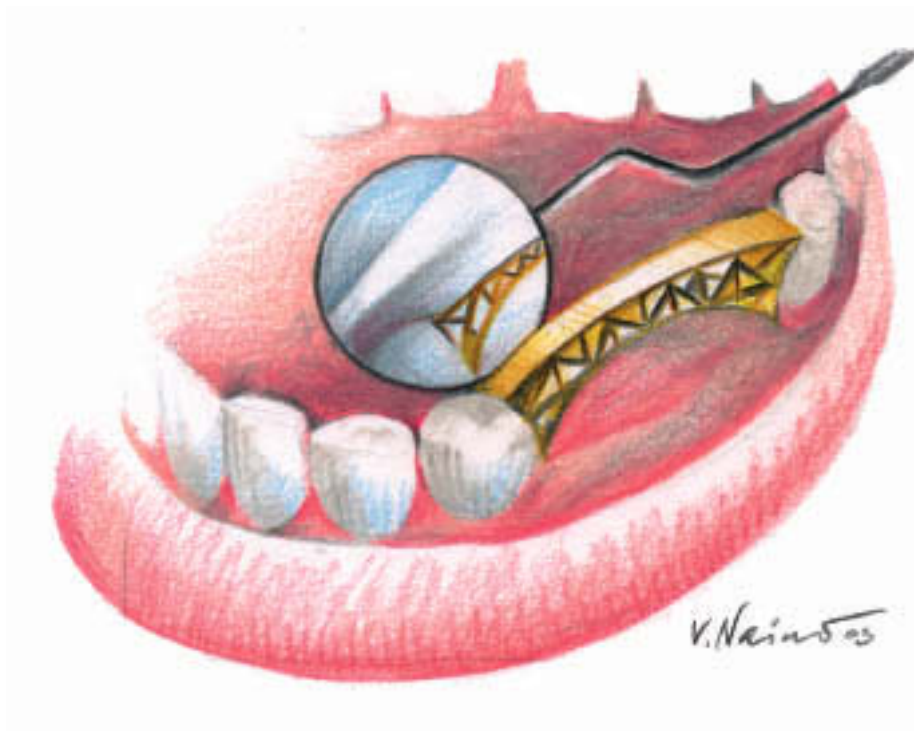


## Centro.studio: Eine Verbindung zwischen Java, C++ und COM

von Bert Rosner

# Brückenbau

Mit Centro.studio steht ein Produkt bereit, mit dem eine bidirektionale Brücke sowohl zwischen Java und C++ als auch zwischen Java und Microsoft COM zur Verfügung gestellt wird. Das ermöglicht die Kommunikation zwischen Java, C++, Visual Basic und Delphi. Durch die Verwendung des Java Native Interface (JNI) wird eine Unabhängigkeit vom verwendeten Java Runtime Environment (JRE) und eine hohe Geschwindigkeit erreicht.



Java bietet mit dem Java Native Interface (JNI) [1] einen Standard für die Kommunikation zwischen Java und kompiliertem Code. Es ist mit JNI möglich, sowohl Java-Methoden in der Programmiersprache C/C++ zu implementieren als auch Java-Methoden von C/C++ aus aufzurufen. Leider ist die Verwendung von JNI mit viel manueller Tipparbeit verbunden.

Das Component Object Model (COM) ist in der Microsoft-Welt der traditionelle Binärstandard für die Kommunikation von Komponenten, die mit unterschiedlichen Programmiersprachen entwickelt wurden. Durch betriebssystem-

spezifische Erweiterungen von Microsoft war es mit der Microsoft-JVM möglich, sowohl Java-COM-Server als auch Java-COM-Clients zu implementieren. So konnten Visual Basic und Java miteinander kommunizieren. Die Microsoft-JVM wird jedoch wegen der bekannten Streitigkeiten mit Sun derzeit nicht weiter entwickelt.

Centro.studio [2] besteht aus den drei Produkten Centro.java, Centro.com und Centro.jcom. In diesem Beitrag werden nur die ersten beiden vorgestellt.

Centro.java verbindet C++ und Java auf einfache Weise unter Verwendung von JNI, die umfangreiche Tipparbeit entfällt.

