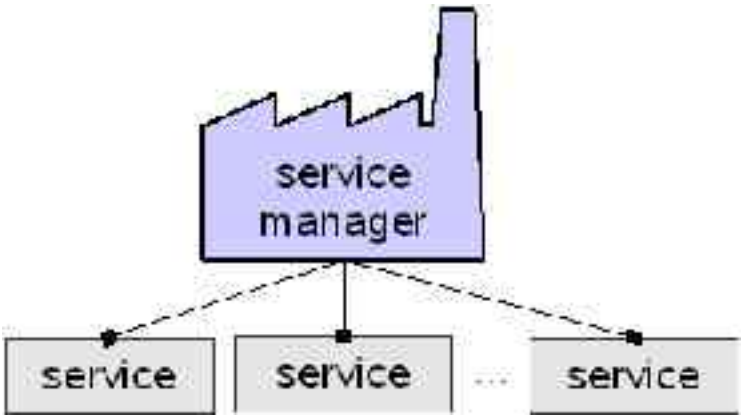
Service Manager

Zunächst eine Begriffsklärung: In unserem Kontext verstehen wir unter einem UNO Objekt eine im Hauptspeicher eines Computers existierende Instanz mit Operationen, die man aufrufen kann, im Gegensatz zu einfacheren Datenstrukturen wie `struct` und `array` oder Typen wie `int`, `boolean` und `string`.

UNO verwendet für die Erzeugung von UNO Objekten ein *Factory Design Pattern*: UNO Objekte werden bei diesem Design Pattern nicht direkt erzeugt (z.B. mit `new SmallTruck()`), sondern über eine Factory-Methode per Namen bei „Factories“ bestellt:

```
aFactory.createInstance("SmallTruck")
```

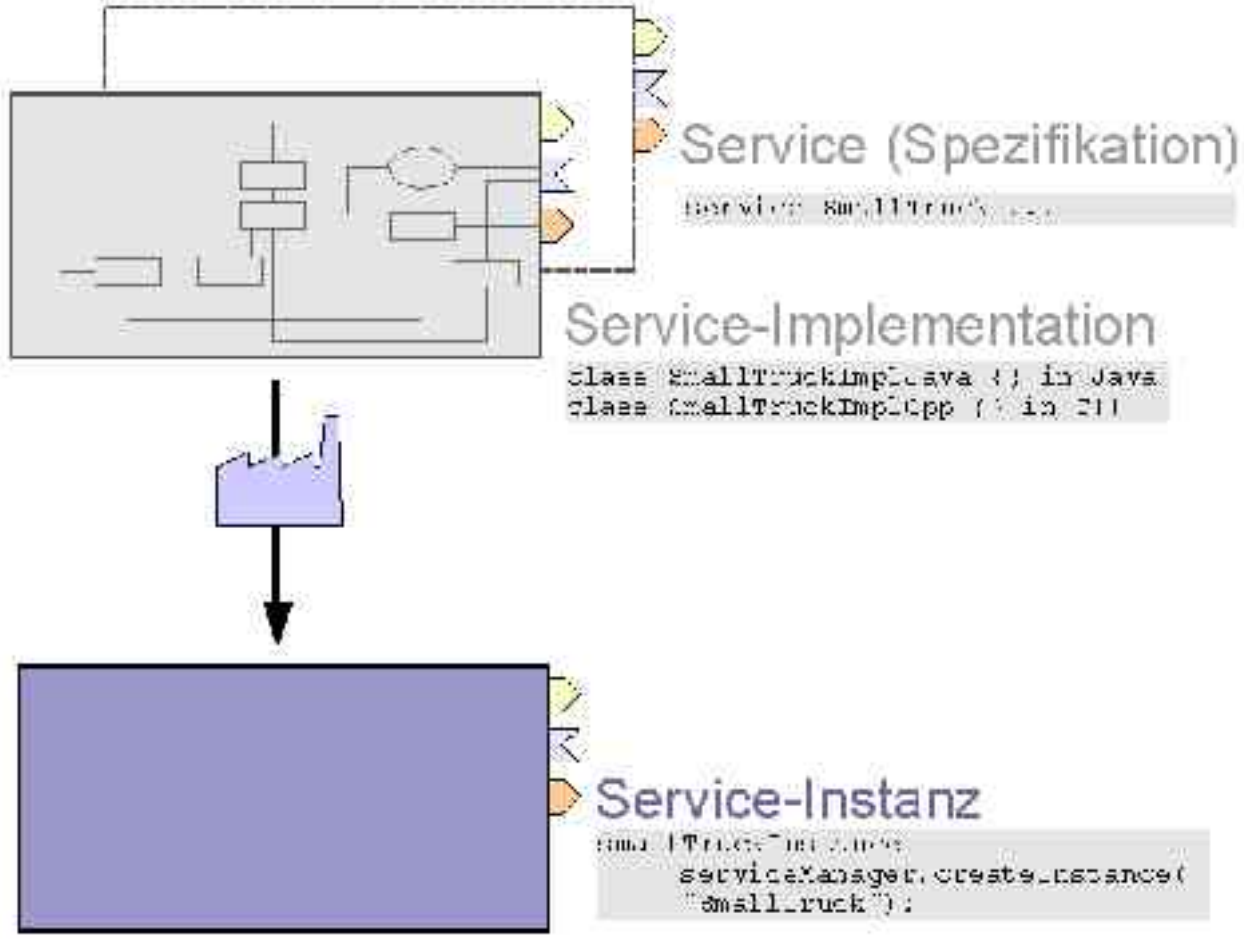
In UNO heißen diese „Factories“ *Service Manager* – sie liefern Instanzen von Objekten anhand des Namens, unter dem sie spezifiziert sind.



