Studienarbeit 01/01

Entwicklung eines COM-Automatisierungsservers als Bestandteil des Softwarewerkzeugs GO

cand. Ing. Carsten Schu

Erklärung

Hiermit versichere ich, dass der vorliegende Text im Rahmen einer selbständig durchgeführten Studienarbeit am Lehrstuhl für Software-Technik der Ruhr-Universität-Bochum entstanden ist und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden.

Weiterhin möchte ich mich hiermit an dieser Stelle für die qualifizierte Unterstützung durch meinen betreuenden Assistenten Dipl. Ing. Carsten Mielke bedanken.
Bedanken möchte ich mich ebenfalls bei Dipl. Inform. Peter Ziesche, der mich mit Literatur und anregenden Fachgesprächen unterstützt hat.

Dank gebührt auch der Firma oTRIs, die mit Ihrem Wunsch nach einer COM-Schnittstelle im GO diese Studienarbeit erst möglich gemacht hat. Namentlich bedanke ich mich hiermit bei Dipl. Ing. Thomas Richter.

Bearbeitender Student:	cand. Ing. Carsten Schu
Matrikelnummer:	108 096 218 161
Betreuender Assistent:	Dipl. Ing. Carsten Mielke
Prüfer:	Prof. Dr.-Ing. habil. Helmut Balzert
Bearbeitungszeitraum:	1. November 2000 – 1. Februar 2001

Zusammenfassung

Ziel dieser Studienarbeit ist die Erweiterung des Software-Werkzeuges GO (***Generating Objects***) zu einem Automatisierungs-Server.
GO wurde am Lehrstuhl für Software-Technik entwickelt. Es ermöglicht die objektorientierte Analyse (OOA) mit Hilfe der UML (***Unified Modelling Language***).
Über die zu erstellende Automatisierungs-Schnittstelle soll es dem Software-Generierungswerkzeug JANUS der Firma oTRIs ermöglicht werden, mit GO zu kommunizieren. Dabei soll das Auslesen des UML-Modells aus GO und das Modifizieren einzelner Modelleigenschaften (Fernsteuerung der Anwendung) über COM (***Component Object Model***) ermöglicht werden.
Um dieses Ziel zu erreichen, mussten die benötigten COM-Komponenten und die für den Zugriff notwendigen COM-Schnittstellen identifiziert werden. Anschließend wurden die COM-Komponenten unter Zuhilfenahme der ATL (***Active Template Library***) implementiert.

Schlüsselwörter

Automatisierungs-Server, COM, C***omponent Object Model***, ATL, ***Active Template Library***, GO, ***Generating Objects***, Objektorientierte Analyse, OOA, UML, ***Unified Modelling Language***

Abstract

The objective of this study is the enhancement of the software-tool GO (generating objects) towards an automation server.
GO was developed at the institute of software engineering. It enables object-oriented analysis by means of UML (unified modelling language).
With the generated automation interface, it should be possible for the JANUS software-generating tool to communicate with GO. JANUS was created by firm oTRIs.
It should be feasible to read a UML-model from GO and to modify several model properties (remote control application) by the use of COM (component object model).
To achieve this aim it was necessary to identify the required COM objects and interfaces needed for access. Subsequently the COM objects were implemented by using the active template library.

Keywords

automation server, COM, component object model, ATL, active template library, GO, generating objects, object-oriented analysis, OOA, UML, unified modelling language

INHALTSVERZEICHNIS

1 EINLEITUNG 1

2 ZIELBESTIMMUNG 3

3 VERWENDETE TECHNIKEN 5

3.1 COM (Component Object Model) 5
3.1.1 COM-Grundlagen 5
3.1.2 COM-Schnittstellen 6
3.1.2.1 Die Schnittstelle IUnknown 8
3.1.2.2 IDL (Interface Definition Language) 9
3.1.2.3 Typbibliotheken 11
3.1.3 COM-Komponenten 12
3.1.3.1 Registrierung 12
3.1.3.2 Erzeugen von Objekten 13
3.1.3.3 Die Schnittstelle IClassFactory 13
3.1.4 Module 14
3.1.5 Automatisierungsserver 16
3.1.5.1 Die Schnittstelle IDispatch 16
3.1.5.2 Automatisierungs-Datentypen 17
3.1.5.3 Dual-Interface 18

3.2 ATL (Active Template Library) 19
3.2.1 ATL-Grundlagen 19
3.2.2 Verwendete Template-Klassen und Makros 20
3.2.2.1 CComModule 20
3.2.2.2 Registry-Scripts 22
3.2.2.3 CComObjectRootEx 23
3.2.2.4 CComCoClass 26
3.2.2.5 IDispatchImpl 26
3.2.2.6 CComObject 28
3.2.2.7 Die Smart-Pointer-Klasse CComQIPtr 29

4 DIE COM-SCHNITTSTELLE VON GO 31

4.1 Bezeichner-Konventionen 31

4.2 Identifikation der COM-Komponenten und Schnittstellen 32
4.2.1 GOApplication 34
4.2.2 GOElement 35
4.2.3 GOCollection 37
4.2.4 GOPackage 38
4.2.5 GOModel 39
4.2.6 GOClass 40
4.2.7 GOAttribute 42
4.2.8 GOOperation 43
4.2.9 GOParameter 44
4.2.10 GOAssociation 45
4.2.11 GORole 46

4.3 Vorgehen bei der Erstellung des Automatisierungsservers 47
4.3.1 Hinzufügen der ATL-Unterstützung zu GO 47
4.3.2 Modellierung der Komponenten und Schnittstellen in Rational Rose 2000 49

4.4 Implementierung der COM-Komponenten 51
4.4.1 Erzeugung und Initialisierung der COM-Objekte 52

4.4.2 Zerstörung der COM-Objekte 54
4.4.3 Lesender und schreibender Zugriff auf Eigenschaften 55
4.4.4 Rückgabe eines Schnittstellenzeigers 56

4.5 Benutzung der COM-Schnittstellen 57

5 STATISTIK 59

6 AUSBLICK 61

ANHANG A – LASTENHEFT 63

ANHANG B – PFLICHTENHEFT 65

ANHANG C – OOA-MODELL 70

ANHANG D – GLOSSAR 85

ANHANG E – QUELLTEXTE 87

E.1 Die Schnittstellen-Definition »GO.idl« 87
E.2 ATL-Unterstützung in der Anwendung 93
E.3 Registry-Dateien 98
E.4 Schnittstellenimplementierungen 104
E.5 Test-Client in Visual Basic 147

ANHANG F – ABBILDUNGSVERZEICHNIS 151

ANHANG G – TABELLENVERZEICHNIS 152

ANHANG H – LITERATURVERZEICHNIS 153

1 Einleitung

Generating Objects (GO) ist ein Software-Modellierungswerkzeug, das am Lehrstuhl für Software-Technik entwickelt wird. GO ermöglicht die objektorientierte Analyse (OOA) unter Zuhilfenahme von UML-Diagrammen. Das Fachkonzept von GO basiert auf dem UML-Metamodell. Auf eine strenge Trennung von Fachkonzept und Darstellung wurde Wert gelegt. So können verschiedene Sichten (Diagramme) des selben Fachkonzeptes dargestellt werden.

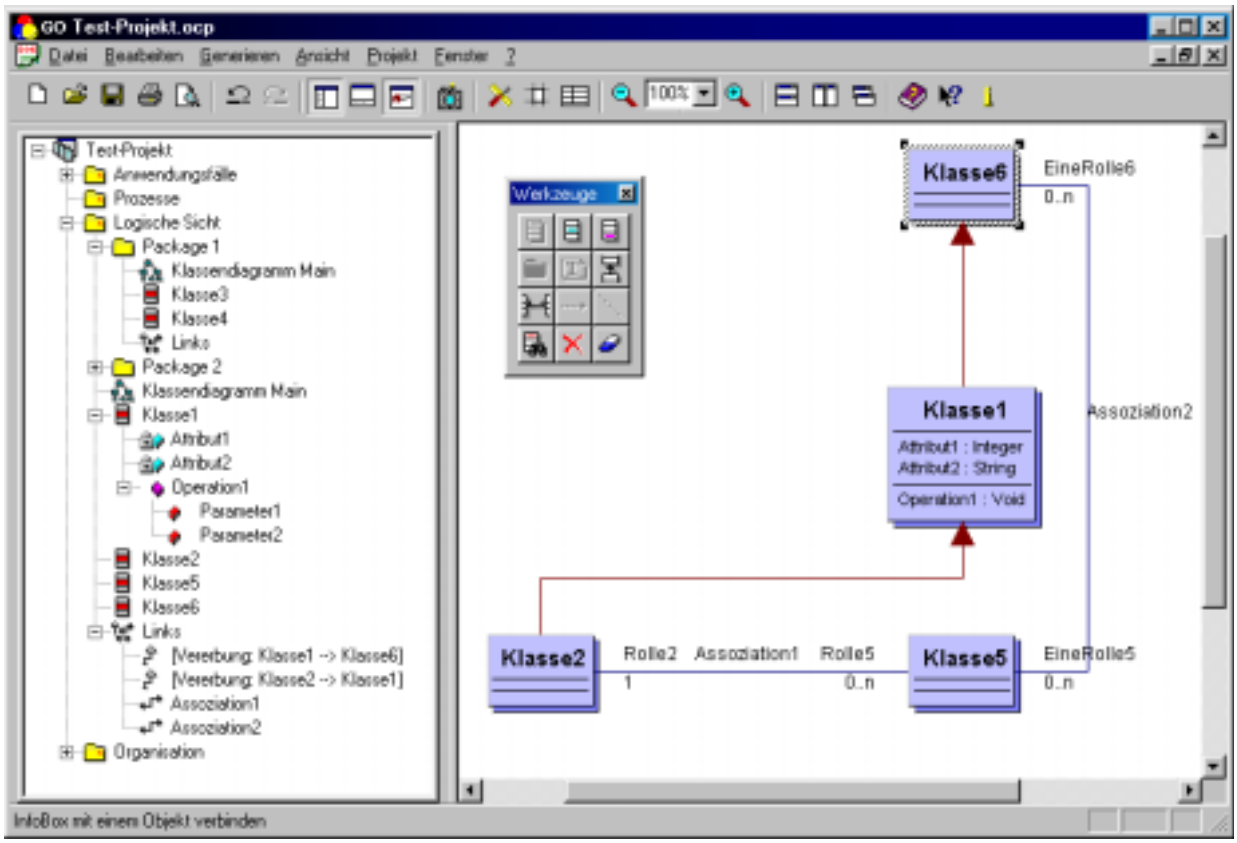

Abb. 1-1: Anwendungsoberfläche von GO (Generating Objects)

Neben einer eigenständigen Codegenerierung soll GO in Zukunft auch als Modellierungswerkzeug (***Frontend***) für den JANUS-Generator der Firma oTRIs genutzt werden.

Der JANUS-Generator ist ein Werkzeug für die Softwareentwicklung im kaufmännischen und administrativen Bereich. Aus einem UML-Modell können sehr schnell voll funktionsfähige Anwendungen generiert werden, die neben einem grundlegenden Fachkonzept eine Datenhaltung und eine graphische Benutzungsoberfläche (***Graphical User Interface*** GUI) enthalten. Durch eine Vielzahl von Einstellungen im UML-Modell kann die Generierung im weiteren Software-Entwicklungsprozess individuell angepasst werden.

1 Einleitung

oTRIs möchte GO für die Erstellung der UML-Modelle nutzen, um unabhängig von Modellierungswerkzeugen von Fremdherstellern zu sein. Dazu muss jedoch eine Kommunikation zwischen GO und dem JANUS-Specifier, einem Werkzeug zur Modifikation von JANUS-spezifischen Attributen im UML-Modell, ermöglicht werden.

Die Kommunikation zwischen GO und JANUS-Specifier soll dabei auf der Basis von COM implementiert werden, da COM ein Standard unter dem Betriebssystem Windows ist.

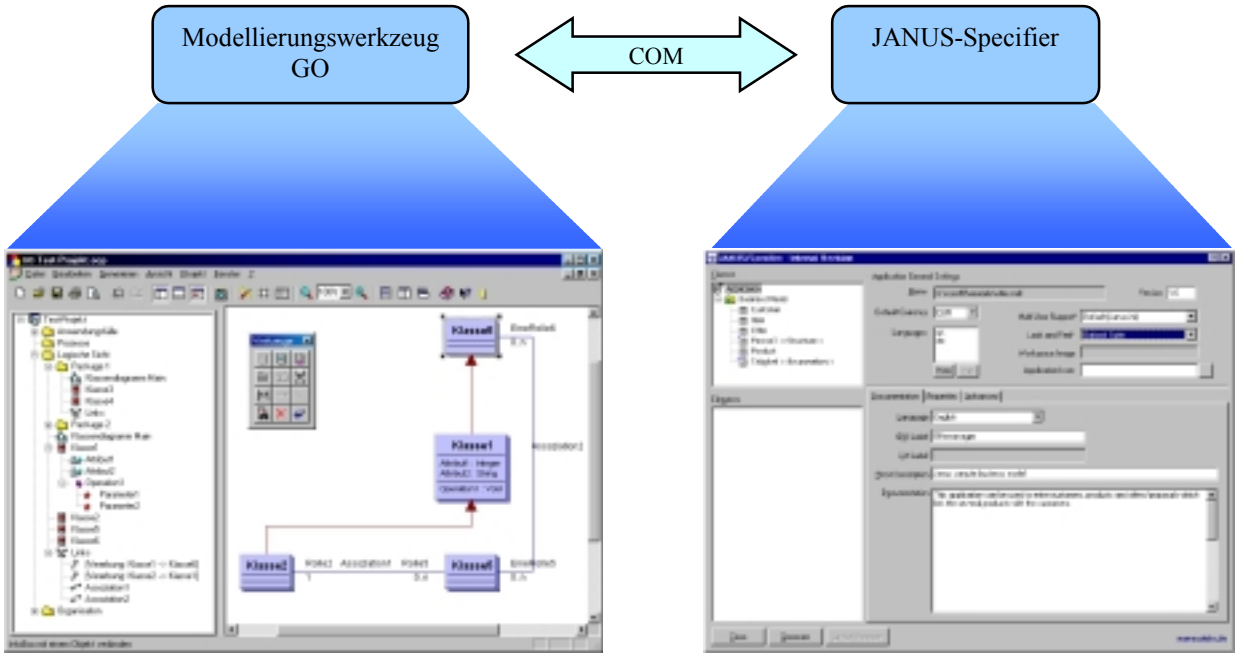

Abb. 1-2: Kommunikation zwischen GO und JANUS-Specifier

Zum Zeitpunkt der Anfertigung dieser Arbeit bestanden am Lehrstuhl kaum Erfahrungen in der Erstellung von COM-Anwendungen unter Visual C++, so dass neben der eigentlichen Problemlösung die Einarbeitung in diese Technik ein Bestandteil dieser Studienarbeit ist.

2 Zielbestimmung

Um in Zukunft das Software-Analysewerkzeug GO für den JANUS-Generator der Firma oTRIs nutzbar zu machen, ist es notwendig, die Kommunikation zwischen GO und dem JANUS-Specifier zu ermöglichen. Der JANUS-Specifier ist ein Werkzeug für den JANUS-Generator und ermöglicht es, ein OOA-Modell mit zusätzlichen für die Generierung notwendigen JANUS-Attributen zu spezifizieren.

Ziel dieser Studienarbeit ist es, GO zu einem Automatisierungsserver zu erweitern und damit die oben angesprochene Kommunikation zu ermöglichen.

Die für die Kommunikation mit dem JANUS-Specifier relevanten COM-Komponenten und COM-Schnittstellen müssen identifiziert und modelliert werden. Anschließend müssen die erkannten COM-Komponenten geeignet implementiert werden.

Bei der Identifikation der Komponenten und Schnittstellen ist darauf zu achten, dass die für den JANUS-Specifier notwendigen Informationen aus den GO-Modellen ausgelesen und bearbeitet werden können. Darüber hinaus sollen zusätzliche JANUS-spezifische Attribute in das Modell eingefügt werden können.

Durch die Konzentration auf die Kommunikation zwischen GO und dem JANUS-Specifier ergibt sich für die Anzahl der zu erstellenden COM-Komponenten und Schnittstellen eine Einschränkung. Es müssen nur die Elemente des Klassendiagramms ferngesteuert werden können. Alle anderen Diagramme von GO, UML- und nicht-UML, bleiben unberücksichtigt.

Bevor ich auf den hier zu erstellenden Automatisierungs-Server eingehen werde, möchte ich zunächst im Kapitel 1 die in dieser Studienarbeit verwendeten Techniken darstellen und dabei in das Komponentenmodell COM einführen und die Benutzung der ATL beschreiben.
Im Kapitel 4 werde ich dann die COM-Schnittstelle von GO vorstellen und detailliert auf die Vorgehensweise zum Hinzufügen der COM-Schnittstelle eingehen. Ferner werde ich die in dieser Studienarbeit erstellte Schnittstellenimplementierung beschreiben.

3 Verwendete Techniken

Für die Kommunikation zwischen GO und anderen Anwendungen – hier in erster Linie dem JANUS-Specifier – soll ein ***COM-Automatisierungs-Server*** erstellt werden.
Bei der Implementierung habe ich mich für die Verwendung der ***ATL*** entschieden, weil dadurch der Implementierungsaufwand im Vergleich zu einer konventionellen COM-Implementierung erheblich verringert werden konnte.

Zum besseren Verständnis der im Kapitel 4 verwendeten Begriffe und Programmiertechniken soll dieses Kapitel nun einen Einblick in COM und ATL vermitteln.

3.1 COM (Component Object Model)

Es sei darauf hingewiesen, dass eine umfassende Beschreibung und Analyse von Komponentenmodellen den Rahmen einer Studienarbeit sicherlich sprengen würde. Deswegen werde ich hier nur auf die für diese Studienarbeit wesentlichen Grundzüge des Komponentenmodells COM eingehen.
Weitergehende Informationen kann der Leser der im Anhang angegebenen Literatur entnehmen.

3.1.1 COM-Grundlagen

COM ist ein properitäres Komponentenmodell der Firma Microsoft, das sich zu einem Industriestandard auf Windows-Plattform entwickelt hat.