Zusätzlich wird das prozedurale Mapping von JNI in ein objektorientiertes konvertiert. Damit können sowohl Java-Klassen in C++ als auch C++-Klassen in Java 1:1 verwendet werden.

Centro.jcom implementiert eine bidirektionale Java-COM-Brücke. Dabei dient JNI ebenfalls als Bindeglied in die COM-Welt, sodass die Beschränkung auf die Microsoft JVM entfällt.

### Java als Server für COM und C++

Um Java-Klassen in C++ oder COM verwenden zu können, müssen C++-Proxyklassen generiert werden. Für die Generierung wird die CLASS-Datei benötigt, die

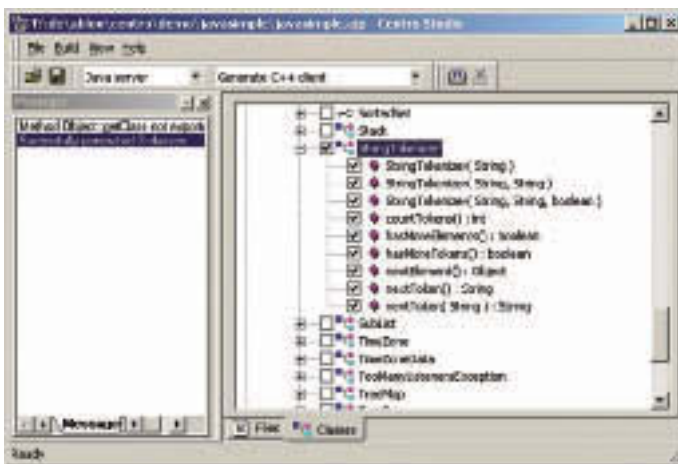


Abb. 1: Auswahl der zu generierenden Klassen, Schnittstellen, Methoden in Centro-Studio

separat oder in einer JAR-Datei vorliegen kann. Die Vorgehensweise wird anhand der Java-Klasse *StringTokenizer* erläutert. Da *StringTokenizer* Bestandteil des JDK ist, wird die JAR-Datei *rt.jar* des JDK benötigt.

Zur Generierung der Proxyklassen wird CentroStudio gestartet. Zuerst wird eingestellt, ob der Proxycode für einen C++-Client oder einen COM-Client generiert werden soll. In einer Baumansicht werden die zu generierenden Klassen, Schnittstellen, Methoden, Eigenschaften und Konstanten ausgewählt (Abb. 1). In unserem Beispiel wird die Java-Klasse *StringTokenizer* mit allen Methoden verwendet. Nach Betätigen der Schaltfläche GENERATION werden die C++-Proxyklassen generiert.

Der C++-Client kann die generierte C++-Proxyklasse wie eine normale C++-Klasse verwenden:

```
StringTokenizer tokenizer("This is a test.");
while (tokenizer.hasMoreTokens())
{
    cout << tokenizer.nextToken() << endl;
}
```

Bei Verwendung des Operators *new* empfiehlt sich die Verwendung des Klassen-Templates *java\_ptr*, welches eine Garbagecollection unter Verwendung von Referenzzählung implementiert:

```
java_ptr<StringTokenizer> tokenizer = new StringTokenizer
    ("This is a test.");
while (tokenizer->hasMoreTokens())
{
    cout << tokenizer->nextToken() << endl;
}
```

Im Falle eines COM-Servers sind die generierten C++-Proxyklassen nur ein Implementierungsdetail. Deshalb generiert CentroStudio eine Visual C++-Projektdatei, die ausgeführt werden muss. Danach existiert eine DLL mit der COM-Implementierung, eine COM-IDL-Datei sowie eine COM-Typbibliothek. Zusätzlich wird der COM-Server registriert.