Die OpenOffice.org API stellt über *Services* ihre Funktionalität zur Verfügung. Beispiele für solche Services sind der Desktop und der Configuration Service.

Service

Entwickler begegnen dem Begriff Service regelmäßig in der API Referenz. Darum ist es wichtig, zu verstehen, was genau ein Service in der API ist. Das folgende Schaubild stellt das Service Konzept dar.



- ***Service ("Funktionsbeschreibung")***
  Ein Service ist eine abstrakte Spezifikation von UNO-Objekten. „Abstrakt" bedeutet, dass ausschließlich die von außen zugreifbaren Operationen festgelegt werden – wie ein Objekt sein externes Verhalten intern bewerkstelligt, bleibt offen. Zugriff auf ein UNO-Objekt gibt es nur über dafür vorgesehene Operationen.
  Wichtig: Was ein Service kann, wird mit einem eindeutigen Namen bezeichnet. Im oben genannten Beispiel für die Verwendung der Factory Methode `createInstance()` wäre dieser Name `"SmallTruck"`. Ein Servicename ist also streng genommen kein Name für ein konkretes Objekt, sondern nur der Name für eine Anzahl von Fähigkeiten.
- ***Service Implementation ("Bauplan")***
  Um eine Spezifikation mit Leben zu erfüllen, muss in einer Programmiersprache eine Klasse geschrieben werden, die diese Spezifikation erfüllt. Die fertige Klasse ist schon konkreter als die abstrakte Spezifikation, aber benutzen kann man sie noch nicht – es wird eine Umgebung benötigt, die Exemplare dieser Klasse erstellen kann. Diese Umgebung ist der Service Manager. Er erstellt aus der Service Implementation Instanzen, die den beschriebenen Service bieten.
- ***Service Instanz ("Produkt")***
  Eine Service Instanz ist ein konkretes Exemplar eines implementierten Services.

Interfaces und Properties

Services werden durch Interfaces und Properties spezifiziert. Sowohl Interfaces als auch Properties beschreiben einen Zugang zum Service auf der Basis von Operationen.

- **Interface** = Satz von Operationen, die *einen* Funktionsaspekt eines Objekts ausmachen. Durch Interfaces erreicht man Kapselung des Objekts durch dedizierte Zugriffsmethoden für einen ganz

bestimmten Zweck. Im obigen Beispiel `SmallTruck` könnte es ein Interface `XFreightHandler` geben, das aus drei Methoden besteht:

```
//Interface XFreighHandler:
Object aFreight = aSmallTruck.getFreight()
void aSmallTruck.setFreight(Object aFreight)
int aSmallTruck.getFreightVolume()
```

Die beiden get/set Methoden von `XFreightHandler` sind ausschließlich für den Zugriff auf die Eigenschaft `Freight` geeignet. Hat ein Objekt sehr viele Eigenschaften, braucht man bei diesem Ansatz entsprechend viele Interfaces mit get/set Methoden.
Das Interface `XFreightHandler` könnte von einem Schiff oder einem Flugzeug wiederverwendet werden.

- **Property** = Attribut eines Objekts, das von außen über seinen Namen erreichbar ist. Hier erfolgt die Kapselung durch generische Zugriffsmethoden, die beliebige Attribute zugänglich machen. Im obigen Beispiel könnte der `SmallTruck` ein Property `MaxCapacity` haben, auf das folgendermaßen zugegriffen wird:

```
Object aProperty = aSmallTruck.getPropertyValue("MaxCapacity")
void aSmallTruck.setPropertyValue("MaxCapacity", 100)
```

  Man braucht also nur ein einziges Interface für beliebig viele Attribute. Das Interface, in dem die o.g. Methoden definiert sind, ist das Interface `com.sun.star.beans.XPropertySet`.
- Außerdem können Services andere Services mit deren Interfaces und Properties hinzunehmen

Fazit: Die Spezifikation eines UNO Service basiert auf Operationen – entweder in spezialisierten Interfaces oder über das generische Interface zur Verwaltung von Properties.