»COM ist inzwischen ein integraler Bestandteil der Windows-Betriebssysteme, auf dem viele wichtige Konzepte aufbauen (OLE, Automatisierung, ActiveX-Steuerelemente).« /Balzert 00/

Ziel dieses Komponentenmodells ist es, ein Standardverfahren anzubieten, mit dessen Hilfe Anwendungen unter dem Betriebssystem Windows miteinander kommunizieren können.

Dazu definiert COM einen binären Standard für die Interaktion von Softwareobjekten. Vorgegeben wird die Darstellung von Schnittstellen auf binärer Ebene.
»Ein Halbfabrikat bzw. eine ***Komponente*** (***componentware***) ist also ein abgeschlossener binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird.« /Balzert 96/

Insbesondere legt die COM-Spezifikation jedoch nicht fest, wie COM an eine bestimmte Programmiersprache angebunden wird. Aus diesem Grund sind COM-Komponenten programmiersprachenunabhängig.

»***Clients*** können von einer ***COM-Komponente*** Objekte erzeugen und diese ***COM-Objekte*** (Exemplar einer COM-Komponente) benutzen, indem sie einen Zeiger auf eine Schnittstelle der Komponente anfordern. Über diesen Zeiger kann der ***Client*** dann Operationen der Komponente aufrufen. Ein ***Client*** erhält aber nie direkten Zugriff auf das COM-Objekt.« /Balzert 00/

COM-Komponenten werden in der Literatur in der Regel als COM-Objekte bezeichnet. In dieser Studienarbeit verwende ich die Terminologie »COM-Objekt« für ein von einer COM-Komponente erzeugtes Objekt.

3.1.2 COM-Schnittstellen

Die Kommunikation mit einem COM-Objekt geschieht bei COM ausschließlich über ***Schnittstellen***, die von den COM-Komponenten implementiert werden. Eine Schnittstelle beschreibt demnach, wie von außerhalb auf die Komponente zugegriffen werden kann. Die Funktionalität der Komponente wird dabei gekapselt. Die ***Client***-Anwendung, die eine Komponente nutzen möchte, nimmt diese Komponente nur als »Black Box« wahr.

Dabei implementieren COM-Komponenten in der Regel nicht nur eine, sondern mehrere Schnittstellen.

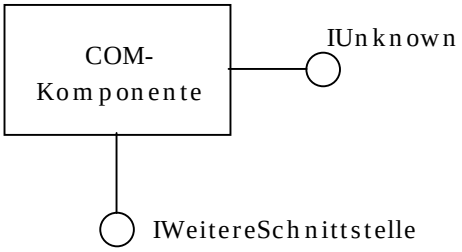

Abb. 3.1-1: Beispiel einer COM-Komponente mit zwei Schnittstellen

Zur Identifikation einer COM-Schnittstelle werden zwei ***Bezeichner*** verwendet.

- Der Programmierer benutzt für eine Schnittstelle einen textuellen Namen, der per Konvention mit dem Buchstaben »I« für ***Interface*** beginnt. Dieser Name muss weltweit ***nicht*** eindeutig sein.
- Als eindeutiger Bezeichner wird ein Schnittstellen-Identifizierer ***IID*** (***Interface Identifier***) festgelegt, der ein ***GUID*** (***Global Unique Identifier***) ist. Dieser Identifizierer ist eine in Raum und Zeit eindeutige 128-Bit-Zahl, die über BIOS-Seriennummer, Seriennummer der Netzwerkkarte, Datum und Uhrzeit gebildet wird. GUIDs können mit Hilfe des Programmes GUIDGEN.exe erzeugt werden.

So ist es möglich, dass zwei Komponenten-Hersteller, die unter Umständen für eine Schnittstelle den gleichen textuellen Bezeichner gewählt haben, trotzdem ihre Komponenten publizieren können, ohne dass dies zu Problemen führt, falls diese Komponenten auf demselben Rechner installiert werden.

COM unterstützt die einfache und mehrfache Vererbung von Schnittstellen.

Eine Schnittstelle, die von einer anderen Schnittstelle erbt, erhält alle Operationen dieser Ober-Schnittstelle hinzugefügt.

Eine Komponente, die eine Schnittstelle implementiert, die wiederum von einer oder mehreren anderen Schnittstellen erbt, muss also alle Operationen der Ober-Schnittstellen und die eigenen Operationen der Schnittstelle implementieren.

3.1.2.1 Die Schnittstelle IUnknown

Die Wurzelschnittstelle, von der jede weitere benutzerdefinierte Schnittstelle direkt oder indirekt erben muss, ist die in dem obigen Beispiel bereits erwähnte Schnittstelle **IUnknown** mit der IID {00000000-0000-C000-000000000046}.

Da die COM-Schnittstellen einer Komponente auch von dieser Komponente implementiert werden müssen, muss jede COM-Komponente IUnknown implementieren.

IUnknown enthält die drei Operationen QueryInterface, AddRef und Release.

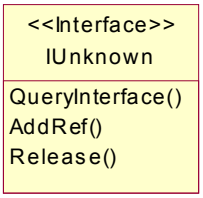

Abb. 3.1-2: Die Schnittstelle IUnknown

IUnknown spielt eine Schlüsselrolle für COM. Wenn ein ***Client*** eine Instanz einer COM-Komponente erzeugt, dann bekommt er eine Referenz auf die IUnknown-Schnittstelle des COM-Objektes übergeben.

Mit Hilfe von **QueryInterface** kann der ***Client*** dann weitere Schnittstellen des COM-Objektes anfordern. Dazu muss er lediglich die IID dieser Schnittstelle kennen. Bei erfolgreichem Aufruf von QueryInterface wird dem aufrufenden ***Client*** ein gültiger Zeiger auf die angeforderte Schnittstelle zurückgegeben. Im Fehlerfall wird ein NULL-Zeiger zurückgegeben. Im Erfolgsfall wird darüber hinaus noch implizit AddRef aufgerufen. Welche Bedeutung dies hat, wird klar, wenn man sich mit dem Lebenszyklus von COM-Objekten beschäftigt.

Eine zentrale Aufgabe von IUnknown ist die ***Referenzzählung***. »COM-Objekte regeln ihren Lebenszyklus über eine Referenzzählung (***reference counting***).« /Balzert 00/

Neben den Operationen der Schnittstelle IUnknown, namentlich QueryInterface, AddRef und Release, die jede COM-Komponente implementieren muss, besitzt jede COM-Komponente noch ein Datenelement, das als ***Referenzzähler*** fungiert und in dem die Anzahl der »Benutzer« dieser Komponente gehalten wird.

»Jedesmal wenn das Komponentenprogramm einen neuen Schnittstellenzeiger (z.B. über QueryInterface) zurückgibt, ruft es **AddRef** auf«, um den Referenzzähler des zugehörigen COM-Objektes zu inkrementieren. »Sobald das ***Client***-Programm den Zeiger nicht mehr benötigt, ruft es **Release** auf.« /Kruglinski, Wingo, Sheperd 98/

Sollte beim Aufrufen von Release der Referenzzähler eines COM-Objektes den Wert 0 erreichen, veranlasst das Objekt seine eigene Entfernung aus dem Speicher.

3.1.2.2 IDL (Interface Definition Language)

Da COM einen programmiersprachenunabhängigen Standard definiert, muss die Deklaration der Schnittstellen einer Komponente ebenfalls programmiersprachenunabhängig erfolgen. Dazu dient die Schnittstellenbeschreibungssprache ***IDL*** (***Interface Definition Language***).

Eine Schnittstellen-Definition in IDL besteht aus zwei Teilen.
Dem Schlüsselwort `interface` wird ein sog. ***interface-header*** vorangestellt, in den eine Liste von speziellen Schnittstellen-Attributen eingefügt werden kann.
Der Schnittstellenrumpf, der auch als ***interface-body*** bezeichnet wird, enthält die eigentliche Schnittstellen-Deklaration mit den nach außen zur Verfügung zu stellenden Operationen.

```
[
    <interface-header>
]
interface <interface-name> [ : <base-interface> ]
{
    <interface-body>
}
```

Als Beispiel sei hier die IDL-Deklaration der Schnittstelle `IUnknown` aufgeführt:

```
[
    local,
    object,
    uuid(00000000-0000-0000-C000-000000000046),
    pointer_default(unique)
]
interface IUnknown
{
    typedef [unique] IUnknown *LPUNKNOWN;

    HRESULT QueryInterface ([in] REFIID riid, [out, iid_is(riid)] void** ppvObject);
    ULONG AddRef();
    ULONG Release();
}
```

Um eine COM-Schnittstelle zu definieren, muss der ***interface-header*** das Attribut `object` enthalten. Außerdem muss über das Attribut ***uuid*** der IID (GUID) der Schnittstelle angegeben werden.

Hinter dem Schlüsselwort `interface` wird der textuelle Bezeichner für die Schnittstelle definiert. Nachfolgend können hinter einem Doppelpunkt Basis-Schnittstellen angegeben werden, von denen die Schnittstelle erben soll. Mehrfach-Schnittstellenvererbung ist möglich.
Schnittstellenvererbung bedeutet dabei, dass eine Komponente, die die deklarierte Schnittstelle implementiert, auch die zugehörigen Basis-Schnittstellen implementieren muss. Deklarierte Operationen der Basis-Schnittstellen stehen damit auch in der Kind-Schnittstelle zur Verfügung, ohne dass sie explizit deklariert werden müssen.

Im Schnittstellenrumpf werden die Operationen deklariert.
Bei COM besitzt jede Operation üblicherweise einen ***HRESULT***-Wert als Rückgabeparameter, der über die erfolgreiche oder nichterfolgreiche Ausführung der Operation informiert.

Nachfolgend seien einige der möglichen Standard-Definitionen für HRESULT-Werte aufgeführt.

HRESULT	*Bedeutung*
S_OK	Die Operation konnte erfolgreich beendet werden
E_NOINTERFACE	Die QueryInterface-Operation konnte die angegebene Schnittstelle nicht finden. Die Schnittstelle wird nicht unterstützt.
E_NOTIMPL	Die Operation enthält keine Implementierung.
E_FAIL	Ein nicht näher spezifizierter Fehler ist aufgetreten.
E_POINTER	Ein Zeiger ist nicht gültig.
E_INVALIDARG	Ein oder mehrere Parameter sind ungültig.
E_UNEXPECTED	Ein katastrophaler Fehler ist aufgetreten.
E_ABORT	Die Ausführung der Operation wurde abgebrochen.

Tabelle 3.1-1: HRESULT-Werte und ihre Bedeutung

Parameter einer Operation werden mit Richtungsattributen gekennzeichnet.
Ein ***Eingabeparameter*** erhält das ***Richtungsattribut*** **in**, ein ***Ausgabeparameter*** das Attribut **out**. Ein Parameter, der sowohl als Eingabe-, als auch als Ausgabeparameter genutzt werden soll, muss beide Richtungsattribute erhalten. Ein ***Rückgabeparameter*** wird zusätzlich zu dem ***out***-Attribut noch mit einem **retval**-Attribut gekennzeichnet.
Alle Ausgabeparameter werden als Zeiger mit einem Sternchen gekennzeichnet.

```
[
    object,
    uuid(...),
]
interface IBeispiel : IUnknown
{
    HRESULT Beep();
    HRESULT SetProperty ([in] long newVal);              // long-Wert als Eingabeparameter
    HRESULT GetProperty ([out, retval] long *pVal);      // Ausgabe-/Rückgabeparameter vom
                                                         // Typ long
}
```

Sollen Komponenten miteinander über Prozessgrenzen hinweg kommunizieren, dann werden für die Kommunikation zwischen ***Client*** und ***Server Stellvertreter*** (***proxies***) und ***Stummel*** (***stubs***) notwendig, da ***Client*** und Komponente eigene Adressräume aufweisen.

/Balzert 00/ beschreibt dies folgendermaßen:
»Stellvertreter laufen im Adressraum des ***Client*** und vertreten sozusagen die eigentliche Komponente. Stummel laufen im Adressraum des ***Servers*** und stellen den Kontakt zwischen Stellvertreter und der eigentlichen Komponente her.«

Stellvertreter und Stummel können aus der IDL-Beschreibung einer Komponente mit ihren Schnittstellen generiert werden. Für die Programmiersprache C++ kann dazu der ***Microsoft IDL-Compiler*** (MIDL-***Compiler***) verwendet werden.

Üblicherweise werden außerhalb des COM-Kontextes Stellvertreter als Stummel (***stubs***) und Stummel als Skelette (***skeletons***) bezeichnet.

3.1.2.3 Typbibliotheken

Typbibliotheken dienen der Selbstbeschreibungsfähigkeit von COM-Komponenten. Sie ermöglichen die Introspektion.

Will man eine Typbibliothek erstellen, muss dies in der IDL-Datei mit dem Schlüsselwort **library** festgelegt werden. Der MIDL-***Compiler*** erzeugt dann eine Typbibliothek mit der Dateiendung »tlb«.

Um an die Typinformationen einer gewünschten Komponente zu gelangen, kann ein ***Client*** eine Anfrage an die ***Registry*** stellen. Dazu benötigt er lediglich den CLSID[1] dieser Komponente.

Beispiel einer IDL-Datei, in der eine Typbibliothek spezifiziert wird /Balzert 00/:

```
import “unknown.idl”

[object, uuid(...), oleautomation]
interface IMyInterface : IUnknown
{
    HRESULT Operation1();
}

[uuid(...), helpstring(“MyComponent Typbibliothek”), version(1.0)]
library Component
{
    importlib”stdole32.tlb”;
    interface IMyInterface;

    [uuid(...)]
    coclass MyComponent
    {
        interface IMyInterface;
    }
};
```

[1] Der CLSID (***class identifier***) wird im Kapitel 3.1.3 erläutert

3.1.3 COM-Komponenten

Unter einer ***COM-Komponente*** versteht man einen binären Softwarebaustein, der eine oder mehrere Schnittstellen implementieren kann. In der Literatur werden COM-Komponenten häufig als COM-Objekte bezeichnet. Exemplare einer COM-Komponente werden jedoch ebenfalls als COM-Objekte bezeichnet. Aus diesem Grund wird in dieser Studienarbeit durchgehend der Begriff COM-Komponente verwendet.

3.1.3.1 Registrierung

Eine COM-Komponente besitzt wie eine COM-Schnittstelle zwei unterschiedliche Bezeichner. Der eine Bezeichner ist dabei der textuelle Komponenten-Name, der andere eine ***GUID***. In der COM-Terminologie wird eine COM-Komponente auch als COM-Klasse bezeichnet. Daraus folgt, dass die GUID im Zusammenhang mit Komponenten als Klassen-Bezeichner (***Class Identifier***) oder ***CLSID*** bezeichnet werden.

Bevor ein ***Client*** eine COM-Komponente benutzen kann, muss diese Komponente in der ***Windows Registry*** registriert werden.

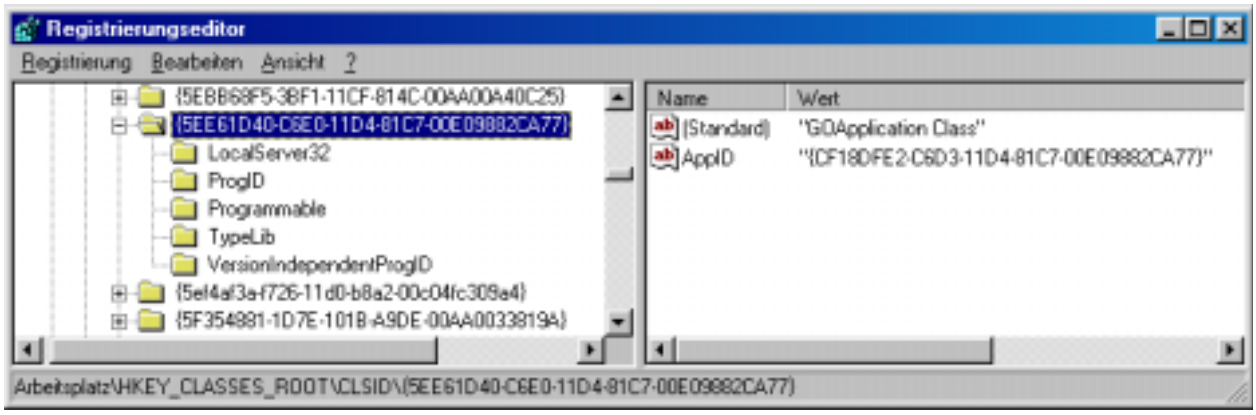

Abb. 3.1-3: Registrierungseditor – Eintrag der Komponente GOApplication

Bei der Registrierung wird ein neuer Schlüssel mit der CLSID der Komponenten im Zweig `HKEY_CLASSES_ROOT\CLSID` erzeugt. Als Wert dieses Schlüssels wird der textuelle Name der Komponente eingetragen. In einem Unterschlüssel (`InprocServer32` bzw. `LocalServer32`) wird der Dateiname und der Pfad zu dem Modul gespeichert, das die Komponente implementiert.
Da die CLSID eine eindeutige Identifizierung der Komponente ermöglicht, kann man durch Suchen einer CLSID in der ***Registry*** immer die richtige Komponente finden, sofern sie registriert wurde.
In der ***Registry*** wird jedoch zusätzlich unter dem Zweig `HKEY_CLASSES_ROOT` der textuelle Name der Komponenten als Schlüssel eingetragen. Bei Komponenten mit gleichen Namen überschreibt die zuletzt registrierte Komponente den Schlüssel der zuvor registrierten Komponente. Dies bedeutet, dass eine eindeutige Identifikation immer nur über die CLSID erfolgen darf.

3.1.3.2 Erzeugen von Objekten

Komponenten werden nur als Binärcode ausgeliefert. Außerdem können ***Client*** und ***Server*** in verschiedenen Programmiersprachen implementiert worden sein. Daher kann der ***Client*** nicht wissen, wie Objekte einer Komponente zu erzeugen sind.

Will der ***Client*** nun ein COM-Objekt erzeugen, ruft er die Operation **CoCreateInstance** der COM-Bibliothek auf. Als Parameter werden die CLSID der gewünschten COM-Komponente und eine IID auf die gewünschte COM-Schnittstelle angegeben.
CoCreateInstance sucht zunächst in der ***Registry*** nach der CLSID der Komponenten und bestimmt damit den Aufenthaltsort des Moduls, das die Komponente beinhaltet.
Nachdem das Modul geladen wurde, wird eine Instanz der COM-Komponente erzeugt und ein Zeiger auf die angeforderte Schnittstelle zurückgegeben.