Centro.jcom verwendet duale Schnittstellen für die Implementierung des COM-Servers. Deshalb kann Visual Basic sowohl mit der *IDispatch*-Schnittstelle als auch mit virtuellen Funktionstabellen arbeiten. Das folgende Beispiel verwendet virtuelle Funktionstabellen:

```
Dim tokenizer As New StringTokenizer
tokenizer.init("This is a StringTokenizer")
```

```
While (tokenizer.hasMoreTokens())
MsgBox tokenizer.nextToken()
Wend
```

Da COM keine Konstruktoren mit Argumenten kennt, wird der Java-Konstruktor in einen COM-Defaultkonstruktor und eine *Init*-Methode aufgeteilt.

### Java als C++-Client

C++-Klassen können auch in Java verwendet werden – dazu das berühmte „Hello World“-Beispiel. Die C++-Klasse *CppOut* mit der Methode *HelloWorld* soll in Java verwendet werden:

```
class CppOut : public de::ablon::centro::AutoObject
{
public: void HelloWorld ()
        { std::cout << "Hello world."; }
};
```

Die C++-Klasse muss dabei von der Basisklasse *de::ablon::centro::AutoObject* abgeleitet werden, wobei Mehrfachvererbung erlaubt ist. Für diese Klasse wird eine so genannte Centro-Schnittstelle erstellt:

```
auto_class(CppOut)
auto_begin

    auto_constructor_0 ();
    auto_method_0 (void, HelloWorld);

auto_end
```

Die Centro-Schnittstelle ist eine Kombination von C++-Makros und -Templates. Besonders hervorzuheben ist die Typsicherheit zur Kompilierzeit – stimmen die Parameter einer Methode in der Centro-Schnittstelle nicht mit ihrer C++-Deklaration überein, wird ein Compilerfehler erzeugt.

Der Code mit der Centro-Schnittstelle wird kompiliert und daraus eine dynamische Bibliothek erstellt. Diese dynamische Bibliothek enthält die Implementierungen der Java *native*-Methoden entsprechend den JNI-Konventionen. Unter Verwendung von *CentroJavaProxyGenerator* werden dann die Java-Klassen mit *native*-Methoden generiert.

Es ist möglich, in den Centro-Schnittstellen Klassen, Schnittstellen, Vererbung, statische Methoden und statische Eigenschaften zu verwenden. So können nahezu alle objektorientierten Konstrukte von C++ nach Java exportiert werden. Als Funktionsparameter sind die einfachen C++-Datentypen, alle von *AutoObject* abgeleiteten Klassen sowie Standardvektoren (*std::vector<...>*) erlaubt. Eigene Datentypen können integriert werden.

Im folgenden Beispiel wird die Klasse *CppInheritedClass* von der Klasse *CppBaseClass* abgeleitet und implementiert die Schnittstelle *CppInterface*.

```
// C++ Headerdatei
class CppInheritedClass : public CppBaseClass
    , virtual public CppInterface
{
    ...
};

// Centro Schnittstelle
auto_class(CppInheritedClass)
```

```
auto_extends(CppBaseClass)
auto_implements(CppInterface)
auto_begin
...
auto_end
```

Durch so genannte Centro-Attribute, gekennzeichnet durch das Makro *auto\_attributes*, können einzelne zusätzliche Eigenschaften gesetzt werden. Im folgenden Beispiel wird in Java der Name der Methode *Hello World* in *hello* verändert:

```
auto_class(CppOut)
auto_begin

auto_constructor_0 ();
auto_attributes.name(„hello“)
auto_method_0 (void, HelloWorld);

auto_end
```

Nebenbei bemerkt kann mit nahezu der gleichen Vorgehensweise unter Verwendung von *Centro.com* ein C++-COM-Server erstellt werden.

## Java als COM-Client

Um COM-Objekte mit *Centro.jcom* in Java verwenden zu können, müssen C++-Proxyklassen generiert werden. Für die Generierung wird die COM-Typbibliothek benötigt.

Zur Generierung wird *CentroStudio* gestartet. In einer Baumansicht können die zu generierenden COM-Objekte,

COM-Schnittstellen, Methoden, Eigenschaften und Aufzählungstypen ausgewählt werden. Nach Betätigen der Schaltfläche *GENERATION* werden die C++-Proxyklassen sowie die Visual C++-Projektdatei generiert.