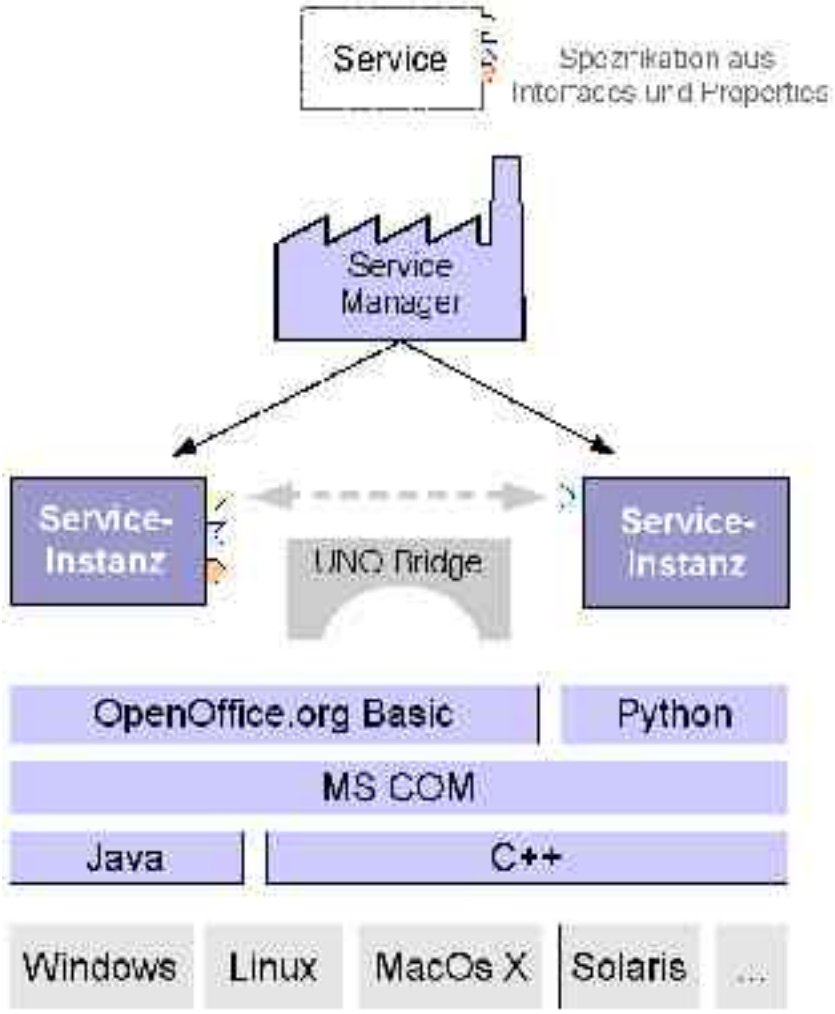
UNO-Bridge

UNO Objekte können miteinander kommunizieren, auch wenn sie in unterschiedlichen Sprachen geschrieben sind oder sich auf verschiedenen Maschinen befinden. Sie tun das, indem Sie über die so genannte *UNO Bridge* gegenseitig ihre Operationen aufrufen. Die UNO Bridge wickelt den Austausch der Daten ab.

Vorteile von UNO

- Implementationen sind ohne Änderungen am Quellcode austauschbar. Das wird durch das Factory Pattern und die abstrakte Spezifikation erreicht. Im Beispiel: Kommt `SmallTruck` mal von einem andern Lieferanten oder hat neue Features, muss nur die Fabrik anders konfiguriert werden – der Quellcode, der den `SmallTruck` bisher benutzt hat, bleibt unverändert.
- Für den Benutzer ist die API stabil, weil bestehende Spezifikationen immer erfüllt werden müssen, und dennoch erweiterbar durch die Einführung zusätzlicher Interfaces und Properties.
- Es ist für den Entwickler transparent, wo sich ein UNO Objekt physisch befindet, er kann seine Operationen immer direkt am entsprechenden Interface aufrufen.
- Das alles plattformunabhängig, vergleichsweise performant und ohne generierte Codeblöcke, die für interne Aufgaben der Komponententechnologie benötigt würden.

Die folgende Grafik zeigt im Überblick die UNO World mit ServiceManager, Service, Interfaces, Properties, Zugriffsmöglichkeiten von verschiedenen Plattformen und den Orten, wo UNO Code laufen kann.



Erster Zugriff auf das Office

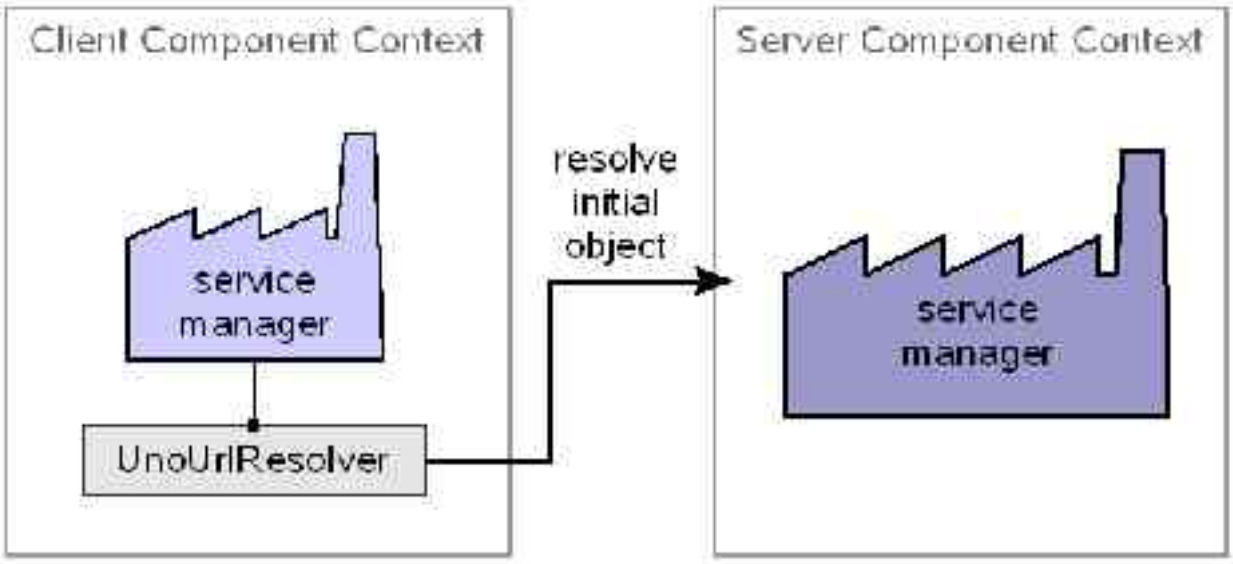
> Demo: Nachdem wir nun die wichtigsten Begriffe der API verstehen, können wir einen ersten praktischen Zugriff auf das Office wagen. Wir demonstrieren die folgenden Beispiele in der Basic IDE und in NetBeans.