Alternativ kann der ***Client*** das Suchen in der Registry und das Erzeugen von Objekten auch getrennt aufrufen. Dies bietet sich insbesondere an, wenn mehrere Objekte einer Komponente erzeugt werden sollen. Dazu dienen die Operationen **CoGetClassObject** und **IClassFactory::CreateInstance**, die in CoCreateInstance subsummiert wurden.

3.1.3.3 Die Schnittstelle IClassFactory

Die Benennung ist etwas missverständlich. Suggeriert »***Class Factory***« doch, dass eine Klassen-Fabrik gemeint wäre. Genau genommen handelt es sich jedoch um die Schnittstelle auf eine Objekt-Fabrik. /Kruglinski, Wingo, Sheperd 98/

Über die Schnittstelle **IClassFactory** können COM-Objekte, also Instanzen einer COM-Komponente, erzeugt werden.

Eine stark vereinfachte Definition der IClassFactory-Schnittstelle sei im Folgenden dargestellt:

```
[object, uuid(00000001-0000-0000-C000-000000000046)]
interface IClassFactory : IUnknown
{
    HRESULT CreateInstance(    [in] IUnknown *pUnkOuter,
                               [in] REFIID riid,
                               [out, iid_is(riid)] void** ppvObject);
}
```

Durch den Aufruf von CreateInstance wird eine Instanz der Komponente erzeugt. Der Parameter pUnkOuter wird nur für eine Aggregation benötigt, sonst wird ein NULL-Zeiger übergeben. Das Konzept von Einschließung und Aggregation wird in dieser Studienarbeit nicht behandelt. Der interessierte Leser möge diese Wiederverwendungs-Mechanismen der im Anhang angegebenen Literatur entnehmen.
Als zweiter Parameter wird die angeforderte Schnittstelle auf das erzeugte COM-Objekt, in der Regel die Schnittstelle IUnknown als IID angegeben. Der dritte Parameter ist ein Rückgabeparameter, der den angeforderten Schnittstellenzeiger anschließend enthält.

3.1.4 Module

»COM-Komponenten werden als ***Module***, d.h. als DLLs oder Anwendungen (.exe) ausgeliefert.« /Balzert 00/
Module enthalten den binären Code von einer oder mehreren COM-Komponenten und werden bei Bedarf vom Betriebssystem in den Arbeitsspeicher geladen.

Ein ***Server*** stellt die Dienste der enthaltenen Komponenten seinen ***Clients*** zur Verfügung.
Je nachdem, in welchem Prozessraum sich ein Modul relativ zum ***Client*** befindet, unterscheidet man verschiedene Arten von Servern.

- ***Prozessinterner Server (in-process server)***
  Eine COM-Komponente, die in einer DLL ausgeliefert wird, wird in den Prozessraum des aufrufenden ***Client*** geladen (Abb. 3.1-4). Da eine DLL allein kein lauffähiges Programm darstellt, wird immer ein ***Client*** benötigt, um den Code des Moduls auszuführen.
  Der Aufruf einer Schnittstellenoperation entspricht hierbei einem direkten Aufruf einer Operation der COM-Komponente. Ein Verpacken (***marshalling***) der Aufrufparameter ist nicht notwendig, da alle Referenzen innerhalb des prozessinternen Raumes gültig bleiben.
  Dies bedeutet insbesondere auch, dass die Kommunikation zwischen ***Client*** und ***Server*** sehr schnell und einfach erfolgen kann.

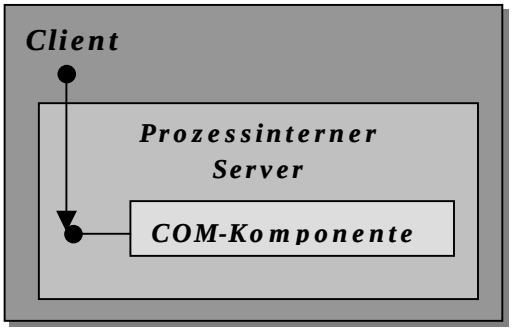

Abb. 3.1-4: Prozessinterner Server /Westhoff 00/

- ***Lokaler Server (local server)***
  Ein lokaler ***Server*** wird zwar auf demselben Rechner wie der ***Client*** ausgeführt, jedoch in einem anderen Prozess (Abb. 3.1-5). Das Modul ist dabei eine eigenständig lauffähige Anwendung, d.h. eine .exe-Datei. Aufgrund der unterschiedlichen Prozessräume müssen die Parameter von Operationen erst verpackt werden (***marshalling***). Das Betriebssystem muss darüber hinaus die Daten zwischen den Prozessen hin und her kopieren. Der Zugriff auf lokale ***Server*** ist daher bedeutend langsamer als auf einen prozessinternen ***Server***.

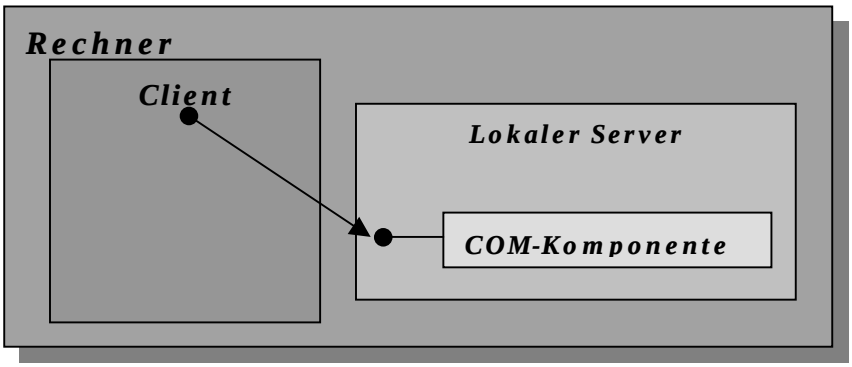

Abb. 3.1-5: Lokaler Server /Westhoff 00/

- ***Entfernter Server***
  Ein entfernter ***Server*** befindet sich auf einem anderen Rechner als der ***Client***. Da ***Client*** und ***Server*** auch hier in unterschiedlichen Prozessräumen laufen, ist für die Kommunikation das Verpacken (***marshalling***) von Parametern notwendig. Als Module können sowohl DLLs, als auch eigenständige Anwendungen verwendet werden. Für den Zugriff auf COM-Objekte auf entfernten Computersystemen wird DCOM (***Distributed*** COM) verwendet, auf das hier nicht weiter eingegangen werden soll.

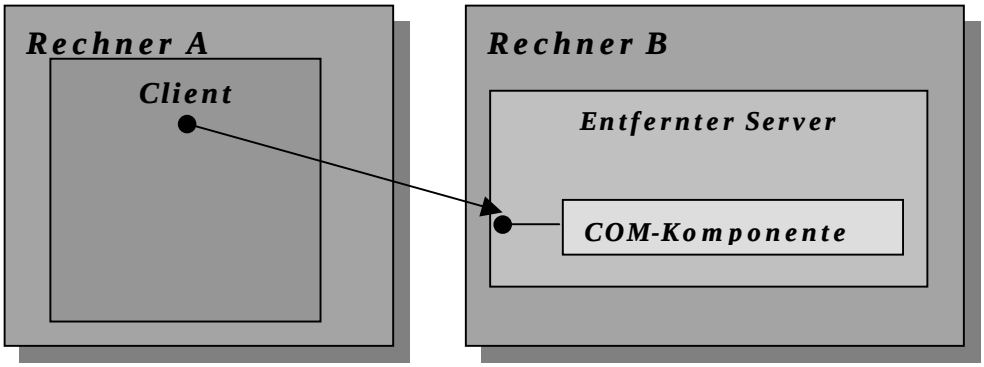

Abb. 3.1-6: Entfernter Server /Westhoff 00/

Die Unterscheidung der unterschiedlichen Server erfolgt in der Literatur keineswegs einheitlich. So unterscheiden einige Autoren auch lediglich prozessinterne Server (***in-process server***) von prozessexternen Servern (***out-process server***).

3.1.5 Automatisierungsserver

Ein auf COM basierendes Konzept ist die ***Automatisierung***. Dieser Begriff ist von Microsoft für Anwendungen geprägt worden, die »***ferngesteuert***« werden können. Hierbei soll auf die Funktionalität einer Anwendung möglichst einfach von außen zugegriffen werden können. Die Anwendung kann somit von anderen Anwendungen, aber insbesondere auch durch Scripte bedient werden.

3.1.5.1 Die Schnittstelle IDispatch

Automatisierungs-Komponenten müssen die Schnittstelle `IDispatch` implementieren, wenn sie auch durch Script-Sprachen (z.B. ***Visual Basic for Applications*** VBA) benutzt werden sollen.

```
interface IDispatch : IUnknown
{
    HRESULT GetTypeInfoCount([out] UINT *pctinfo);
    HRESULT GetTypeInfo(  [in] UINT iTInfo,
                          [in] LCID lcid,
                          [out] ITypeInfo **ppTInfo);
    HRESULT GetIDsOfNames([in] REFIID riid,
                          [in, size_is(cNames)] LPOLESTR *rgszNames,
                          [in] UINT cNames,
                          [in] LCID lcid,
                          [out, size_is(cNames)] DISPID *rgDispID);
    HRESULT Invoke(   [in] DISPID dispIdMember,
                      [in] REFIID riid,
                      [in] LCID lcid,
                      [in] WORD wFlags,
                      [in, out] DISPARAMS *pDispParams,
                      [out] VARIANT *pVarResult,
                      [out] EXCEPINFO *pExcepInfo,
                      [out] UINT *puArgErr);
}
```

In COM besteht eine Schnittstelle aus einer ***Tabelle virtueller Funktionen*** (***virtual function table***, ***VTBL***). In dieser Tabelle werden Einsprungpunkte für die Operationen der Schnittstelle festgelegt. Eine Operation wird in der binären Darstellung der Schnittstelle nicht über einen textuellen Bezeichner identifiziert, sondern ganz einfach über die Position innerhalb der Tabelle.

`IDispatch` erlaubt nun die Verwendung symbolischer Namen für Operationen und Eigenschaften. Eine Eigenschaft wird in der COM-Schnittstelle dabei eigentlich über zwei Operationen (***propget*** und ***propput***) spezifiziert.
Jeder Eigenschaft und jeder Operation der Komponente, die über `IDispatch` angesprochen werden sollen, wird eine eindeutige Nummer zugeordnet, die sogenannte ***dispatch***-ID (***DISPID***).
Zur Übersetzungszeit der ***Client***-Anwendung ist die Zuordnung der symbolischen Namen zu dieser DISPID unter Umständen noch nicht bekannt. Die Operation *GetIDsOfNames* ermöglicht es, zu einem Operationsnamen die zugehörige DISPID zurückzuliefern.

Mit *Invoke* kann dann unter Angabe der DISPID die gewünschte Operation aufgerufen werden.

3.1.5.2 Automatisierungs-Datentypen

Eine Schnittstelle muss für die Automatisierung bestimmte Eigenschaften erfüllen. Sie muss von IDispatch oder IUnknown erben. Weiterhin wird sie im ***interface-header*** mit dem Attribut **oleautomation** versehen. Dieses Attribut bestimmt, dass für das Verpacken (***marshalling***) von Parametern der »Automatisierungs-***marshaller***« benutzt werden kann.
Aus diesem Grund bestehen auch Einschränkungen hinsichtlich der verwendeten Datentypen.
Alle Parametertypen (Eingabe-, Ausgabe-, Rückgabe-Parameter) müssen automatisierungskompatibel sein.

Welche Typen unterstützt werden, kann der nachfolgenden Tabelle /MSDN 00/ entnommen werden.

Type	***Beschreibung***
boolean (**VARIANT_BOOL**)	Kann die Werte VARIANT_TRUE oder VARIANT_FALSE annehmen
unsigned char	Vorzeichenloses 8-bit-Datenelement
Double	64-Bit IEEE Fließkomma-Zahl
Float	32-Bit IEEE Fließkomma-Zahl
Int	Ganzzahl mit systemabhängiger Größe. Auf 32-Bit-Betriebssystemen vorzeichenbehaftete 32-Bit Ganzzahl.
Long	Vorzeichenbehaftete 32-Bit-Ganzzahl.
Short	Vorzeichenbehaftete 16-Bit-Ganzzahl.
BSTR (*Basic String*)	Zeichenkette mit vorangestellter Längenangabe. BSTRs enthalten 2-Byte Breitzeichen (Unicode).
CY	8-Byte-Festkomma-Zahl (ehemals CURRENCY).
DATE	64-Bit Fließkomma-Bruchzahl der Tage seit dem 30. Dezember 1899.
SCODE	Eingebauter Fehlertype, entspricht ***HRESULT***.
Enum	Vorzeichenbehaftete Ganzzahl mit systemabhängiger Größe.
IDispatch *	Zeiger auf eine IDispatch-Schnittstelle.
IUnknown *	Zeiger auf eine Schnittstelle, die nicht von IDispatch erbt. Jede COM-Schnittstelle kann durch ihre IUnknown-Schnittstelle repräsentiert werden!

Tabelle 3.1-2: Automatisierungskompatible Datentypen

In obiger Tabelle sind die in dieser Studienarbeit verwendeten Parametertypen ***fett*** dargestellt.

Bei Zeichenketten ist darauf zu achten, dass der reservierte Speicherbereich immer mit der Längenangabe beginnt, damit die Speicherfreigabe auf dem ***Client*** auch einwandfrei funktionieren kann.
Zur Reservierung und Freigabe von BSTR-Objekten existieren die COM-Operationen SysAllocString und SysFreeString.

Da IDispatch-Zeiger als Parameter fungieren können, können auch Zeiger auf benutzerdefinierte Schnittstellen als Parameter benutzt werden, die von IDispatch abgeleitet worden sind. Dies wird in den GO-Schnittstellen häufig verwendet.

3.1.5.3 Dual-Interface

Typischerweise werden bei der Implementierung von Automatisierungsservern sogenannte ***Dual-Interfaces*** benutzt. Sie erlauben den Zugriff auf Eigenschaften und Operationen einer Komponente sowohl über die Schnittstelle IDispatch, als auch direkt über die VTBL (wie bei jedem anderen COM-Interface).
In der IDL-Spezifikation werden ***Dual-Interfaces*** durch das Attribut »***dual***« gekennzeichnet.

Beispiel für ein IDL-Spezifikation eines ***Dual-Interface*** mit der Eigenschaft »Name«:

```
[
    uuid(...),
    oleautomation,
    dual
]
interface InterfaceName : IDispatch
{
    [propget, id(1)] HRESULT Name([out, retval] BSTR *pVal);
    [propput, id(1)] HRESULT Name([in] BSTR newVal);
};
```

Eine direkte VTBL-Bindung unterstützt eine hohe Geschwindigkeit. Für die Bindung über IDispatch spricht eine hohe Flexibilität. Ein ***Dual-Interface*** unterstützt beides. Aus diesem Grund sollten ***Dual-Interfaces*** eingesetzt werden, wenn immer dies möglich ist.

3.2 ATL (Active Template Library)

Ebenso wie die Einführung in COM kann auch eine Einführung in die ATL im Rahmen einer Studienarbeit nicht vollständig sein. In diesem Kapitel werde ich die grundlegenden Techniken der ATL vorstellen, um einen COM-***Server*** zu implementieren. Zu erwähnen sei jedoch, dass die ATL für ein weitaus größeres Feld von Anwendungsbereichen genutzt werden kann.

3.2.1 ATL-Grundlagen

Ursprünglich stand die Abkürzung ATL für ***ActiveX Template Library***. Dieser Benennung kann man entnehmen, dass die ATL eine umfangreiche Unterstützung zum Erstellen von ActiveX-Controls bietet.
Die ATL hat sich im Laufe der Entwicklung jedoch zu einer viel mächtigeren Bibliothek entwickelt, so dass man die ATL auch zur Unterstützung von COM-Anwendungen – ohne ActiveX-Controls – verwenden kann.

Genau genommen ist die ***ATL*** (***Active Template Library***) eine C++-Klassenbibliothek, ähnlich der STL (***Standard Template Library***).
Die STL hat einen einheitlichen Standard für die Verwendung von Iteratoren, Containern und weiteren Hilfsklassen etabliert. Ebenso definiert die ATL einen Standard für die Implementierung von COM-Anwendungen.

Die ATL bietet daher ***Wrapper***-Klassen zur Verwaltung von Datentypen, wie Schnittstellen-Referenzen, VARIANT und BSTR. Sie stellt darüber hinaus Klassen zur Verfügung, die Implementierungen der COM-Basis-Schnittstellen wie IUnknown, IClassFactory und IDispatch anbieten.
Ebenso sind in dieser Bibliothek Klassen enthalten, die Management-Funktionen für einen COM-Server übernehmen. Dazu gehören die Selbstregistrierung von COM-Objekten und das Lebensdauer-Management des Servers. /Rector, Sells 00/

Der größte Vorteil der ATL besteht darin, dass man sich als Programmierer eines COM-Servers nicht um die immer wiederkehrende Implementierung der Schnittstellen IUnknown bzw. IDispatch kümmern muss. Dies bedeutet insbesondere, dass die Implementierung der Referenzzählung für COM-Objekte vollständig der ATL überlassen werden kann. Ebenso kann auf vorgefertigte Objekt-Fabriken zurückgegriffen werden.

Im folgenden Unterkapitel werde ich auf die in dieser Studienarbeit verwendeten ATL-***Template***-Klassen und die in der ATL definierten Makros eingehen. Ein grundlegendes Verständnis für die ebenfalls nachfolgend beschriebenen ATL-Vererbungsstrukturen ist notwendig, um anschließend die Implementierung der COM-Schnittstellen des GO nachvollziehen zu können.

3.2.2 Verwendete Template-Klassen und Makros

Um die ATL-Template-Klassen und Makros in einer Anwendung benutzen zu können, müssen die folgenden ***Header***-Dateien in die Datei »stdafx.h« eingefügt werden.

```
#include <atlbase.h>
#include <atlcom.h>
```

3.2.2.1 CComModule

Wie im Kapitel 3.1.4 bereits erwähnt, werden COM-Komponenten als Module ausgeliefert. Um aus einer beliebigen Anwendung aber ein COM-Modul zu machen, muss diese Anwendung den ***Clients*** den Zugriff auf die enthaltenen COM-Komponenten ermöglichen.