Analog zum *Centro.jcom* COM-Server sind die generierten Proxyklassen nur ein Implementierungsdetail. Deshalb muss die generierte Visual C++-Projektdatei ausgeführt werden. Danach existieren Java-Proxyklassen für alle COM-Schnittstellen und COM-Objekte.

Besonders einfach ist in *Centro.jcom* die Ereignisbehandlung gelöst. Die Ableitung von einer generierten Java-Proxyklasse ist ausreichend, um einen Eventhandler zu erstellen.

Im folgenden Beispiel löst der COM-Server *ComObject* Ereignisse der Schnittstelle *\_IEvent* aus. Dazu die COM-IDL

```
dispinterface _IEvent
{
...
[id(1), helpstring(“Methode OnEvent“)] HRESULT
OnEvent([in] int Index);
};
coclass ComObject
{
...
[default, source] dispinterface _IEvent;
};
```

Für die Schnittstelle *\_IEvent* wird die Java-Proxyklasse *\_IEventImpl* generiert.

Von dieser Klasse muss der Eventhandler abgeleitet werden.

```
public class EventListener extends _IEventImpl
{
public void OnEvent( int Index)
{ System.out.println( “Listener Index:“ + Index);}
}
```

Nun kann die ebenfalls generierte Methode *addListener* zum Anmelden des Eventhandlers benutzt werden und fertig ist die Ereignisbehandlung:

```
ComObject obj= new ComObject();
obj.addListener( new EventListener());
```

## Ausnahmebehandlung

Sowohl in *Centro.java* als auch in *Centro.jcom* werden die Ausnahmen über die Systemgrenzen korrekt weitergegeben. Java-Ausnahmen werden in C++-Ausnahmen bzw. in erweiterte COM-Fehlercodes (Schnittstelle *IErrorInfo*) umgewandelt. C++-Ausnahmen werden in Java-Ausnahmen konvertiert. Analog werden COM-Fehler in Java-Fehler umgewandelt. Die Fehlerbehandlung ist über das Singleton-Pattern implementiert, sodass durch Überschreiben der von *Centro.studio* definierten Ausnahmebehandlung eine eigene Ausnahmebehandlung verwendet werden kann.

## Fazit

*Centro.studio* ist ein solides Produkt für die Verbindung von Java mit C++ und COM. Als vorteilhaft erweist sich, dass bei beiden Brücken jeweils beide Richtungen vorhanden sind. Durch umfangreiche Demoprogramme fällt die Einarbeitung leicht. Anwender, die sowohl eine Brücke zwischen Java und C++ als auch zwischen Java und COM benötigen, können mit einer einzigen Technologie beide Anforderungen lösen. Für zukünftige Versionen wäre es schön, wenn für *Centro.java* nicht nur die Runtime, sondern auch die Codegeneratoren unter den verschiedenen Unix-Systemen einsatzfähig wären. ■

## Links & Literatur

- [1] Tutorial JNI: [java.sun.com/docs/books/tutorial/native1.1/](http://java.sun.com/docs/books/tutorial/native1.1/)
- [2] *Centro.studio* Website: [www.ablon.de/](http://www.ablon.de/)

## Die Centro-Elemente

### Centro.java

- Bidirektionale Java-C++-Brücke
- Java-Klassen können in C++-Programmen wie C++-Klassen verwendet werden
- C++-Klassen können in Java-Programmen wie Java-Klassen verwendet werden
- Betriebssysteme: Windows, Linux, AIX, IRIX
- Preis (exkl. MwSt.) Windows: 850.- pro Entwickler – keine Runtime-Lizenzen
- Demoversion: [www.ablon.de](http://www.ablon.de)

### Centro.jcom

- Bidirektionale Java-COM-Brücke
- Erstellung von Java-COM-Clients mit Ereignisbehandlung
- Erstellung von Java-COM-Servern
- Preis (excl. MwSt.) Vollversion: 1900.- € pro Entwickler – keine Runtime-Lizenzen
- Preis (excl. MwSt.) Client ohne Events: 850.- € pro Entwickler – keine Runtime-Lizenzen
- Demoversion: [www.ablon.de](http://www.ablon.de)