Zunächst öffnen wir ein leeres Dokument in Basic. Wir erzeugen mit **Extras – Makros – Makro** eine neue Subroutine und geben folgenden Code ein:

```
' Basic
Dim noProps() 'optionale Parameter für den Ladevorgang
oServiceManager = getProcessServiceManager() 'fest eingebaut
oDesktop = oServiceManager.createInstance("com.sun.star.frame.Desktop")
oDesktop.loadComponentFromURL("private:factory/swriter", "_blank", 0, noProps())
```

Im Basic Editor rufen wir mit dem Symbol **Makro** den Makrodialog auf und führen die Subroutine aus.

Nun dasselbe in Java. In Java ist es notwendig, eine Verbindung zum Office herzustellen. Dazu verwendet die Java UNO Umgebung einen Service names `UnoURLResolver`, der es einem Client erlaubt, eine Referenz auf den Service Manager des Office zu holen. Da nur UNO Objekte über die UNO Bridge kommunizieren können, werden auf beiden Seiten werden UNO Objekte benötigt, die ein UNO Service Manager bereitstellt:



Wie das im einzelnen funktioniert, führt hier zu weit, aber der Anhang enthält eine zehnzeilige Methode `getRemoteServiceManager()`, die das erledigt. Das Kapitel „First Steps“ im Developer's Guide zum SDK enthält eine Anleitung, die den Aufbau der Verbindung Schritt für Schritt erklärt.

Das Java Pendant zum obigen Basic Beispiel unterscheidet sich dadurch, dass man auf die Typprüfungen des Compilers Rücksicht nehmen muss, der sich beim Aufruf von Interface Methoden des jeweiligen Interfaces bewusst sein muss. Der Rest funktioniert genau wie in Basic.

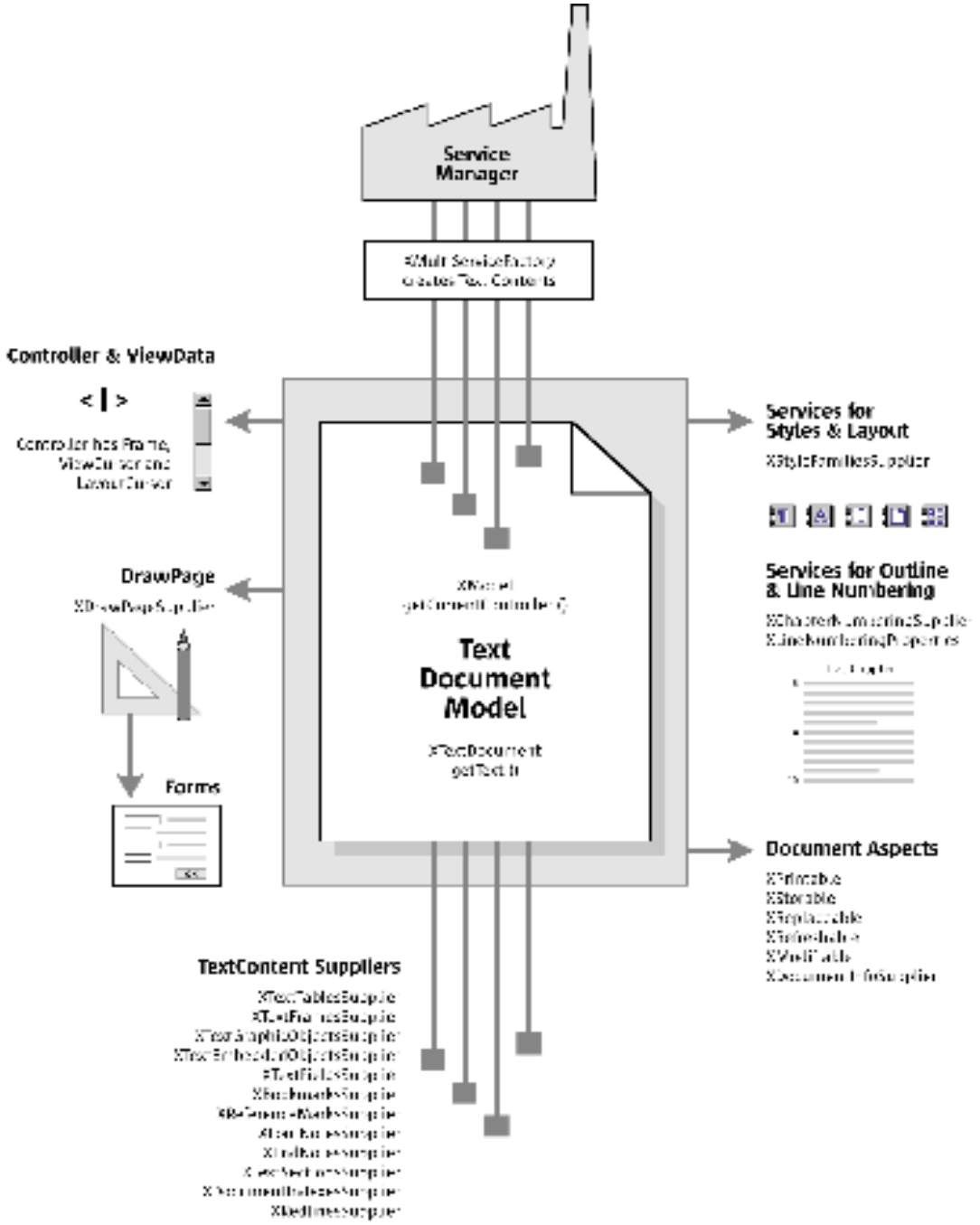
```
import com.sun.star.beans.PropertyValue;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.frame.XComponentLoader;

PropertyValue[] noProps = new PropertyValue[0];     //struct PropertyValue

Object oServiceManager = getRemoteServiceManager(); //liefert Verbindung zum Office
                                                    //im Anhang beschrieben
XMultiServiceFactory xServiceManager =
    (XMultiServiceFactory)UnoRuntime.queryInterface(
        XMultiServiceFactory.class, oServiceManager);

oDesktop = oServiceManager.createInstance("com.sun.star.frame.Desktop");

XComponentLoader xComponentLoader =
    (XComponentLoader)UnoRuntime.queryInterface(
        XComponentLoader.class, oDesktop);

xComponentLoader.loadComponentFromURL(
    "private:factory/swriter", "blank", 0, noProps());
```