`CComModule` ist eine Klasse, die ein COM-***Server***-Modul implementiert und den Zugriff auf die enthaltenen Komponenten erlaubt. `CComModule` kann dabei sowohl für prozessinterne ***Server***, als auch für lokale ***Server*** eingesetzt werden.

In der Anwendung wird üblicherweise eine globale Instanz einer von `CComModule` abgeleiteten Klasse eingefügt. Diese Instanz erhält per Konvention den Bezeichner `_Module`.

Die Instanz `_Module` benutzt eine ***Objektliste***[1] (***object map***), die CLSIDs auf Klassen abbildet, die die entsprechenden Komponenten implementieren.

Diese Liste wird in der Anwendungsklasse über spezielle ATL-Makros definiert.

```
BEGIN_OBJECT_MAP(x)
    OBJECT_ENTRY(clsid, class)
END_OBJECT_MAP()
```

Der Parameter `x` enthält den Variablennamen für die Liste. Zwischen dem einleitenden Makro `BEGIN_OBJECT_MAP(x)` und dem abschließenden Makro `END_OBJECT_MAP()` wird für jede COM-Komponente ein Makroeintrag `OBJECT_ENTRY(clsid, class)` vorgenommen. Dabei entspricht der Parameter `clsid` der GUID der zugehörigen COM-Komponente, die in die Objektliste eingetragen werden soll. Über den Parameter `class` wird angegeben, in welcher Klasse diese Komponente implementiert wurde.

Nachfolgend sollen einige wichtige Operationen von CComModule beschrieben werden.

- ***Init***
  Bsp.: `_Module.Init(ObjectMap, AfxGetInstanceHandle());`

  Diese Funktion dient dazu, `_Module` mit der Objektliste und der Instanz der Anwendung (DLLMain oder WinMain) zu initialisieren.

[1] Objektliste: In der COM-Terminologie wird die COM-Komponente als COM-Objekt bezeichnet. Genauer genommen handelt es sich hier um eine Komponentenliste.

- ***UpdateRegistryFromResource bzw. UpdateRegistryFromResourceD***
  Bsp.: `_Module.UpdateRegistryFromResource(IDR_GO, TRUE);`

  Diese Operation führt ein ***Windows Registry-Script*** aus. Der erste Parameter der Funktion gibt die ***Resource***-ID an, unter der das Script in der zu der Anwendung gehörenden Resourcen-Datei zu finden ist. Besitzt der zweite Parameter den Wahrheitswert TRUE, dann wird die Anwendung in der ***Registry*** »registriert«, beim Wahrheitswert FALSE wird sie aus der ***Registry*** wieder ausgetragen.

- ***RegisterServer***
  Bsp.: `_Module.RegisterServer(TRUE);`

  Bei einem Aufruf von RegisterServer werden alle COM-Komponenten, die in der Objektliste eingetragen worden sind, in der ***Windows Registry*** eingetragen. Bei Bedarf kann auch eine einzelne COM-Komponente registriert werden, indem die zugehörige CLSID als Parameter übergeben wird.
  Der Parameter, der im obigen Beispiel mit TRUE angenommen wurde, sorgt dafür, dass die Typbibliothek-Informationen ebenfalls in der ***Registry*** aktualisiert werden.

- ***UnregisterServer***
  Bsp.: `_Module.UnregisterServer(TRUE);`

  Macht die Einträge von RegisterServer wieder rückgängig.

- ***RegisterClassObjects***
  Bsp.:
  ```
  hRes = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER,
      REGCLS_MULTIPLEUSE);
  ```

  Eine Klasse, welche die Schnittstelle `IClassFactory` implementiert, wird unter COM auch als ***Class Object*** bezeichnet. Die Operation `RegisterClassObjects` registriert die Objekt-Fabrik für die aktuelle Objektliste.

- ***RevokeClassObjects***
  Bsp.: `_Module.RevokeClassObjects();`

  Nimmt die Registrierung von `RegisterClassObjects` wieder zurück.

- ***Term***
  Bsp.: `_Module.Term();`

  Gibt alle Daten von `_Module` wieder frei.

Wie aus den obigen Funktionen ersichtlich, übernimmt `CComModule` diverse Management-Funktionen für den COM-Server, wie z.B. die Selbstregistrierung von COM-Komponenten in der ***Windows Registry***.

3.2.2.2 Registry-Scripts

Wie bei der Operation `UpdateRegistryFromResource` der Klasse `CComModule` bereits angedeutet, werden für die Registrierung von COM-Komponenten in der ***Windows Registry*** sogenannte ***Registry-Scripts*** ausgeführt.

Diese Skript-Dateien werden üblicherweise nicht von Hand erstellt, sondern über sog. ATL-***Wizards*** bei der Erstellung einer COM-Komponente aus der Programmierumgebung heraus generiert.
Dabei wird für jede COM-Komponente eine ***Registry-Script***-Datei erzeugt.
Nachfolgend sei beispielhaft die Skript-Datei »GoApplication.rgs« dargestellt.

```
HKCR
{
    GO.GOApplication.1 = s 'GOApplication Class'
    {
        CLSID = s '{5EE61D40-C6E0-11D4-81C7-00E09882CA77}'
    }
    GO.GOApplication = s 'GOApplication Class'
    {
        CLSID = s '{5EE61D40-C6E0-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOApplication.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5EE61D40-C6E0-11D4-81C7-00E09882CA77} = s 'GOApplication Class'
        {
            ProgID = s 'GO.GOApplication.1'
            VersionIndependentProgID = s 'GO.GOApplication'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

Die meisten Zeilen dieser Datei entsprechen direkt den Einträgen in der ***Registry***.
Registry-Scripts werden sowohl für die Registrierung von COM-Komponenten, als auch zum Austragen der Informationen aus der ***Registry*** verwendet.
Standardmäßig werden bei der Registrierung alle Schlüssel in der ***Registry*** hinzugefügt, unabhängig davon, welche Schlüssel eventuell vorher schon existiert haben.
Dieses Standardverhalten kann durch Hinzufügen des Schlüsselwortes **`ForceRemove`** verändert werden. Der Eintrag, der mit `ForceRemove` gekennzeichnet worden ist, wird vor der erneuten Registrierung komplett (d.h. inklusive aller evtl. vorhandenen Unterschlüssel) gelöscht.
Beim Austragen aus der ***Registry*** werden standardmäßig alle aufgeführten Schlüssel (inklusive der enthaltenen Unterschlüssel) entfernt. Dieses Verhalten ist insbesondere für den Eintrag CLSID gefährlich, da bei der gewünschten Löschung ***einer*** bestimmten Komponente ***alle*** in der ***Registry*** eingetragenen Komponenten ausgetragen würden.
Um dies zu verhindern, existiert das Schlüsselwort **`NoRemove`**. Ein Eintrag, der mit diesem Schlüsselwort gekennzeichnet wurde, wird grundsätzlich nicht aus der ***Registry*** gelöscht.

Die ***Registry-Script***-Dateien werden in die ***Ressourc***e-Datei der Anwendung eingebunden und erhalten eine ***Resource***-ID, über die sie angesprochen werden können (siehe `CComModule::UpdateRegistryFromResource`).

Jede COM-Komponente wird durch eine Klasse implementiert. In dieser Klasse wird mit Hilfe des Makros **`DECLARE_REGISTRY_RESOURCEID`** die zugehörige ***Resource***-ID festgelegt.

3.2.2.3 CComObjectRootEx

CComObjectRootEx ist eine ***Template***-Klasse. Bei ***Template***-Klassen oder sog. Schablonen handelt es sich um parametrisierte Klassen. Einer parametrisierten Klasse kann als formaler Parameter ein Typ angegeben werden.

```
template< class ThreadModel >
class CComObjectRootEx : public CComObjectRootBase
```

Jede Klasse, die einen COM-Server implementiert, muss von CComObjectRootEx (oder CComObjectRoot) erben.

Diese Klasse sorgt für das Management der Objekt-Referenzzählung.
Dazu enthält diese Klasse ein Datenelement namens m_dwRef. Für ein COM-Objekt, eine Instanz einer COM-Komponente, enthält dieses Datenelement die Anzahl der Referenzen auf dieses Objekt.

Achtung: Obwohl CComObjectRootEx alle Funktionalität für IUnknown kapselt, implementiert sie diese Schnittstelle jedoch nicht (s. Kap. 3.2.2.5 und 3.2.2.6).

Auf den Wiederverwendungsmechanismus Aggregation, bei dem die Referenzzählung geringfügig anders arbeitet, möchte ich in dieser Studienarbeit nicht eingehen.

Als Parameter dieser ***Template***-Klasse wird das sog. ***Thread-Model*** angegeben.

Drei verschiedene Klassen stehen als ***Thread-Model*** zur Verfügung.

- **CComSingleThreadModel** besitzt zwei Operationen zum Inkrementieren und Dekrementieren des Wertes einer Variablen. Im Gegensatz zu den beiden nachfolgenden Klassen sind bei CComSingleThreadModel diese Operationen nicht ***Thread-Safe***. Solange auf ein COM-Objekt immer nur ein ***Client*** zu einer Zeit zugreifen kann, reicht dieses ***Thread-Model*** aus.

- **CComMultiThreadModel** weist ebenfalls die beiden Operationen Increment und Decrement auf. Im Gegensatz zu CComSingleThreadModel sind diese Operationen jedoch ***Thread-Safe*** implementiert.

- **CComMultiThreadModelNoCS** entspricht CComMultiThreadModel mit dem Unterschied, dass ***Critical Sections*** nur als Vortäuschung (***fake***) implementiert werden. Operationen wie Lock und Unlock zeigen somit keine Wirkung mehr.

Jede COM-Komponente besitzt in der realisierenden Klasse eine ***Schnittstellentabelle*** (***interface map***), die alle diejenigen Schnittstellen enthält, die von einem ***Client*** über QueryInterface angefordert werden können. Diese Tabelle wird auch als ***COM-map*** bezeichnet.

Für die Spezifikation dieser Tabelle stellt die ATL mehrere Makros zur Verfügung.
Eingeleitet wird die Tabelle durch **BEGIN_COM_MAP(x)**, wobei x der Name der Klasse ist, die die COM-Komponente realisiert. Für die Einträge in der Tabelle existieren mehrere COM_INTERFACE_ENTRY–Makros. Abgeschlossen wird die Tabelle mit dem Makro **END_COM_MAP()**.

Ein Beispiel:

```
BEGIN_COM_MAP(CMyKomponent)
     COM_INTERFACE_ENTRY(IDispatch)
     COM_INTERFACE_ENTRY(IMYKomponent)
END_COM_MAP()
```

Einige der möglichen COM_INTERFACE_ENTRY-Makros möchte ich im Folgenden vorstellen.

- **COM_INTERFACE_ENTRY(x)**
  Dies ist das einfachste Makro zum Hinzufügen einer Schnittstelle zur Schnittstellentabelle. Der Parameter x enthält den Namen einer Schnittstelle, von der die Klasse ***direkt*** erbt.

- **COM_INTERFACE_ENTRY2(x,x2)**
  Dieses Makro dient dazu, die Mehrdeutigkeit von zwei Ästen einer Vererbungsstruktur aufzulösen. Das Problem der Doppeldeutigkeit tritt z.B. auf, wenn eine Komponente von zwei ***dual***-Schnittstellen abgeleitet wurde. Dann ist nicht mehr eindeutig festgelegt, von welcher dieser zwei ***dual***-Schnittstellen die Schnittstelle IDispatch übernommen werden soll.

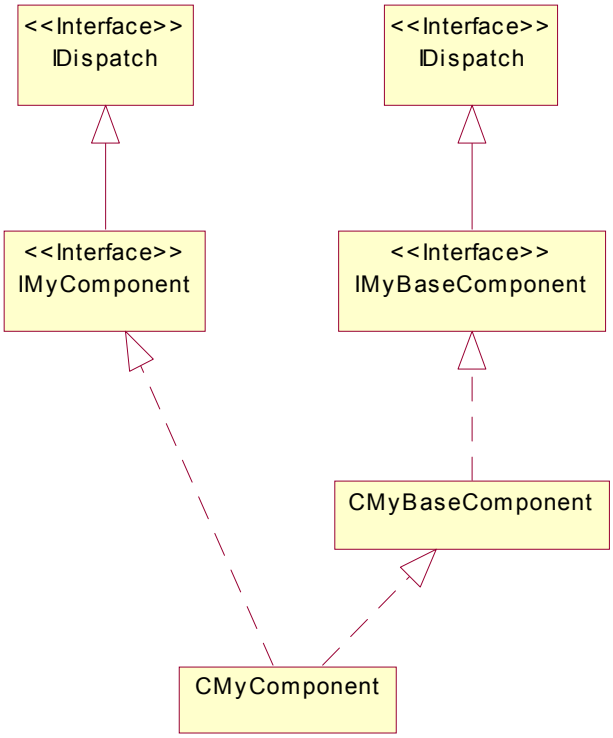

Abb. 3.2-1: Beispiel für COM_INTERFACE_ENTRY2

```
class CMyComponent
: public CMyBaseComponent  // Erbt von IDispatch
, public IDispatchImpl<IMyComponent, &IID_IMyComponent, &LIBID_MyLib>
{
     ...

     BEGIN_COM_MAP(CMyComponent)
          COM_INTERFACE_ENTRY2(IDispatch, IMyBaseComponent)
          ...
     END_COM_MAP()
};
```

Im obigen Quellcode-Beispiel wird festgelegt, dass die Schnittstelle IDispatch über IMyBaseComponent, d.h. über den rechten Ast obiger Vererbungsstruktur erreicht werden soll.

Der Parameter x beschreibt also die Schnittstelle, die nach außen offengelegt werden soll, während der Parameter x2 den Namen des Astes erhält, über den x offengelegt werden soll.

- **COM_INTERFACE_ENTRY_CHAIN(classname)**
  Dieses Makro kann bei Strukturen, wie sie in Abb. 3.2-1 dargestellt wurden, ebenfalls zum Einsatz kommen.
  Der Parameter classname enthält den Klassennamen einer Basisklasse der gegenwärtig betrachteten Komponente.
  Wenn dieser Eintrag in der Schnittstellentabelle erreicht wird, verarbeitet dieses Makro die Schnittstellentabelle der angegebenen Oberklasse.

  Beispiel:

```
class CMyComponent
: public CMyBaseComponent  // Erbt von IDispatch
, public IDispatchImpl<IMyComponent, &IID_IMyComponent, &LIBID_MyLib>
{
    ...

    BEGIN_COM_MAP(CMyComponent)
        COM_INTERFACE_ENTRY2(IDispatch, IMyBaseComponent)
        COM_INTERFACE_CHAIN(CMyBaseComponent)
    END_COM_MAP()
};
```

  Zu beachten ist, dass COM_INTERFACE_CHAIN(classname) nicht der oberste Eintrag einer Schnittstellentabelle sein darf. Dies liegt daran, dass der erste Schnittstelleneintrag eine direkte Schnittstelle der zu implementierenden Komponente sein muss. Soll die Schnittstellentabelle einer anderen als der aktuellen Komponente zuerst durchsucht werden, so muss zumindest der Eintrag für die Schnittstelle IUnknown vorgezogen werden.

```
BEGIN_COM_MAP(CMyComponent)
    COM_INTERFACE_ENTRY(IUnknown)
    COM_INTERFACE_CHAIN(CMyBaseComponent)
    ... // weitere Schnittstelleneinträge von MyComponent
END_COM_MAP()
```

  In allen anderen Fällen ist ein Eintrag für IUnknown nicht notwendig, da IUnknown grundsätzlich als vorhanden angenommen wird.

3.2.2.4 CComCoClass

CComCoClass ist eine weitere wichtige ***Template***-Klasse der ATL.

```
template< class T, const CLSID* pclsid >
class CComCoClass
```

Sie stellt Operationen bereit, um konsistent die CLSID einer Komponente zu erlangen und bestimmte Fehler-Informationen zu setzen. Jede COM-Komponente, von der Objekte von außerhalb erzeugt werden können, sollte von CComCoClass erben.

```
class CMyComponent
: ...
, public CComCoClass<CMyComponent, &CLSID_CMyComponent>
{
    ...
};
```

Weiterhin definiert diese ***Template***-Klasse die Standard-Objektfabrik (in der COM-Terminologie als ***class factory*** bezeichnet). Als Fabrik-Klasse wird dabei standardmäßig die Klasse CComClassFactory verwendet, die eine Implementierung der Schnittstelle IClassFactory vornimmt.

Über entsprechende Makros besteht jedoch auch die Möglichkeit, eine benutzerdefinierte Fabrik-Klasse festzulegen, um somit auf die Erzeugung von COM-Objekten Einfluss nehmen zu können.

Der Parameter T enthält die aktuelle Klasse, die von CComCoClass erben soll. Der zweite Parameter beschreibt die CLSID der COM-Komponente.

3.2.2.5 IDispatchImpl

Die für ***dual***-Schnittstellen wohl wichtigste Klasse ist IDispatchImpl.

```
template< class T, const IID* piid, const GUID* plibid, WORD wMajor = 1, WORD wMinor = 0, class
tihclass = CComTypeInfoHolder >
class IDispatchImpl : public T
```

IDispatchImpl stellt eine vorgegebene Implementierung für die Schnittstelle IDispatch zur Verfügung. Dies bedeutet, dass insbesondere die beiden Operationen GetIDsOfNames und Invoke in dieser ***Template***-Klasse implementiert sind.
Der Konstruktor dieser Klasse ruft AddRef auf, während der Destruktor ein Release durchführt.

Als Parameter werden dieser ***Template***-Klasse in der Regel der Name einer ***dual***-Schnittstelle, ein Zeiger auf die IID dieser Schnittstelle und ein Zeiger auf die ID der Typbibliothek übergeben. Die weiteren möglichen Parameter werden in der Praxis kaum benötigt, so dass ich hier nicht auf sie eingehen möchte. Den interessierten Leser verweise ich einmal mehr auf die Literaturhinweise im Anhang.

Das nachfolgende Beispiel zeigt nun eine typische Implementierung einer ***dual***-Schnittstelle.

```
class ATL_NO_VTABLE CYourClass
: public CComObjectRootEx<CComSingleThreadModel>
, public CComCoClass<CYourClass, &CLSID_YourClass>
, public IDispatchImpl<IYourClass, &IID_IYourClass, &LIBID_YourLib>
{
    ...
};
```

Das in dem obigen Beispiel bereits verwendete Makro **ATL_NO_VTABLE** teilt dem ***Compiler*** mit, dass für diese Klasse keine VTBL erzeugt werden soll. Dieses Makro kann nur verwendet werden, wenn von der nachfolgend deklarierten Klasse ***direkt*** keine Instanzen erzeugt werden können.

Dabei stellt sich die folgende Frage: Wozu soll eine Klasse definiert werden, von der anschließend keine Objekte erzeugen werden dürfen?
Die Antwort auf diese Frage kann in Form einer weiteren ATL-Klasse gegeben werden.

Die ATL lässt diese neue Klasse von einer Komponenten-Implementationsklasse (z.B. `CYourClass`) erben und erzeugt Instanzen immer von dieser neuen Unterklasse. Aus diesem Grund benötigt `CYourClass` auch keine VTBL, so dass mit dem Makro `ATL_NO_VTABLE` Speicherplatz gespart werden kann.

Als Unterklasse wird die ***Template***-Klasse `CComObject` verwendet, die unseren Klassennamen als Parameter erhält.

Folgendes Vererbungsdiagramm zeigt noch einmal die Abhängigkeiten.

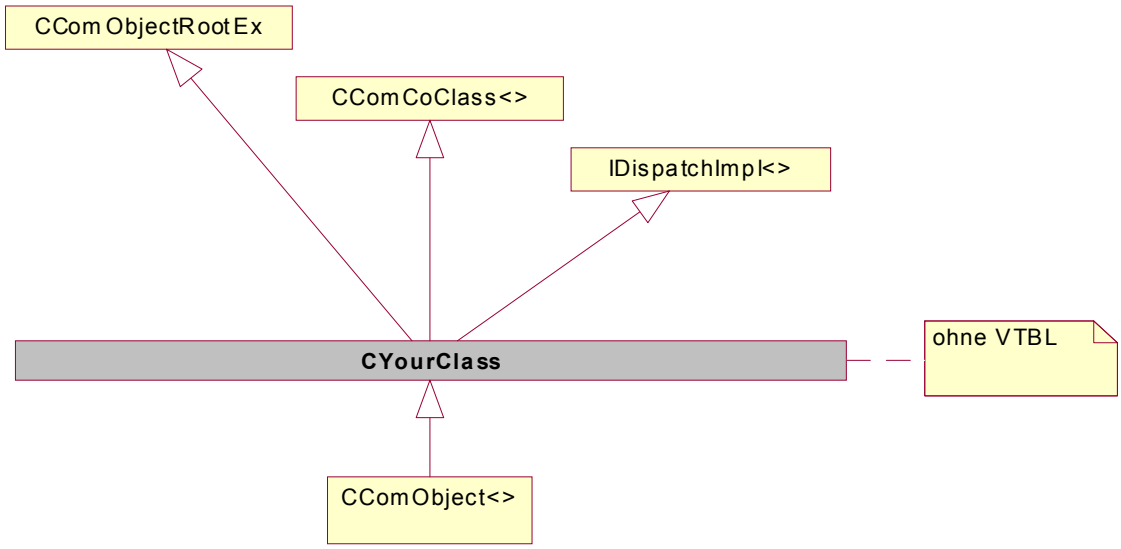

Abb. 3.2-2: Vererbungsdiagramm COM-Komponente

Zusammengefasst:
Eine COM-Komponente, die eine ***dual***-Schnittstelle implementiert, wird in der ATL durch eine Klasse implementiert, die von drei ***Template***-Klassen erbt. **CComObjectRoot** bzw. **CComObjectRootEx** übernimmt die Referenzzählung für die COM-Komponente, **CComCoClass** befasst sich mit der Objekt-Fabrik und **IDispatchImpl** liefert eine Standard-Implementation für IDispatch. COM-Objekte werden als Instanzen nicht direkt von der Implementierungs-Klasse, sondern von der ***Template***-Unterklasse **CComObject** erzeugt.

3.2.2.6 CComObject

COM-Objekte werden in ATL über die Klasse CComObject (oder der ihr verwandten Klassen CComAggObject bzw. CComPolyObject) erzeugt.

```
template< class Base >
class CComObject : public Base
```

Der Parameter ***Base*** enthält den Namen der benutzerdefinierten Oberklasse, die von CComObjectRoot bzw. CComObjectRootEx erben muss. Des weiteren erbt diese Oberklasse selbstverständlich auch von den Schnittstellen, welche die Komponente unterstützen soll.

CComObject implementiert die Schnittstelle **IUnknown**, wobei Aufrufe von QueryInterface, AddRef und Release jedoch an die entsprechenden Operationen in CComObjectRootEx delegiert werden.

CComObject muss deshalb die am weitesten abgeleitete Klasse sein, damit die Implementierungen für die Operationen von IUnknown von allen Schnittstellen, die von IUnknown erben, gemeinsam benutzt werden können.

Die bis jetzt vorgestellten Konzepte lassen vermuten, dass ein COM-Objekt nicht einfach durch den Aufruf von new auf die Implementierungsklasse erzeugt werden kann. Ein solcher Versuch wird von der ATL mit Fehlermeldungen quittiert. Um COM-Objekte zu erzeugen, kann stattdessen CoCreateInstance oder auch CComObject::CreateInstance verwendet werden.

3.2.2.7 Die Smart-Pointer-Klasse CComQIPtr

Ein ***Smart-Pointer*** ist eine Hüllklasse für Zeiger. Durch eine solche Kapselung kann sichergestellt werden, dass bestimmte Standard-Operationen automatisch ausgeführt werden. So kann unter anderem automatisch überprüft werden, ob alle Zeiger korrekt initialisiert wurden. Auf diese Weise können Programmabstürze vermieden werden, die durch ungültige Zeiger bedingt sind. Dies ist u.a. ein Grund, warum diese Hüllklassen auch als intelligente C++-Zeiger bezeichnet werden.
Eine ***Smart-Pointer***-Hüllklasse kapselt aber nicht nur den tatsächlichen Zeiger, sondern stellt auch einen Elementauswahloperator »->« zur Verfügung der bewirkt, dass sich die Klasse wie ein »normaler« C++-Zeiger verhält.

Für COM sind ***Smart-Pointer*** deshalb interessant, da sie eine sehr grundlegende Funktion, nämlich die Referenzzählung, übernehmen können.

In der ersten Version der ATL wurde die ***Smart-Pointer***-Klasse `CComPtr` eingeführt. Diese besitzt jedoch keine Unterstützung für `QueryInterface`. Daher veröffentlichte Microsoft eine überarbeitete, leistungsfähigere Version namens **`CComQIPtr`**. /Kruglinski, Wingo, Sheperd 98/

```
template< class T, const IID* piid >
class CComQIPtr
```

Der erste Parameter beschreibt den Typ des COM-Schnittstellenzeigers, der gekapselt werden soll. Der zweite Parameter enthält die IID der entsprechenden Schnittstelle. Seit der ATL-Version 2.1 muss der zweite Parameter nicht mehr explizit angegeben werden, sondern wird automatisch aus dem übergebenen Schnittstellentyp abgeleitet.

`CComQIPtr` bietet mehrere Konstruktoren an: einen Standardkonstruktor, einen Kopierkonstruktor, einen Konstruktor, der einen untypisierten Schnittstellenzeiger akzeptiert, und einen Konstruktor, dem als Parameter ein `IUnknown`-Schnittstellenzeiger übergeben werden kann.

Der Standardkonstruktor setzt den internen Zeiger zu NULL. Der Kopierkonstruktor setzt den internen Zeiger auf den entsprechenden internen Zeiger des übergebenen ***Smart-Pointers***. Anschließend ruft es AddRef für den internen Zeiger auf.
Wird dem Konstruktor ein Zeigertyp übergeben, der nicht vom Typ des ersten ***Template***-Parameters ist, so ruft der Konstruktor `QueryInterface` auf, um den internen Zeiger auf einen passenden Schnittstellenzeiger zu setzen.

Der Destruktor von `CComQIPtr` ruft `Release` auf.

Auf eine in dieser Arbeit verwendete Operation von `CComQIPtr` sei hier noch kurz hingewiesen.

```
HRESULT CComQIPtr::CopyTo( T** ppT )
```

Die Operation **`CopyTo`** kopiert den internen Zeiger in den Zielzeiger. »T** ppt« ist dabei ein Zeiger auf den Zielzeiger.
Der Referenzzähler des zugehörigen COM-Objektes wird bei Erfolg inkrementiert.
Der Rückgabeparameter vom Typ `HRESULT` erhält den Wert `S_OK`, wenn der Zeiger erfolgreich kopiert werden konnte. Sollte `ppT` gleich NULL sein, wird `E_POINTER` zurückgegeben.

4 Die COM-Schnittstelle von GO

GO wurde im Rahmen dieser Studienarbeit zu einem Automatisierungsserver erweitert. Dazu wurde GO mit einer Reihe von Schnittstellen auf COM-Komponenten versehen, die es ermöglichen, auf verschiedene Fachkonzeptobjekte über COM zuzugreifen. Letztendlich wurden in dieser Arbeit 11 COM-Komponenten und 11 benutzerdefinierte COM-Schnittstellen identifiziert. Zu beachten ist dabei, dass für jede COM-Schnittstelle eine zugehörige COM-Komponente existiert, wobei mehrere COM-Komponenten jedoch auch mehr als eine benutzerdefinierte Schnittstelle implementieren.

Bei dem Modul handelt es sich um einen lokalen Server, da sowohl der JANUS-Specifier, als auch das GO zwar auf demselben Rechner, jedoch in verschiedenen Prozessräumen ablaufen.

4.1 Bezeichner-Konventionen

Um eine eindeutige Unterscheidung zwischen COM-Modul, COM-Komponenten, COM-Schnittstellen und Klassen, die eine Komponente implementieren, vornehmen zu können, wurden in dieser Studienarbeit nachfolgende Bezeichnerkonventionen verwendet.

Das COM-Modul, das alle COM-Komponenten beinhaltet, ist die Anwendung »GO.exe«, eine monolithische C++-Anwendung.

Für die Bezeichnung der COM-Komponenten habe ich den Prefix »GO« vorgesehen.
Beispiele: `GOApplication, GOElement`

COM-Schnittstellen-Bezeichner beginnen per Konvention mit dem Buchstaben »I« und enthalten ebenfalls den Zusatz »GO« vor dem eigentlichen semantischen Bezeichner.
Beispiele: `IGOApplication, IGOElement`

Klassen, die eine Komponente (und damit alle Schnittstellen der Komponente) implementieren, möchte ich im Weiteren als Implementierungsklassen bezeichnen.

Diese Komponenten-Implementierungsklassen sind gekennzeichnet durch das Prefix »CGO«.
Beispiele : `CGOApplication, CGOElement`
Die entsprechenden Quellcode-Dateien für z.B. `CGOElement` heißen dann `CGOElement.h` und `CGOElement.cpp`.

4.2 Identifikation der COM-Komponenten und Schnittstellen

Die Identifikation der Komponenten und Schnittstellen erfolgte in mehreren Schritten. In einem ersten Schritt wurde die COM-Automatisierungsschnittstelle des Software-Analysewerkzeuges Rational Rose analysiert. Die dort implementierten Komponenten und Schnittstellen übersteigen die Anforderungen an das GO jedoch bei weitem. So ist in Rose eine nahezu vollständige Fernsteuerung der Anwendung – mit der Möglichkeit der Veränderung der Anwendung selbst – vorhanden.
So weit konnte und sollte die COM-Schnittstelle des GO im Rahmen dieser Studienarbeit nicht entwickelt werden.

In einem zweiten Schritt wurde die schon bestehende Kommunikation zwischen dem JANUS-Specifier und dem Software-Analysewerkzeug Rational Rose untersucht.

Aus der Anforderung, dass die Kommunikation zwischen JANUS und GO möglichst analog zu der Kommunikation zwischen JANUS und Rose ermöglicht werden sollte, wurde unter Berücksichtigung der GO-eigenen Strukturen (UML-Metamodell bildet das Fachkonzept) die nachfolgend beschriebene Komponenten- und Schnittstellen-Lösung identifiziert.

Hinweisen möchte ich an dieser Stelle bereits darauf, dass nicht die bestehenden Fachkonzeptklassen des GO als Implementierungsklassen der Komponenten gewählt wurden. Das GO-Fachkonzept besteht mittlerweile aus 180 Klassen, die untereinander starke Bindungen besitzen. Um diese Klassen in COM-Implementierungsklassen zu portieren, wäre ein enorm großer Aufwand notwendig geworden. Die Klassen müssten dazu nahezu neu programmiert werden. Deshalb habe ich mich entschieden, eine COM-Zugriffsschicht einzubauen, die einen Zugriff auf das Fachkonzept ermöglicht, dieses jedoch nach außen hin kapselt. Diese Struktur entspricht dem ***Fascade-Pattern*** in /Gamma et al. 95/.

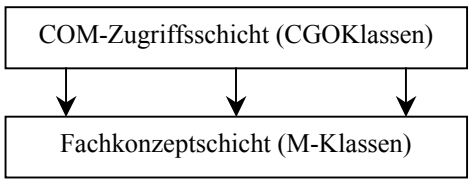

Abb. 4.2-1: Lineare Schichtentrennung

Aus dieser Entscheidung ergab sich während der Implementierung der enorme Vorteil, dass es kaum Überschneidungen bei der Bearbeitung von Klassen innerhalb des Entwicklerteams des GO gab.

Da das GO während der Erstellungszeit der Studienarbeit fortwährend weiterentwickelt wurde und auch in Zukunft noch weiterentwickelt werden wird, müssen die Schnittstellen dieser Entwicklung in Zukunft angepasst werden.

Bevor ich näher auf die einzelnen Schnittstellen eingehe, möchte ich zunächst die identifizierten COM-Komponenten mit ihren Schnittstellen tabellarisch vorstellen.

Bei dieser und auch allen nachfolgenden Darstellungen wurde aus Gründen der Übersichtlichkeit auf die Standard-Schnittstellen `IDispatch` und `IUnknown` verzichtet. Aufgeführt werden jeweils nur die benutzerdefinierten COM-Schnittstellen.
Alle COM-Komponenten des GO implementieren jedoch auch die Schnittstelle `IDispatch` und damit auch `IUnknown`.

COM-Komponente	*COM-Schnittstellen*	*Gekapselte M-Klasse*
GOApplication	IGOApplication	(CApplication)
GOCollection	IGOCollection	---
GOElement	IGOElement	MModelElement
GOPackage	IGOPackage, IGOElement	MPackage
GOModel	IGOModel, IGOPackage, IGOElement	MModel
GOClass	IGOClass, IGOElement	MClass
GOAttribute	IGOAttribute, IGOElement	MAttribute
GOOperation	IGOOperation, IGOElement	MOperation
GOParameter	IGOParameter, IGOElement	MParameter
GOAssociation	IGOAssociation, IGOElement	MAssociation
GORole	IGORole, IGOElement	MAssociationEnd

Tabelle 4.2-1: COM-Komponenten und ihre zugehörigen Schnittstellen in GO

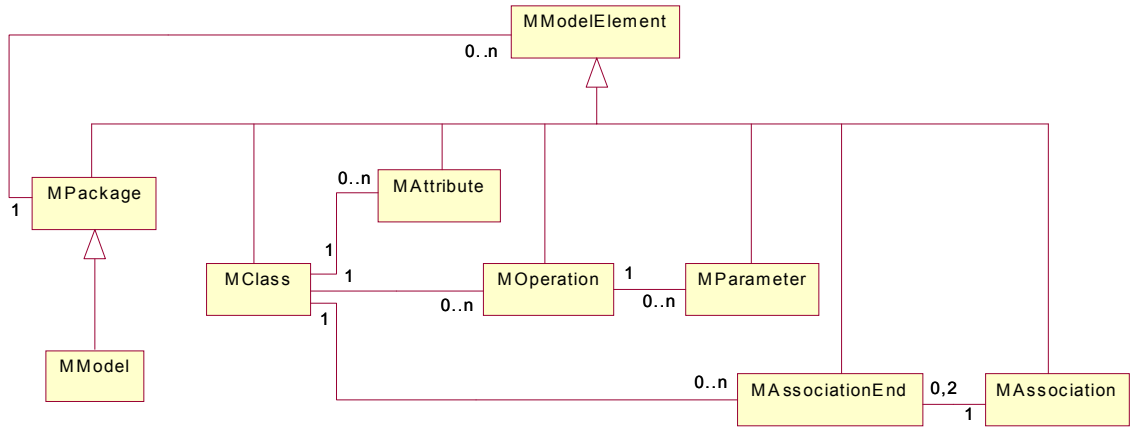

Abb. 4.2-2: UML-Modell der relevanten M-Klassen

Beispielhaft sei in Abb. 4.2-3 die COM-Komponente `CGOClass` mit ihrer benutzerdefinierten Schnittstelle `IGOClass` dargestellt. Sie besitzt eine unidirektionale Assoziation zu `MClass`, da entsprechend der linearen Schichtentrennung die COM-Komponente zwar die gekapselte Fachkonzeptklasse kennen muss, umgekehrt jedoch die Fachkonzeptklasse nicht die COM-Komponente.

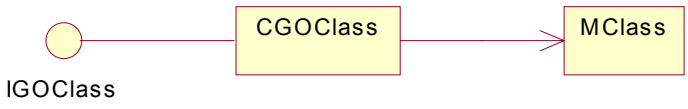

Abb. 4.2-3: COM-Komponente CGOClass mit Schnittstelle und Fachkonzeptklasse

4.2.1 GOApplication

Eine Instanz der COM-Komponente **GOApplication** repräsentiert das Anwendungsobjekt (in GO eine Instanz der Klasse CApplication) nach außen. Sie kapselt dabei die Sicht auf die Anwendung.
GOApplication besitzt nur eine benutzerdefinierte Schnittstelle, nämlich **IGOApplication**.
Wie bei allen weiteren Schnittstellen handelt es sich um eine ***dual***-Schnittstelle, die von IDispatch erbt und ausschließlich automatisierungskompatible Typen verwendet.
IGOApplication besitzt in dieser ersten Version des Automatisierungsservers nur eine einzige Operation, nämlich **GetCurrentModel**. Diese Operation liefert als Ausgabeparameter einen Zeiger auf einen Schnittstellenzeiger vom Typ IGOModel zurück.
Dieser Schnittstellenzeiger weist auf das COM-Objekt, welches das aktuelle Modell in GO repräsentiert.

Nachfolgend ist die IDL-Spezifikation der COM-Komponente GOApplication angegeben.