Wir betrachten nun die Dokumentmodelle von OpenOffice.org, um zu sehen, wie man mit Dokumenten arbeitet.

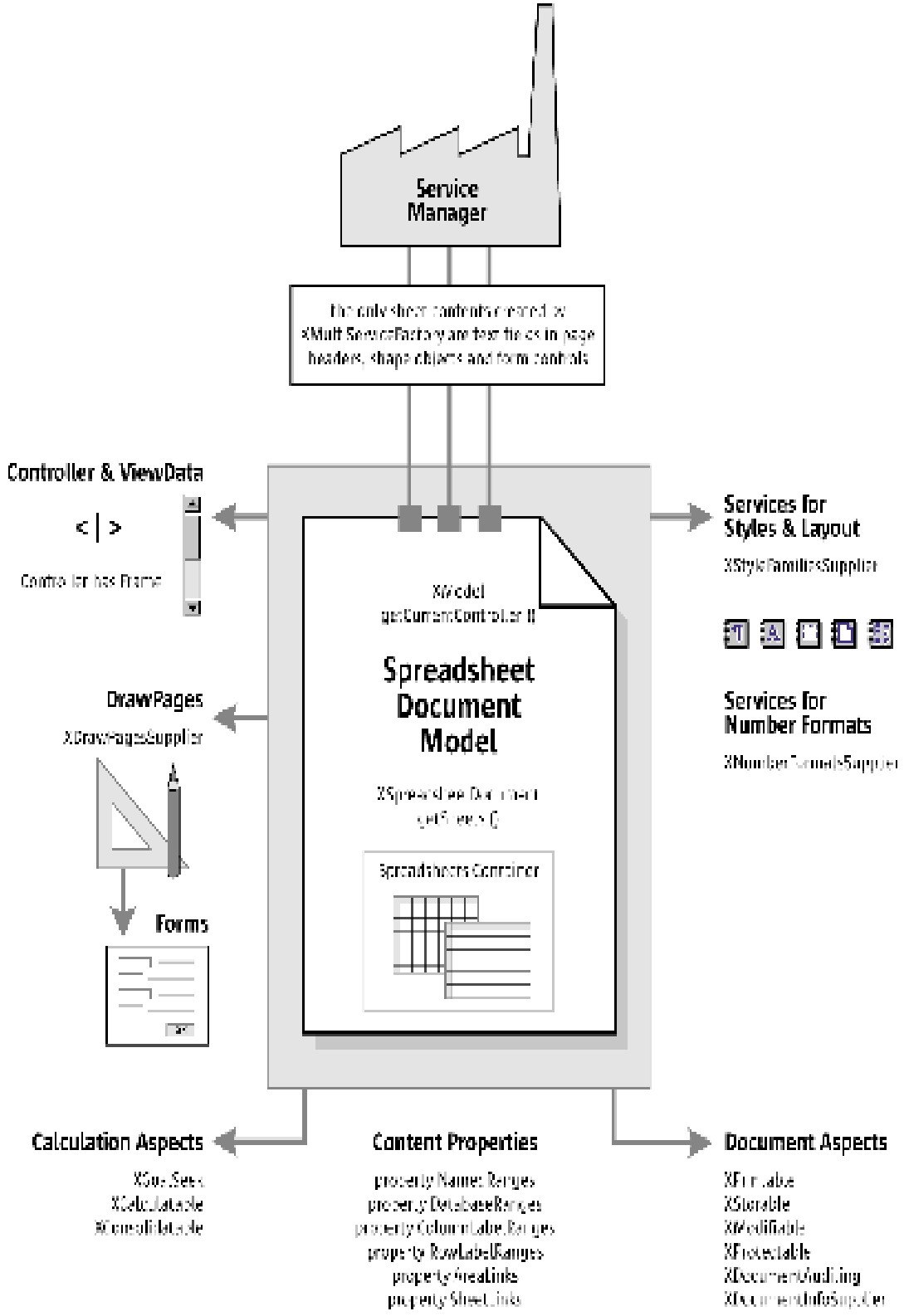
Dokumentmodelle in OpenOffice.org

Die folgenden Schaubilder zeigen zunächst die drei wichtigsten Dokumentmodelle in OpenOffice.org. Im Anschluss besprechen wir die wesentlichen Gemeinsamkeiten aller Dokumentmodelle.

Zunächst das Dokumentmodell eines Writer Dokuments. Es erfüllt die Service Spezifikation `com.sun.star.text.TextDocument`. Wichtig ist im Augenblick der Zugriff auf den Inhalt des Textdokuments. Man bekommt den Inhalt über das Hauptinterface des `TextDocument` Services, `com.sun.star.text.XTextDocument`. Dort gibt es die Methode `getText()`.



Ein Calc Dokument ist ein `com.sun.star.sheet.SpreadsheetDocument`. Zugriff auf die einzelnen Spreadsheets im Dokument gibt es über das Hauptinterface `com.sun.star.sheet.XSpreadsheetDocument`. Dieses Interface hat eine Methode `getSheets()`.



Das dritte Dokumentmodell ist das Dokumentmodell für Draw Dokumente. Dieses Dokumentmodell ist fast deckungsgleich mit dem Modell für Impress.

Draw Dokumente sind `com.sun.star.drawing.DrawingDocument` Services. Das Interface, das Zugriff auf die Zeichnungen im Dokument bietet, ist `com.sun.star.drawing.XDrawPagesSupplier`. Dieses Interface mit seiner Methode `getDrawPages()` steht darum im Mittelpunkt des Schaubilds.

Da auch Writer und Calc Zeichnungen enthalten können, haben auch sie ein entsprechendes Interface, aber dort steht es nicht im Mittelpunkt.

Impress Dokumente sind `com.sun.star.presentation.PresentationDocument` Services. Sie erweitern das DrawingDocument um Interfaces für die Steuerung von Präsentationen.