```
// ------------------------------------------------------------------------------
// GOApplication

[
    uuid(5EE61D40-C6E0-11D4-81C7-00E09882CA77),
    helpstring("GOApplication Class")
]
coclass GOApplication
{
    [default] interface IGOApplication;
};

[
    uuid(5EE61D41-C6E0-11D4-81C7-00E09882CA77),
    helpstring("IGOApplication Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOApplication : IDispatch
{
    [helpstring("method GetCurrentModel"), id(1)] HRESULT GetCurrentModel([out, retval] IGOModel
**ppModel);
};
```

4.2.2 GOElement

Die Komponente **GOElement** dient dazu, den Zugriff auf ein nicht weiter spezifiziertes Modell-Element nach außen zur Verfügung zu stellen. Diese COM-Komponente kapselt die Fachkonzeptklasse MModelElement des GO. GOElement besitzt die benutzerdefinierte Schnittstelle **IGOElement.**

IGOElement bietet Operationen für den Zugriff auf Eigenschaften, die jedes Modell-Element besitzt.
Es werden Lese- und Schreib-Operationen für die Eigenschaften **Name**, **Documentation** und **Stereotype** offengelegt, wobei jeweils ein Parameter vom Typ BSTR benutzt wird.
Zu jedem COM-Objekt, das eine IGOElement-Schnittstelle besitzt, liefert die Schnittstellen-Operation **GetID** einen eindeutigen Zahlenwert zurück, um dieses Objekt eindeutig identifizieren zu können. Dabei handelt es sich um einen durch die GO-Anwendung festgelegten Wert, der innerhalb des aktuellen Modells eindeutig ist, jedoch ***keine*** GUID ist.
Weiterhin bietet IGOElement die Operationen **GetProperty** und **SetProperty** an. Diese Operationen sind dazu geschaffen worden, um JANUS-spezifische Attribute als ***Tagged-Values*** in das GO-Modell einzufügen und wieder auslesen zu können.
Tagged-Values sind Werte (***Value***), die mit einem bestimmten Kennzeichen (***Tag***) zusammen gespeichert werden.
GetProperty besitzt einen Eingabeparameter PropertyName vom Typ BSTR, über den der Eigenschaftsname des gesuchten Wertes angegeben wird. Der Ausgabeparameter, der den gesuchten Wert zurückgibt, ist ebenfalls vom Typ BSTR.
Die Operation SetProperty besitzt neben den beiden BSTR-Eingabeparametern PropertyName und Value noch einen Ausgabeparameter vom Typ VARIANT_BOOL. Dieser Parameter liefert den Wert VARIANT_TRUE zurück, wenn der Wert gesetzt werden konnte. VARIANT_FALSE wird zurückgegeben, wenn der Wert nicht gesetzt werden konnte, weil z.B. der angegebene ***Tagged-Value***-Eintrag im Modell nicht existierte.

Die IDL-Spezifikation der COM-Komponente GOElement:

```
// ------------------------------------------------------------------------
// GOElement

[
    uuid(CA9476D1-CAAC-11D4-81C7-00E09882CA77),
    helpstring("GOElement Class")
]
coclass GOElement
{
    [default] interface IGOElement;
};
```

```
[
    uuid(CA9476D2-CAAC-11D4-81C7-00E09882CA77),
    helpstring("IGOElement Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOElement : IDispatch
{
    [propget, helpstring("property Name"), id(1)] HRESULT Name([out, retval] BSTR *pVal);
    [propput, helpstring("property Name"), id(1)] HRESULT Name([in] BSTR newVal);
    [propget, helpstring("property Documentation"), id(2)] HRESULT Documentation([out, retval]
BSTR *pVal);
    [propput, helpstring("property Documentation"), id(2)] HRESULT Documentation([in] BSTR
newVal);
    [propget, helpstring("property Stereotype"), id(3)] HRESULT Stereotype([out, retval] BSTR
*pVal);
    [propput, helpstring("property Stereotype"), id(3)] HRESULT Stereotype([in] BSTR newVal);
    [helpstring("method GetID"), id(4)] HRESULT GetID([out, retval] long *pID);
    [helpstring("method GetProperty"), id(5)] HRESULT GetProperty([in] BSTR PropertyName, [out,
retval] BSTR *pValue);
    [helpstring("method SetProperty"), id(6)] HRESULT SetProperty([in] BSTR PropertyName, [in]
BSTR Value, [out, retval] VARIANT_BOOL *ret);
};
```

4.2.3 GOCollection

Die Komponente **GOCollection** stellt eine Besonderheit dar. Sie repräsentiert nicht ein einzelnes Fachkonzept-Objekt, sondern eine Ansammlung von Objekten.
Sie bietet den Zugriff auf die in einer Ansammlung vorhandenen COM-Objekte. Die COM-Komponenten dieser COM-Objekte müssen jedoch die Schnittstelle IGOElement implementieren.

Einzige benutzerdefinierte Schnittstelle von GOCollection ist **IGOCollection**.

Sie enthält die Eigenschaft Count, die jedoch nur zum Lesen spezifiziert wurde. **Count** liefert dem ***Client*** die Anzahl der Elemente in der Objekt-Ansammlung.
Die Operation **GetAt** dient dann dazu einen Zeiger auf den IGOElement-Schnittstellenzeiger des gewünschten Objektes zurückzuliefern. Der Eingangsparameter ***Index*** gibt die Position innerhalb der Ansammlung an und liegt in einem Bereich von 0 bis ***Count*** – 1.

```
// ------------------------------------------------------------------------
// GOCollection

[
    uuid(715C16E5-CB80-11D4-81C7-00E09882CA77),
    helpstring("GOCollection Class")
]
coclass GOCollection
{
    [default] interface IGOCollection;
};

[
    uuid(715C16E6-CB80-11D4-81C7-00E09882CA77),
    helpstring("IGOCollection Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOCollection : IDispatch
{
    [propget, helpstring("property Count"), id(1)] HRESULT Count([out, retval] long *pVal);
    [helpstring("method GetAt"), id(2)] HRESULT GetAt([in] long Index, [out, retval] IGOElement
**ppElement);
};
```

In dieser ersten Version der IGOCollection-Schnittstelle existiert noch keine Operation zum Hinzufügen von COM-Objekten zu einer Sammlung. Die Funktionalität zum Erstellen eigener Collections (z.B. per Add) war für die Kommunikation zwischen GO und JANUS nicht erforderlich, ist für die Zukunft jedoch sicherlich eine sinnvolle Ergänzung.

4.2.4 GOPackage

GOPackage repräsentiert ein Package im UML-Modell. Die von dieser Komponente gekapselte Fachkonzeptklasse ist MPackage. Ein ***Package*** ist ein Strukturelement der UML. Es zeichnet sich dadurch aus, das es weitere Unter-***Packages*** und Klassen enthalten kann.
Daher besitzt die Schnittstelle **IGOPackage** auch Operationen, um Ansammlungen von Klassen und ***Packages*** in Form eines Zeigers auf einen IGOCollection-Schnittstellenzeiger zurückzugeben. Dies sind die Operationen **GetClasses** und **GetPackages**.

Da über die COM-Kommunikation zwischen GO und JANUS auch neue ***Packages*** in das Modell eingefügt werden sollen, existiert eine weitere Operation namens **AddPackage**. Dieser Operation wird nur der Name des zu erstellenden ***Packages*** übergeben. Zurückgegeben wird im Erfolgsfall ein Zeiger auf einen IGOPackage-Schnittstellenzeiger, der auf das neu angelegte ***Package***-Objekt zeigt.

Zu beachten ist, dass beim Hinzufügen eines neuen ***Package*** nur ein Fachkonzeptobjekt erzeugt wird, welches GO in der Baumansicht auch anzeigt. Es wird jedoch kein Diagramm-Objekt erzeugt, so dass das ***Package*** in den Diagrammen zunächst nicht angezeigt wird.

Weiterhin implementiert die Komponente GOPackage noch die Schnittstelle **IGOElement**, damit auch auf die Standardeigenschaften (z.B. Name des Package) zugegriffen werden kann.

```
// ------------------------------------------------------------------------------
// GOPackage

[
    uuid(CA9476CB-CAAC-11D4-81C7-00E09882CA77),
    helpstring("GOPackage Class")
]
coclass GOPackage
{
    [default] interface IGOPackage;
    interface IGOElement;
};

[
    uuid(CA9476CC-CAAC-11D4-81C7-00E09882CA77),
    helpstring("IGOPackage Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOPackage : IDispatch
{
    [helpstring("method AddPackage"), id(1)] HRESULT AddPackage([in] BSTR name, [out, retval]
IGOPackage **ppPackage);
    [helpstring("method GetAllClasses"), id(2)] HRESULT GetClasses([out, retval] IGOCollection
**ppCollection);
    [helpstring("method GetPackages"), id(3)] HRESULT GetPackages([out, retval] IGOCollection
**ppCollection);
};
```

4.2.5 GOModel

Die Komponente **GOModel** repräsentiert das Fachkonzeptmodell, welches alle weiteren Modell-Elemente beinhaltet. Die zugehörige Fachkonzeptklasse heißt MModel.
Semantisch betrachtet stellt ein Modell nichts weiter dar, als ein spezielles Package, nämlich das oberste Package in einer Hierarchie. Deswegen implementiert die Komponente GOModel neben **IGOElement** auch die Schnittstelle **IGOPackage**.
Die Schnittstelle **IGOModel** ermöglicht darüber hinaus einen direkten Zugriff auf die logische Sicht des UML-Modells über die Operation **GetLogicalView**. Diese logische Sicht ist ein ***Package*** auf der obersten Ebene des Modells.
Eine weitere wichtige Funktion des Modells ist es, alle enthaltenen Elemente identifizieren zu können. Deshalb enthält IGOModel eine weitere Operation namens **FindElementWithID**.
Diese erwartet als Eingabeparameter die ID eines Modell-Elementes, wie sie die Funktion GetID der Schnittstelle IGOElement zurückliefert. Zurückgegeben wird von FindElementWithID ein Zeiger auf einen IGOElement-Schnittstellenzeiger.

```
// ------------------------------------------------------------------------------
// GOModel

[
    uuid(CA9476C8-CAAC-11D4-81C7-00E09882CA77),
    helpstring("GOModel Class")
]
coclass GOModel
{
    [default] interface IGOModel;
    interface IGOPackage;
    interface IGOElement;
};

[
    uuid(CA9476C9-CAAC-11D4-81C7-00E09882CA77),
    helpstring("IGOModel Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOModel : IDispatch
{
    [helpstring("method FindElementWithID"), id(1)] HRESULT FindElementWithID([in] long ID, [out,
retval] IGOElement **ppElement);
    [helpstring("method GetLogicalView"), id(2)] HRESULT GetLogicalView([out, retval] IGOPackage
**ppPackage);
};
```

4.2.6 GOClass

Die Komponente **GOClass** repräsentiert die Fachkonzeptklasse MClass.
Neben der Schnittstelle **IGOElement** besitzt GOClass die benutzerdefinierte Schnittstelle **IGOClass**.
Diese beinhaltet lesende und schreibende Operationen auf die Eigenschaft **Abstract**. Der Parameter vom Typ VARIANT_BOOL kann dabei die Werte VARIANT_TRUE oder VARIANT_FALSE annehmen.
Eine Klasse kann Attribute und Operationen besitzen. Darüber hinaus kann eine Klasse von einer oder mehreren Oberklassen erben und Assoziationen zu anderen Klassen besitzen.
Eine Assoziationsinstanz beschreibt Beziehungen zwischen Objekten. Modelliert werden Assoziationen jedoch zwischen Klassen. Daher ist es üblich von Assoziationen zwischen Klassen zu sprechen.
Die Schnittstelle **IGOClass** besitzt einige Operationen für den Zugriff auf die mit der Klasse verknüpften COM-Objekte. Dies sind im einzelnen **GetAttributes**, **GetOperations**, **GetAssociations** und **GetSuperclasses**, die jeweils einen Zeiger auf einen IGOCollection-Schnittstellenzeiger zurückgeben.
Zum Hinzufügen von Attributen zu einer Klasse existiert die Operation **AddAttribute**.
Als Eingabeparameter vom Typ BSTR werden der Name des Attributes, der Typ und ein Anfangswert erwartet. Zurückgegeben wird im Erfolgsfall ein Zeiger auf den IGOAttribute-Schnittstellenzeiger des erzeugten COM-Objektes.
Attribute können über die Operation **DeleteAttribute** auch wieder gelöscht werden. Dazu muss als Eingabeparameter ein IGOAttribute-Schnittstellenzeiger übergeben werden. Der Rückgabewert vom Typ VARIANT_BOOL liefert VARIANT_TRUE zurück, wenn das Attribut gelöscht werden konnte, andernfalls VARIANT_FALSE.

Neben IGOClass implementiert GOClass auch die Schnittstelle **IGOElement.**

```
// ------------------------------------------------------------------------
// GOClass

[
    uuid(5A1FC5A3-CB79-11D4-81C7-00E09882CA77),
    helpstring("GOClass Class")
]
coclass GOClass
{
    [default] interface IGOClass;
    interface IGOElement;
};
```

```
[
    uuid(5A1FC5A4-CB79-11D4-81C7-00E09882CA77),
    helpstring("IGOClass Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOClass : IDispatch
{
    [propget, helpstring("property Abstract"), id(1)] HRESULT Abstract([out, retval] VARIANT_BOOL
*pVal);
    [propput, helpstring("property Abstract"), id(1)] HRESULT Abstract([in] VARIANT_BOOL newVal);
    [helpstring("method GetAllAttributes"), id(2)] HRESULT GetAttributes([out, retval]
IGOCollection **ppCollection);
    [helpstring("method GetAllOperations"), id(3)] HRESULT GetOperations([out, retval]
IGOCollection **ppCollection);
    [helpstring("method AddAttribute"), id(4)] HRESULT AddAttribute([in] BSTR Name, [in] BSTR
Type, [in] BSTR InitValue, [out, retval] IGOAttribute **ppAttribute);
    [helpstring("method DeleteAttribute"), id(5)] HRESULT DeleteAttribute([in] IGOAttribute
*pAttribute, [out, retval] VARIANT_BOOL *ret);
    [helpstring("method GetAllAssociations"), id(6)] HRESULT GetAssociations([out, retval]
IGOCollection **ppCollection);
    [helpstring("method GetAllSuperclasses"), id(7)] HRESULT GetSuperclasses([out, retval]
IGOCollection **ppCollection);
};
```

4.2.7 GOAttribute

Die Komponente **GOAttribute** kapselt die Fachkonzeptklasse MAttribute.
Neben **IGOElement** besitzt diese Komponente die weitere Schnittstelle **IGOAttribute**.

Attribute besitzen grundsätzlich einen Attribut-Typ und einen Anfangswert.
IGOAttribute bietet den Zugriff auf diese beiden Eigenschaften **Type** und **InitValue** sowohl lesend als auch schreibend an. Als Parameter dient jeweils ein Wert vom Typ BSTR.

Weiterhin kann ein Attribut ein Klassenattribut sein oder auch ein sog. abgeleitetes Attribut.
Abgeleitete Attribute lassen sich aus anderen Attributwerten jederzeit berechnen. Sie entstehen in der Regel aus Modellierungsgründen. In der Analysephase wird ein Attribut modelliert, in der Entwurfsphase erkennt man dann, dass sich dies auch berechnen bzw. ableiten lässt.
Um feststellen zu können, ob es sich bei einem Attribut um ein Klassenattribut (***static***) oder um ein abgeleitetes Attribut (***derived***) handelt, existieren die beiden Eigenschaften **Static** und **Derived**. Beide benutzen einen Parameter vom Typ VARIANT_BOOL für den Zugriff.

```
// ------------------------------------------------------------------------
// GOAttribute

[
    uuid(5A1FC5A5-CB79-11D4-81C7-00E09882CA77),
    helpstring("GOAttribute Class")
]
coclass GOAttribute
{
    [default] interface IGOAttribute;
    interface IGOElement;
};

[
    uuid(5A1FC5A6-CB79-11D4-81C7-00E09882CA77),
    helpstring("IGOAttribute Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOAttribute : IDispatch
{
    [propget, helpstring("property Type"), id(1)] HRESULT Type([out, retval] BSTR *pVal);
    [propput, helpstring("property Type"), id(1)] HRESULT Type([in] BSTR newVal);
    [propget, helpstring("property Static"), id(2)] HRESULT Static([out, retval] VARIANT_BOOL
*pVal);
    [propput, helpstring("property Static"), id(2)] HRESULT Static([in] VARIANT_BOOL newVal);
    [propget, helpstring("property Derived"), id(3)] HRESULT Derived([out, retval] VARIANT_BOOL
*pVal);
    [propput, helpstring("property Derived"), id(3)] HRESULT Derived([in] VARIANT_BOOL newVal);
    [propget, helpstring("property InitValue"), id(4)] HRESULT InitValue([out, retval] BSTR
*pVal);
    [propput, helpstring("property InitValue"), id(4)] HRESULT InitValue([in] BSTR newVal);
};
```

4.2.8 GOOperation

Die Komponente **GOOperation** repräsentiert Operationen einer Klasse und somit die Fachkonzeptklasse MOperation.
Sie besitzt neben **IGOElement** die Schnittstelle **IGOOperation**.
IGOOperation enthält Operationen, um den Rückgabetyp einer Operation über die Eigenschaft **ReturnType** lesen und schreiben zu können. Der dazu benutzte Parameter ist vom Typ BSTR.
Die Signatur einer Operation zeichnet sich neben dem Namen und dem Rückgabetyp der Operation auch durch die Parameter der Operation aus. Deshalb beinhaltet IGOOperation die Operation **GetParameters**, die einen Zeiger auf einen IGOCollection-Schnittstellenzeiger zurückgibt.
Parameter können über die COM-Schnittstelle auch zu einer Operation hinzugefügt werden. Dazu dient die Operation **AddParameter**. Als Eingabeparameter werden ***Name***, ***Typ*** und ***Anfangswert*** des Parameters als BSTRs erwartet. Weiterhin kann noch die Position, an der der Parameter eingefügt werden soll, angegeben werden. Zurückgegeben wird im Erfolgsfall ein Zeiger auf die IGOParameter-Schnittstelle des erzeugten COM-Objektes.
Die Operation **RemoveAllParameters** besitzt werden Ein- noch Ausgabeparameter und dient dem Löschen aller Parameter der aktuellen Operation.

```
// ------------------------------------------------------------------------------
// GOOperation

[
    uuid(5A1FC5AB-CB79-11D4-81C7-00E09882CA77),
    helpstring("GOOperation Class")
]
coclass GOOperation
{
    [default] interface IGOOperation;
    interface IGOElement;
};

[
    uuid(5A1FC5AC-CB79-11D4-81C7-00E09882CA77),
    helpstring("IGOOperation Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOOperation : IDispatch
{
    [propget, helpstring("property ReturnType"), id(1)] HRESULT ReturnType([out, retval] BSTR
*pVal);
    [propput, helpstring("property ReturnType"), id(1)] HRESULT ReturnType([in] BSTR newVal);
    [helpstring("method GetParameters"), id(2)] HRESULT GetParameters([out, retval] IGOCollection
**ppCollection);
    [helpstring("method RemoveAllParameters"), id(3)] HRESULT RemoveAllParameters();
    [helpstring("method AddParameter"), id(4)] HRESULT AddParameter([in] BSTR Name, [in] BSTR
Type, [in] BSTR InitValue, [in] long Position, [out, retval] IGOParameter **ppParameter);
};
```

4.2.9 GOParameter

Die Komponente **GOParameter** stellt einen Parameter einer Operation nach außen zur Verfügung. Sie kapselt damit die Fachkonzeptklasse MParameter.

Auf die Standardeigenschaften eines Parameters kann über die Schnittstelle **IGOElement** zugegriffen werden.
Für die zusätzliche Eigenschaft **Type** besitzt GOParameter die Schnittstelle IGOParameter.
Auf Type kann lesend wie schreibend über einen Parameter vom Typ BSTR zugegriffen werden.

```
// ------------------------------------------------------------------------------
// GOParameter

[
    uuid(5A1FC5AD-CB79-11D4-81C7-00E09882CA77),
    helpstring("GOParameter Class")
]
coclass GOParameter
{
    [default] interface IGOParameter;
    interface IGOElement;
};

[
    uuid(5A1FC5AE-CB79-11D4-81C7-00E09882CA77),
    helpstring("IGOParameter Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOParameter : IDispatch
{
    [propget, helpstring("property Type"), id(1)] HRESULT Type([out, retval] BSTR *pVal);
    [propput, helpstring("property Type"), id(1)] HRESULT Type([in] BSTR newVal);
};
```

4.2.10 GOAssociation

GOAssociation kapselt die Fachkonzeptklasse MAssociation.
Eine Assoziation verbindet zwei Klassen miteinander, deren Objekte Beziehungen eingehen können. In der UML zeichnet sie sich dadurch aus, dass an den Assoziations-Enden Rollennamen und Kardinalitäten angegeben werden können.

Das GO-Schnittstellenmodell sieht vor, dass eine Klasse alle ihre Assoziationen kennt, die Assoziation jedoch nur ***indirekt*** den Zugriff auf die beiden Klassen an den Assoziationsenden ermöglicht.

Statt dessen erlaubt die Schnittstelle **IGOAssociation** den Zugriff auf zwei COM-Objekte vom Typ GORole. Die Operationen **GetRole1** und **GetRole2** liefern jeweils einen Zeiger auf den IGORole-Schnittstellenzeiger dieser Rollen-Objekte.

Erst über die Rollen-Objekte ist ein Zugriff auf die Klassen-Objekte der Assoziation möglich.

```
// ------------------------------------------------------------------------
// GOAssociation

[
    uuid(5A1FC5A7-CB79-11D4-81C7-00E09882CA77),
    helpstring("GOAssociation Class")
]
coclass GOAssociation
{
    [default] interface IGOAssociation;
    interface IGOElement;
};

[
    uuid(5A1FC5A8-CB79-11D4-81C7-00E09882CA77),
    helpstring("IGOAssociation Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGOAssociation : IDispatch
{
    [helpstring("method GetRole1"), id(1)] HRESULT GetRole1([out, retval] IGORole **ppRole1);
    [helpstring("method GetRole2"), id(2)] HRESULT GetRole2([out, retval] IGORole **ppRole2);
};
```

4.2.11 GORole

Die COM-Komponente **GORole** stellt von der Benennung her einen Sonderfall dar. Obwohl die zugehörige Fachkonzeptklasse im GO MAssociationEnd (UML-konform, /OMG 99/) heißt, wurde diese Komponente nicht GOAssociationEnd bezeichnet. Dies liegt an der Namensgebung der Komponenten und Schnittstellen im Software-Analysewerkzeug Rose, an dem ich mich bei der Erstellung der COM-Schnittstelle des GO orientiert habe. In Rose gibt es eine COM-Komponente namens Role, die der hier vorgestellten Komponente GORole entspricht.

Außer **IGOElement** implementiert GORole auch noch die Schnittstelle **IGORole**.
Die Kardinalität eines Assoziationsendes kann über die Eigenschaft **Cardinality** gelesen und geschrieben werden. Dazu wird ein BSTR-Parameter verwendet.
Außerdem wird eine Eigenschaft namens **Navigable** über einen VARIANT_BOOL-Parameter offengelegt. Hierzu ist folgende wichtige Anmerkung zu machen. GO benutzt im Zusammenhang mit gerichteten Assoziationen den Ausdruck »gerichtet« anstatt »navigierbar« (dieser Ausdruck entstammt dem Rational Rose bzw. dem UML-Metamodell). Eine Assoziation, die von Klasse A nach Klasse B gerichtet ist, erhält in GO für Klasse B das Attribut ***navigable***, da die Assoziation auf Klasse B gerichtet ist.

Über die Operation **GetClass**, die einen Zeiger auf einen IGOClass-Schnittstellenzeiger zurückliefert, kann auf die mit dieser Rolle verbundene Klasse zugegriffen werden.
Die Operation **GetOtherRole** liefert einen Zeiger auf den IGORole-Schnittstellenzeiger des COM-Objektes der anderen Rolle. Gemeint ist damit das andere Assoziationsende der zugehörigen Assoziation.

```
// ------------------------------------------------------------------------------
// GORole

[
    uuid(5A1FC5A9-CB79-11D4-81C7-00E09882CA77),
    helpstring("GORole Class")
]
coclass GORole
{
    [default] interface IGORole;
    interface IGOElement;
};

[
    uuid(5A1FC5AA-CB79-11D4-81C7-00E09882CA77),
    helpstring("IGORole Interface"),
    dual,
    object,
    oleautomation,
    pointer_default(unique)
]
interface IGORole : IDispatch
{
    [propget, helpstring("property Cardinality"), id(1)] HRESULT Cardinality([out, retval] BSTR
*pVal);
    [propput, helpstring("property Cardinality"), id(1)] HRESULT Cardinality([in] BSTR newVal);
    [propget, helpstring("property Navigable"), id(2)] HRESULT Navigable([out, retval]
VARIANT_BOOL *pVal);
    [propput, helpstring("property Navigable"), id(2)] HRESULT Navigable([in] VARIANT_BOOL
newVal);
    [helpstring("method GetClass"), id(3)] HRESULT GetClass([out, retval] IGOClass **ppClass);
    [helpstring("method GetOtherRole"), id(4)] HRESULT GetOtherRole([out, retval] IGORole
**ppRole);
};
```

4.3 Vorgehen bei der Erstellung des Automatisierungsservers

Im Kapitel 4.2 wurde beschrieben, welche COM-Komponenten und COM-Schnittstellen in dieser Arbeit identifiziert wurden. Zum besseren Verständnis wurden bereits die Spezifikationen in IDL angegeben. Die Erstellung des Automatisierungsservers wurde jedoch nicht vollständig von Hand durchgeführt, sondern durch geeignete Werkzeuge unterstützt. Die vorliegende IDL-Datei wurde z.B. von Rational Rose 2000 generiert und anschließend den endgültigen Bedürfnissen von Hand angepasst.

In diesem Kapitel soll auf die Unterstützung durch Software-Werkzeuge bei der Erstellung des Automatisierungsservers eingegangen werden.

4.3.1 Hinzufügen der ATL-Unterstützung zu GO

Um in einem bestehenden ***Microsoft Visual C++***-Projekt COM-Komponenten mit Hilfe der ATL implementieren zu können, muss zunächst die sogenannte ***ATL-Unterstützung*** zu dem Projekt hinzugefügt werden. Diese beinhaltet das Einbinden der ATL-***Header***-Dateien, das Implementieren der Standardfunktionalität für das Modul, das Einbinden einer IDL-Datei und der zugehörigen ***Registry-Script***-Datei und einige Einstellungen in den Projekt-Dateien.

Verwendet man bei der Erstellung einer neuen Anwendung den ATL-COM-Anwendungs-assistenten, wird die ATL-Unterstützung automatisch hinzugefügt. Auch bei einem bestehenden MFC-Projekt kann die ATL-Unterstützung nachträglich hinzugefügt werden.
Dazu kann der ATL-Objekt-Assistent verwendet werden.

In der Programmierumgebung des MSVC wählt man »Einfügen: Neues ATL-Objekt...«. Daraufhin stellt der ATL-Assistent (***Wizard***) die Frage »Wollen Sie Ihrem MFC-Projekt die ATL-Unterstützung hinzufügen?“. Nachdem diese Frage mit »Ja« beantwortet und der ATL-Assistent anschließend abgebrochen wurde, sollte die ATL-Unterstützung hinzugefügt worden sein.
Dieses Vorgehen wird in /MSDN 00/ und /Rector, Sells 00/ beschrieben.
Beim Aufruf des ATL-Objekt-Assistenten wird dabei eine ***Script***-Datei namens »Addatl.ctl« ausgeführt. Dieses ***Script*** geht davon aus, dass die Dateien, die die Anwendungsklasse (abgeleitet von `CWinApp`) implementieren, mit dem Namen des Projektes übereinstimmen. D.h. für das Projekt »GO« wurden die Dateien »`GO.h`“ und »`GO.cpp`« erwartet. In GO wurde die Anwendungsklasse jedoch mit »CApplication« benannt, weshalb die entsprechenden Dateinamen auch »`Application.h`« und »`Application.cpp`« lauten.

Aus diesem Grund habe ich die ***Script***-Datei zum Hinzufügen der ATL-Unterstützung für diesen speziellen Fall modifiziert.

Der in der Datei »Addatl.ctl« enthaltene Code-Abschnitt

```
[!if!(ProjectNameHeader)]
[!AddStringToSymbol(ProjectNameHeader, ProjectName, ".h")]
[!endif]

[!if!(ProjectNameCPP)]
[!AddStringToSymbol(ProjectNameCPP, ProjectName, ".cpp")]
[!endif]
```

wurde durch die nachfolgenden zwei Zeilen ersetzt.

```
[!set(ProjectNameHeader, "Application.h")]
[!set(ProjectNameCPP, "Application.cpp")]
```

4.3.2 Modellierung der Komponenten und Schnittstellen in Rational Rose 2000

Das Software-Analyse- und Modellierungs-Werkzeug Rational Rose 2000 besitzt die Möglichkeit, sowohl ein bestehendes MSVC-Projekt zu analysieren (***Reengineering***), als auch aus einem gegebenen Modell Quelltext zu generieren.

Von GO existierte bereits eine Modell-Datei für das Rose 2000, da auch das Fachkonzept von GO zunächst in Rose modelliert und dann als Quelltext generiert worden ist.

Zur Aktualisierung des bestehenden Modells wurde in Rose zunächst der Befehl »***Update Model from Code***« aufgerufen.

Um mit Rose jedoch auch COM-Komponenten einlesen und erstellen zu können, mussten noch einige Maßnahmen getroffen werden.

So wurde in der Komponenten-Ansicht eine neue Komponente namens »Go.idl« hinzugefügt.
Für diese neue Komponente wurden als Einstellungen ***Stereotype*** auf »MIDL« und ***Language*** auf »VC++« gesetzt.
In den allgemeinen Einstellungen (***Properties → General***) wurde als ***WorkspaceFile*** das GO-Projekt ausgewählt.

Anschließend konnte das erste Mal eine Aktualisierung des Modells aus dem Quelltext erfolgen, wobei auch die COM-Komponenten und Schnittstellen berücksichtigt wurden.

Über einen in Rational Rose verfügbaren ATL-Objekt-Assistenten konnten die im Kapitel 4.2 identifizierten Komponenten und Schnittstellen eingegeben und somit dem Modell hinzugefügt werden.

Die hinzugefügten Komponenten werden grafisch in der UML-Notation angezeigt.

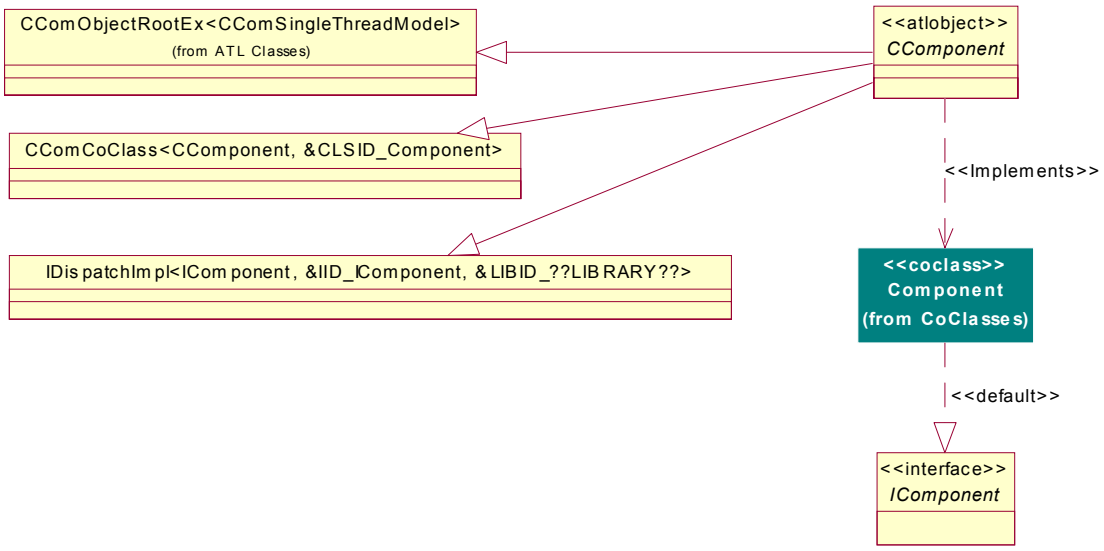

Abb. 4.3-1: OOA-Diagramm einer COM-Komponente

In Abb. 4.3-1 ist eine COM-Komponente namens `Component` mit einer COM-Schnittstelle `IComponent` dargestellt. Üblicherweise werden die drei Basisklassen `CComObjectRootEx`, `CComCoClass` und `IDispatchImpl` jedoch nicht in den Diagrammen dargestellt. Nach dem Hinzufügen einer Eigenschaft `Name` und einer Operation `Method`, sieht das zugehörige ***vereinfachte OOA-Diagramm*** wie folgt aus:

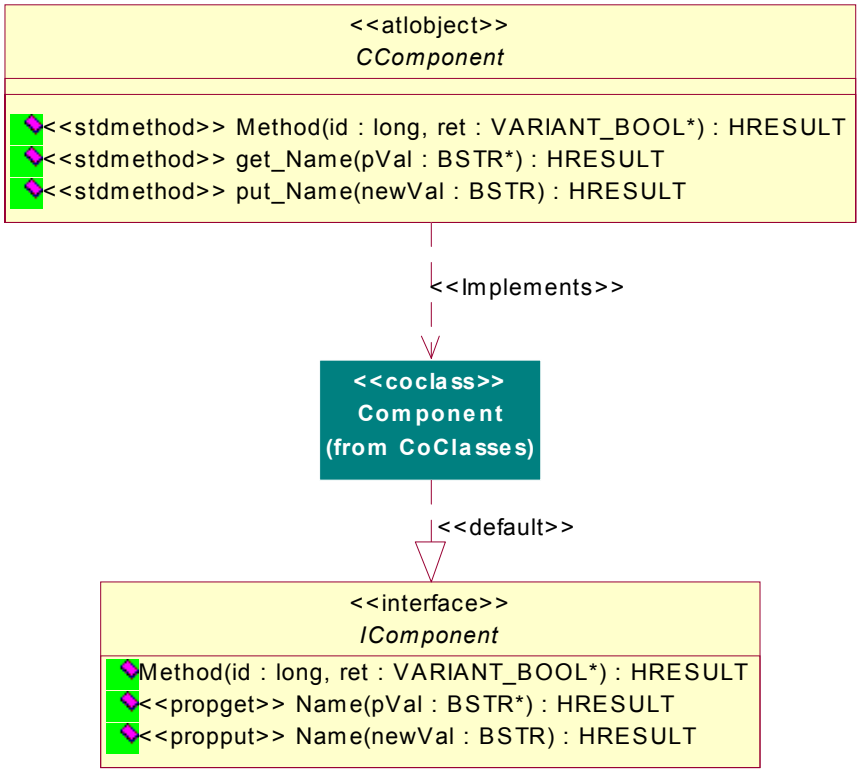

Abb. 4.3-2: Vereinfachtes OOA-Diagramm einer COM-Komponente

Die OOA-Diagramme der GO-Komponenten sind im Anhang C zu finden.

Durch Aufrufen des Befehls »Update Code« wurden neben der IDL-Datei und den ***Registry-Script***-Dateien auch Rümpfe für die zu implementierenden Klassen in das MSVC-Projekt generiert. Dadurch konnte der Gesamt-Implementierungsaufwand deutlich reduziert werden.

Änderungen konnten sodann nach Bedarf sowohl im Quelltext, als auch im Modell erfolgen, wenn anschließend der notwendige Abgleich durchgeführt wurde.

4.4 Implementierung der COM-Komponenten

Um die IGOElement-Schnittstelle für die COM-Komponenten CGOElement, CGOPackage, CGOModel, CGOClass, CGOAttribute, CGOOperation, CGOParameter, CGOAssociation und CGORole nicht mehrfach implementieren zu müssen, wurde bei der Implementierung von Vererbung Gebrauch gemacht.
Einen Überblick über die Vererbungshierarchie gibt Abb. C-1.

Von der Möglichkeit der Schnittstellen-Vererbung wurde abgesehen, da dies zu Problemen bei der Versionierung der Schnittstellen (zu hohe Abhängigkeiten) geführt hätte.

Entsprechend der Beschreibung in Kapitel 4.3.1 wurde die ATL-COM-Unterstützung zu GO hinzugefügt. Die entsprechenden Quelltext-Änderungen bzw. Erweiterungen sind dem Anhang E.2 zu entnehmen.

Zu erwähnen ist, dass das Füllen der ***ObjectMap*** von Hand erfolgte, da beim ***Code Update*** des Softwarewerkzeuges Rational Rose hierfür keine Einträge erzeugt wurden.