Die wesentlichen Gemeinsamkeiten der Dokumentmodelle sind:

- Es gibt ein Hauptinterface, das Zugriff auf den eigentlichen Dokumentinhalt bietet. Bei Writer ist das `com.sun.star.text.XTextDocument`, bei Calc `com.sun.star.sheet.XSpreadsheetDocument`, bei Draw und Impress `com.sun.star.draw.XDrawPagesSupplier`.
- Jedes Dokument hat eine eigene Service Factory für Objekte, die im Dokument eingefügt werden können.
- Jedes Dokumentmodell hat einen Controller, der für die Bildschirmpräsentation zuständig ist und den umgebenden Frame kennt. Dieser Controller wird über das `com.sun.star.frame.XModel` Interface bereitgestellt
- Zeichnungselemente sind auf der so genannten Draw Page zu finden. Writer Dokumente haben eine Draw Page, Spreadsheet und Draw/Impress haben mehrere Draw Pages.
- Dokumentweite Styles sind beim `com.sun.star.styles.XStyleFamiliesSupplier` erhältlich.
- Alle Modelle haben gemeinsam, dass sie Office Dokumente sind. Das heißt, man kann sie speichern, drucken, ändern und mit allgemeinen Informationen versehen, wie z.B. das Thema oder den Autor. Der Dokument-Aspekt wird über gemeinsame Dokument Interfaces abgewickelt, die im Service `com.sun.star.document.OfficeDokument` definiert sind. Dessen vorgeschriebene Interfaces sind `com.sun.star.util.XPrintable`, `com.sun.star.frame.XStorable` und `com.sun.star.view.XModifiable`.

Arbeiten mit der API Referenz

Die folgenden Beispiele verwenden eine gemeinsame Routine `ManipulateText()`, um Text in einem Writer, einem Calc und einem Draw Dokument zu bearbeiten. Wir legen unseren Schwerpunkt darauf, wie man selbständig in der API Referenz Problemlösungen erarbeitet auf der Basis des jeweiligen Dokumentmodells und mit Kenntnis der UNO Begrifflichkeit.

`UseTextDocument()` benutzt das Hauptinterface des `TextDocument` Service, um an den Text eines Writer Dokuments zu kommen. `ManipulateText()` bietet Gelegenheit, die Hierarchie des `XText` Interfaces zu beleuchten, und Properties des Text Cursors sowie Konstanten nachzuschlagen.

```
Sub ManipulateText(oText as Variant)
        Dim clause1 as String, clause2 as String
        clause1 = "When Mr. Bilbo Baggins of Bag End announced that he would " + _
                  "shortly be celebrating his eleventy-first birthday,"
        clause2 = " there was much talk and excitement in Hobbiton"
        oText.setString(clause1 + clause2)

        oCursor = oText.createTextCursor()
        oCursor.gotoStart(false)                                    'false: nicht markieren
        oCursor.goRight(len(clause1) - 1, false)  'ans Ende von clause1
        oText.insertString(oCursor, _
                " with a party of special magnificence", false)

        oCursor.goLeft(len("special magnificence"), true)
```

```
        oCursor.setPropertyValue("CharWeight", _
                com.sun.star.awt.FontWeight.BOLD)
End Sub

Sub UseTextDocument
        Dim noProps()
        oServiceManager = getProcessServiceManager() 'fest eingebaut
        oDesktop = oServiceManager.createInstance("com.sun.star.frame.Desktop")
        oDoc = oDesktop.loadComponentFromURL("private:factory/swriter", _
                "_blank", 0, noProps())
        oText = oDoc.getText()
        ManipulateText(oText)
End Sub
```

Bei der Anwendung von `ManipulateText()` auf Calc müssen wir uns zunächst eine Zelle besorgen. Wir zeigen, wie man ausgehend vom Hauptinterface `XSpreadsheetDocument` die nötigen Informationen erhält, um die Zelle zu bearbeiten. Dabei zeigt sich, dass der Text umbrochen werden sollte. Wir gehen der Frage nach, wie man diese Aufgabe anhand der API Referenz löst.

```
Sub UseSpreadsheetDocument
        Dim noProps()
        oServiceManager = getProcessServiceManager() 'fest eingebaut
        oDesktop = oServiceManager.createInstance("com.sun.star.frame.Desktop")
        'oDesktop.loadComponentFromURL("file:///c|/test.sxw", "_blank", 0, noProps())
        oDoc = oDesktop.loadComponentFromURL("private:factory/scalc", _
                "_blank", 0, noProps())
        oSheets = oDoc.getSheets()
        oSheet = oSheets.getByIndex(0)
        oCell = oSheet.getCellByPosition(0, 0)
        oCell.setPropertyValue("IsTextWrapped", true)
        ManipulateText(oCell)
End Sub
```

Text in Draw Dokumenten befindet sich in Shapes auf der DrawPage, die über das Hauptinterface `XDrawPagesSupplier` geliefert wird. Wir erarbeiten anhand der API Referenz die Verwendung der DrawPage und das Einfügen von Shapes auf der DrawPage. Beim Einfügen einer `RectangleShape` stoßen wir auf die Structs `Point` und `Size` und schlagen ihre Verwendung nach. Es zeigt sich, dass eine einfache `RectangleShape` keinen Hintergrund hat und nicht umbricht, sie ist also weder als Shape zu erkennen noch passt sie auf die Seite. Wir nehmen mit Hilfe der API Referenz die nötigen Anpassungen vor.

```
Sub UseDrawingDocument
        Dim noProps()
        Dim aSize as new com.sun.star.awt.Size
        Dim aPosition as new com.sun.star.awt.Point
        oServiceManager = getProcessServiceManager() 'fest eingebaut
        oDesktop = oServiceManager.createInstance("com.sun.star.frame.Desktop")
        'oDesktop.loadComponentFromURL("file:///c|/test.sxw", "_blank", 0, noProps())
        oDoc = oDesktop.loadComponentFromURL("private:factory/sdraw", _
                "_blank", 0, noProps())
        oDrawPages = oDoc.getDrawPages()
        oDrawPage = oDrawPages.getByIndex(0)

        oRect = oDoc.createInstance("com.sun.star.drawing.RectangleShape")
        aSize.Width = 10000
        aSize.Height = 20000
        aPosition.X = 5000
        aPosition.Y = 5000
        oRect.setSize(aSize)
        oRect.setPosition(aPosition)
        oRect.setPropertyValue("TextContourFrame", true)
        oDrawPage.add(oRect)
        ManipulateText(oRect)
End Sub
```

Der Vollständigkeit halber noch die Subroutinen, die die Beispieldokumente speichern und drucken:

```
Sub StoreDocument(oDocument as Variant, aURL as String)
        Dim noProps()
        oDocument.storeAsURL(aURL, noProps())
End Sub

Sub PrintDocument(oDocument as Variant)
        Dim noProps()
        oDocument.print(noProps())
End Sub
```

In Java sind es wieder im wesentlichen die Interfaces und der genauere Umgang mit Typen, die zu beachten sind. Die grundlegende Vorgehensweise ist aber dieselbe, und anhand der Erarbeitung der Lösung in der API Referenz kennen wir die nötigen Interfaces, die man per `queryInterface()` erfragen muss.

```
protected void manipulateText(XText xText) throws com.sun.star.uno.Exception {
  // simply set whole text as one string
  xText.setString("He lay flat on the brown, pine-needled floor of the forest, "
      + "his chin on his folded arms, and high overhead the wind blew in the tops "
```

```
        + "of the pine trees.");

    // create text cursor for selecting and formatting
    XTextCursor xTextCursor = xText.createTextCursor();
    XPropertySet xCursorProps = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xTextCursor);
    // use cursor to select "He lay" and apply bold italic
    xTextCursor.gotoStart(false);
    xTextCursor.goRight((short)6, true);
    // from CharacterProperties
    xCursorProps.setPropertyValue("CharPosture",
        com.sun.star.awt.FontSlant.ITALIC);
    xCursorProps.setPropertyValue("CharWeight",
        new Float(com.sun.star.awt.FontWeight.BOLD));

    // add more text at the end of the text using insertString
    xTextCursor.gotoEnd(false);
    xText.insertString(xTextCursor, " The mountainside sloped gently where he lay;"
     + "but below it was steep and he could see the dark of the oiled road "
     + "winding through the pass. There was a stream alongside the road "
     + "and far down the pass he saw a mill beside the stream and the falling water "
     + "of the dam, white in the summer sunlight.", false);
     // after insertString the cursor is behind the inserted text, insert more text
     xText.insertString(xTextCursor, "\n  \"Is that the mill?\" he asked.", false);
}
```

Ausblick

Die OpenOffice.org API wird nicht nur beim Scripting verwendet. Sie ist auch die Basis für UNO Komponenten, die das Office um neue Funktionalität erweitern. UNO Komponenten sind shared libraries oder jars mit der Fähigkeit, UNO Objekte zu instantiieren, die sich in die OpenOffice.org Umgebung integrieren. Eine neue UNO Komponente kann bestehende Features des Office nutzen, und sie kann aus OpenOffice.org heraus durch die Kommunikationsmöglichkeiten von UNO verwendet werden.

OpenOffice.org bietet eine ganze Anzahl von Ansatzpunkten für solche Erweiterungen:

- Beliebige Objekte in Java oder C++ können aus der Oberfläche von OpenOffice.org angesprochen werden, ihr eigenes User Interface zeigen und die ganze Applikation nutzen
- Calc Add-Ins bieten die Möglichkeit, neue Formeln in Calc einzubauen, die im Formelautopiloten angezeigt werden
- Chart Add-Ins können neue Charttypen in das Charting Tool einführen, das für die grafische Aufbereitung von Daten verwendet wird
- Neue Datenbanktreiber lassen sich im Office installieren, die den Datenzugriff erweitern. Derzeit ist beispielsweise ein nativer PostGreSQL Treiber in Arbeit.
- Ganze Module der Anwendung sind austauschbar, so zum Beispiel das linguistics Modul, das für OpenOffice.org eine andere Implementation als für StarOffice hat.
- Entwickler können neue Dokumenttypen integrieren, zum Beispiel könnte ein PIM Dokumente für Nachrichten, Kalender, Aufgaben und Journal zum Office hinzufügen, ein Projekt Manager könnte ein neues Projekt Dokument liefern.
- Filterentwickler können auf das offene XML Dateiformat aufsetzen und neue Dateiformate lesen und schreiben unter Ausnutzung des XML Formats als Zwischenschicht. Solche Filterkomponenten lassen sich nahtlos in die Oberfläche integrieren

Ab OpenOffice.org 1.1 gibt es umfassende Unterstützung für die Erweiterung durch solche Komponenten. Der gesamte Produktzyklus einer Komponente ist jetzt abgedeckt.

Das Design und die Entwicklung von Java Komponenten wird durch neue Wizards für NetBeans erleichtert. Diese Wizards sind im SDK enthalten. Es gibt Wizards für allgemeine Komponenten, für Calc AddIns und für die Erstellung von Service-Spezifikationen, die man braucht, um selbst neue Services zu definieren.

Komponenten können sich mit Hilfe von XML Konfigurationsdateien in die Oberfläche von OpenOffice.org integrieren. Man kann eigene Menüs, Symbole für Symbolleisten und Hilfemenüs hinzufügen.

Die Verteilung und Installation von Komponenten wird durch einen Package Installer erledigt, der neue Komponenten und ihre Benutzungsschnittstelle in Netzwerk- und Einzelplatzinstallationen einbaut. In der Produktionsphase vereinfacht der Package Installer die Wartung der Komponente, indem er einfache Bugfixes und Upgrades erlaubt. Wenn eine Komponente nicht länger benötigt wird, kann der Package Installer sie mit einem simplen Kommandozeilenaufruf komplett aus der Office Installation entfernen.

Neben der Möglichkeit neue Komponenten zum Office hinzuzufügen, kann natürlich auch an der Codebasis von OpenOffice.org selbst gearbeitet werden. Die OpenOffice.org API und UNO wird direkt in der Codebasis von OpenOffice.org verwendet, so dass man mit dem Erlernen der API automatisch die Mittel an die Hand bekommt, die Codebasis zu erweitern oder Bugs zu beheben.

Auch wenn Entwickler sich anfangs oft eine vereinfachte API wünschen, bei der man sich um UNO nicht zu kümmern braucht, ist es darum sehr lohnend, sich mit UNO zu befassen und die API zu erlernen. Nur auf diesem Weg wird es für OpenOffice.org zunehmend Beiträge aus der OpenSource Gemeinde geben.

Anhang

Nachfolgend ein Codebeispiel zum Aufbau einer Connection mit Java. Das komplette Beispiel mit Erklärung finden Sie im Kapitel First Steps aus unserem Developer's Guide zum SDK.

```
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;

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

    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) {
                 System.err.println("No process listening on the resource");
                 e.printStackTrace();
                 throw e;
         }
         catch (com.sun.star.lang.DisposedException e) { //works from Patch 1
            xRemoteContext = null;
            throw e;
        }
    }

    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);

            // 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();
    }
}
```