```
BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_GOApplication, CGOApplication)
OBJECT_ENTRY(CLSID_GOCollection, CGOCollection)
OBJECT_ENTRY(CLSID_GOElement, CGOElement)
OBJECT_ENTRY(CLSID_GOPackage, CGOPackage)
OBJECT_ENTRY(CLSID_GOModel, CGOModel)
OBJECT_ENTRY(CLSID_GOClass, CGOClass)
OBJECT_ENTRY(CLSID_GOAttribute, CGOAttribute)
OBJECT_ENTRY(CLSID_GOOperation, CGOOperation)
OBJECT_ENTRY(CLSID_GOParameter, CGOParameter)
OBJECT_ENTRY(CLSID_GOAssociation, CGOAssociation)
OBJECT_ENTRY(CLSID_GORole, CGORole)
END_OBJECT_MAP()
```

Die Implementierung jeder einzelnen Komponente im Detail zu beschreiben, wäre sicherlich nicht sinnvoll, da viele Operationen nahezu identisch ausprogrammiert werden konnten.

Ich will daher grundsätzliche Implementierungsdetails anhand einzelner Komponenten beschreiben. Analoge Implementierungen lassen sich dann in der Regel in weiteren Komponenten (s. Anhang E.4) finden.

Als erstes möchte ich auf die Initialisierung des Schnittstellenmodells eingehen.

4.4.1 Erzeugung und Initialisierung der COM-Objekte

Neben dem Fachkonzeptmodell muss ein eigenständiges Schnittstellenmodell aus COM-Objekten aufgebaut werden. Die erzeugten COM-Objekte müssen jeweils die Fachkonzept-Objekte kennen, die sie repräsentieren sollen. Die Fachkonzeptobjekte sollen jedoch von der Existenz des Schnittstellenmodells nichts wissen. So erreicht man eine lineare Schichtentrennung. COM-Objekte greifen auf Fachkonzeptobjekte zu, umgekehrt jedoch nicht.

Das Erzeugen der einzelnen COM-Objekte ist an sich schon nicht trivial, da sie nicht über den new-Operator erzeugt werden können. Vielmehr muss mittels CreateInstance auf ein CComObject<...> eine Instanz der Komponente erzeugt werden. Damit eine solche Instanz nicht direkt wieder zerstört wird, muss anschließend unbedingt AddRef aufgerufen werden.

Deklaration in CGOApplication.h:

```
CComObject<CGOModel> *pGOModel;     // Pointer auf das Model
```

Erzeugen eines Objektes in CGOApplication.cpp:

```
pGOModel = NULL;

HRESULT hr = CComObject<CGOModel>::CreateInstance(&pGOModel);

if (SUCCEEDED(hr))
{
     pGOModel->AddRef();
     ...
}
```

Das Schnittstellenmodell soll initialisiert werden, sobald ein ***Client***-Programm mittels CoCreateInstance eine Instanz der Komponente GOApplication erzeugt. Aus dem Konstruktor der Klasse CGOApplication heraus wird die Erzeugung aller weiteren COM-Objekte über die Operation CreateCOMObjects angestoßen.

Da jedes COM-Objekt wissen muss, welches Fachkonzeptobjekt es vertritt, wird mit der Operation SetMModelElement jedem COM-Objekt das zugehörige MModelElement mitgeteilt. Damit die Rückgängig-/Wiederherstellen-Technik des GO benutzt werden kann, muss jedem COM-Objekt eine sog. ***ActionGroup*** zugewiesen werden. Dazu dient die Operation SetActionGroup.

Nachfolgend sei der Konstruktor und die Operation CreateCOMObjects der Klasse CGOApplication aufgeführt.

```
CGOApplication::CGOApplication()
{
     if (theApp.GetDocument() == NULL)
     {
          //AfxMessageBox("GO muss gestartet sein und ein Modell muss existieren!");
          return;
     }

     pGOModel = NULL;

     pActionManager = theApp.GetDocument()->GetActionManager();
     pActionGroup = new ActionGroup("Änderungen über COM");

     CreateCOMObjects();
}

CGOApplication::CreateCOMObjects()
{
     HRESULT hr = CComObject<CGOModel>::CreateInstance(&pGOModel);

     if (SUCCEEDED(hr))
     {
          pGOModel->AddRef();
          // Das MModelElement initialisieren
          pGOModel->SetMModelElement((MModelElement*)theApp.GetDocument()->GetMModel());
          pGOModel->SetActionGroup(pActionGroup);
          pGOModel->CreateCOMObjects();
     }
     else
     {
          AfxMessageBox("Konnte in 'CGOApplication' kein GOModel anlegen.");
          pGOModel = NULL;
     }
}
```

In obigem Quelltext ist die Aufrufreihenfolge für das Anlegen eines CGOModel-Objektes aus der Komponente CGOApplication heraus ersichtlich.

Da das Modell beliebig komplex sein kann und die Reihenfolge der Aufrufe bei der Initialisierung eine wichtige Rolle spielt, habe ich im Anhang C ein stark vereinfachtes Sequenzdiagramm der Initialisierungsphase wiedergegeben (Abb. C-13 & Abb. C-14).
In dem Diagramm wird die Erzeugung von nur einer Klasse mit einem Attribut und einer Operation, die wiederum nur einen Parameter besitzt, dargestellt. Weiterhin wird eine Assoziation erzeugt, wobei vernachlässigt wird, dass diese Assoziation natürlich zwei Assoziationsenden besitzen muss und die Assoziation natürlich auch bei zwei CGOClass-Objekten in die Assoziationsliste aufgenommen werden muss. Da das Diagramm so schon umfangreich genug ist, wurde auf die Darstellung von Unterpackages ganz verzichtet. Außerdem wurde die ***Superclass-Collection*** bei einem CGOClass-Objekt nicht berücksichtigt.

4.4.2 Zerstörung der COM-Objekte

Damit bei der Zerstörung der einzigen Instanz der GOApplication-Komponente auch wieder alle Objekte des Schnittstellenmodells zerstört werden, die in der Initialisierungsphase erzeugt wurden, muss in den Destruktoren jeweils ein `Release` auf alle enthaltenen COM-Objektreferenzen aufgerufen werden.

Als Beispiel sei hier der Destruktor von `CGOModel` aufgeführt.

```
CGOModel::~CGOModel()
{
    if (pClassCollection)
        pClassCollection->Release();

    if (pPackageCollection)
        pPackageCollection->Release();

    if (pLogicalView)
        pLogicalView->Release();

    if (pAssociationCollection)
        pAssociationCollection->Release();
}
```

Eine ***Ausnahme*** stellt lediglich die Klasse **`CGORole`** dar.
Für den enthaltenen Zeiger auf das jeweils andere Assoziationsende darf kein `Release` aufgerufen werden.

Für diesen Zeiger wird auch bei der Initialisierung absichtlich kein `AddRef` aufgerufen, um keine Blockierung der Programmausführung (***deadlock***) zu erzeugen. Aufgrund der gegenseitigen Referenz der beiden Rollen einer Assoziation aufeinander würden die Referenzzähler für beide Rollen-Objekte auf »2« inkrementiert werden, womit eine automatische Zerstörung der Objekte durch das Aufrufen von `Release` nicht mehr ausgelöst werden könnte. Aus diesem Grund wird in diesem Fall die Lebenszeit der Rollen an die Lebenszeit der zugehörigen Assoziation geknüpft.

4.4.3 Lesender und schreibender Zugriff auf Eigenschaften

Beim lesenden Zugriff auf eine Eigenschaft muss zunächst geprüft werden, ob der übergebene Zeiger für den Ausgabeparameter gültig, d.h. nicht NULL ist. Sollte der Zeiger nicht gültig sein, wird als HRESULT-Wert der COM-Operation E_POINTER zurückgegeben. Andernfalls wird der entsprechende Eigenschaftswert aus dem Fachkonzept bestimmt und an die Adresse des Zeigers geschrieben. Um die erfolgreiche Ausführung zu dokumentieren liefert die Operation S_OK zurück.

Bei Variablen vom Typ CString muss eine Typ-Konvertierung in BSTR vorgenommen werden (siehe Kapitel 3.1.5.2). Dazu dient die Funktion AllocSysString.

Als Beispiel sei hier der lesende Zugriff auf die Eigenschaft Name der Komponente GOElement aufgeführt:

```
STDMETHODIMP CGOElement::get_Name(BSTR* pVal)
{
    if (pVal)
    {
        CString text = theMModelElement->getname();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}
```

Für das Schreiben von Eigenschaftswerten besitzt GO eine Reihe von sog. ***Action***-Klassen. Diese dokumentieren jede Werteänderung im Modell und erlauben damit einen Rückgängig-/Wiederherstellen-Mechanismus. Eigenschaftswerte dürfen deshalb nicht direkt ins Fachkonzept geschrieben werden, sondern müssen über Operationen wie z.B. SetChangedName der zugehörigen ***Action***-Klassen gesetzt werden. Durch den Aufruf der Operation Do wird die zuvor eingetragene Änderung auch im Fachkonzept durchgeführt.

Der Anwender soll jedoch alle über COM vorgenommenen Änderungen auf einmal wieder rückgängig machen können. Deshalb werden alle ***Actions*** zu einer ***ActionGroup*** zusammengefasst. Jede ***Action*** muss dafür mittels AddSubAction dieser ***ActionGroup*** hinzugefügt werden.

Nachfolgend sei noch einmal die Eigenschaft Name der Komponente GOElement als Beispiel verwendet, um den schreibenden Zugriff zu zeigen:

```
STDMETHODIMP CGOElement::put_Name(BSTR newVal)
{
    CString text = CString(newVal);

    ActionEditMModelElement* pEdit = new ActionEditMModelElement(theMModelElement);
    pActionGroup->AddSubAction(pEdit);

    pEdit->SetChangedName(text);
    pEdit->Do();

    return S_OK;
}
```

4.4.4 Rückgabe eines Schnittstellenzeigers

Als Beispiel für die Rückgabe eines Schnittstellenzeigers diene die Operation GetCurrentModel der Klasse CGOOperation.

```
STDMETHODIMP CGOApplication::GetCurrentModel(IGOModel** ppModel)
{
    CComQIPtr<IGOModel> d_pGOModel (pGOModel);  // Calls QI

    if (d_pGOModel)
    {
        return d_pGOModel.CopyTo(ppModel);      // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück

    return E_FAIL;
}
```

Das Makro STDMETHODIMP deklariert einen HRESULT-Wert als Rückgabeparameter, der für jede Operation zurückgegeben werden muss.

In der IDL wurde ein [out, retval]-Parameter ppModel als Zeiger auf einen IGOModel-Schnittstellenzeiger deklariert. Diesen Zeiger gilt es zu füllen.

Um dies zu erreichen, wird zunächst ein ***Smart-Pointer*** d_pGOModel erzeugt und mit pGOModel, dem Zeiger auf das CComObject<CGOModel>, initialisiert. Bei dieser Initialisierung wird intern ein QueryInterface aufgerufen, um an den gewünschten Schnittstellenzeiger zu gelangen.

Wenn diese Initialisierung erfolgreich war, wird mittels der Operation CopyTo der ***Smart-Pointer*** d_pGOModel in den Zeiger auf den IGOModel-Schnittstellenzeiger ppModel kopiert.
Damit wird die Lebenszeit des COM-Objektes, welches über seinen Schnittstellenzeiger vom ***Client*** angesprochen werden kann, von der Lebenszeit des ***Smart-Pointers*** abgekoppelt. Wichtig ist auch, dass CopyTo automatisch ein AddRef auf den neuen Schnittstellenzeiger ausführt, damit das COM-Objekt nicht sofort wieder zerstört wird. Erst wenn der ***Client*** Release aufruft und somit der Referenzzähler des Objektes wieder 0 erreicht, zerstört sich das Objekt selbst.

Ich weise noch einmal darauf hin, dass keine Aufrufe von QueryInterface oder Release für den ***Smart-Pointer*** notwendig sind, da diese Aufrufe automatisch ausgeführt werden.
Die Lebenszeit des ***Smart-Pointers*** beschränkt sich hierbei auf die Ausführungszeit obiger Operation (lokale Deklaration).

4.5 Benutzung der COM-Schnittstellen

Bevor GO als Automatisierungsserver verwendet werden kann, müssen die mit dem Modul »GO.exe« ausgelieferten COM-Komponenten in der ***Windows Registry*** registriert werden.

Um dies zu erreichen, wird die Anwendung »GO.exe« mit der Option »***RegSever***« aufgerufen.
Ein entsprechender Aufruf in der DOS-Kommandozeile könnte folgendermaßen aussehen:

```
GO -RegServer
```

Diese Registrierung sollte beim Deinstallieren von GO auch wieder rückgängig gemacht werden. Dazu wird »GO.exe« mit der Option »***UnregServer***« aufgerufen.

```
GO -UnregServer
```

Diese Registrierungsaufrufe sollten vor einer zukünftigen Auslieferung des GO in ein Installations-***Script*** übernommen werden.

Nach der Registrierung kann GO auf herkömmliche Art und Weise gestartet und benutzt werden. Das Vorhandensein der Automatisierungsschnittstelle macht sich zunächst nicht bemerkbar.

Zum Testen der implementierten COM-Komponenten mit ihren Schnittstellen wurde von mir ein ***Dummy-Specifier*** in Visual Basic implementiert, der das Auslesen eines Modells mit seinen Eigenschaften und das Verändern einzelner Eigenschaften ermöglicht.

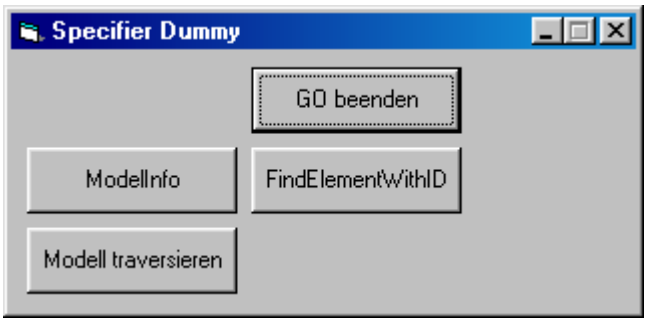

Abb. 4.5-1: Hauptfenster des Dummy-Specifiers

Um die GO-Komponenten in Visual Basic benutzen zu können, muss die Typbibliothek »Go.tlb« bei den Verweis-Einstellungen in Visual Basic eingetragen werden.

Der folgende Quelltext zeigt, wie einfach der Zugriff auf die COM-Komponenten in Visual Basic zu realisieren ist.

```
Private theApp As GOApplication 'Declared only in this module!!!
Private theModel As GOModel
Private logicalView As GOPackage

Set theApp = New GOApplication
Set theModel = theApp.GetCurrentModel()
Set logicalView = theModel.GetLogicalView()
```

Durch den Aufruf von »`New GOApplication`« wird das GO-Schnittstellenmodell zu einer bereits laufenden GO-Anwendung initialisiert. Es wird keine neue GO-Instanz erzeugt.

Alle weiteren COM-Objekte sind über die entsprechenden COM-Schnittstellen ebenfalls leicht zu erreichen.

Nach dem Anklicken von »Modell traversieren« im Hauptfenster des ***Dummy-Specifiers*** öffnet sich ein Dialogfenster, in dem alle Modellelemente des jeweiligen ***Packages*** mit all seinen Eigenschaften angezeigt wird.

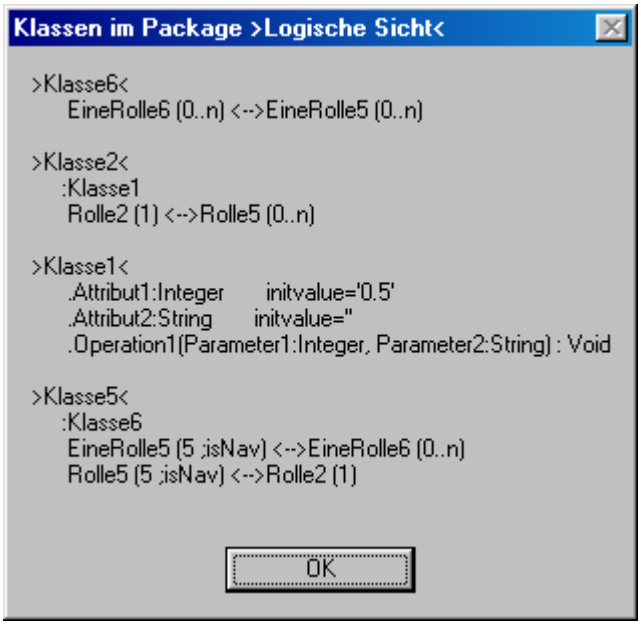

Abb. 4.5-2: Traversier-Dialog des Dummy-Specifiers

Im Anhang E.5 ist der vollständige Quelltext des ***Dummy-Specifier*** aufgeführt.

5 Statistik

In diesem Kapitel soll kurz auf den Arbeitsaufwand und den Umfang der in dieser Studienarbeit erstellten Software eingegangen werden.

Aus der folgenden Tabelle sind die LOC (***Lines Of Code***) der in dieser Arbeit neu entstandenen Dateien ersichtlich:

Datei	*LOC*
GO.idl	360
CGOApplication.h	60
CGOApplivation.cpp	67
CGOCollection.h	62
CGOCollection.cpp	98
CGOElement.h	78
CGOElement.cpp	135
CGOPackage.h	62
CGOPackage.cpp	228
CGOModel.h	60
CGOModel.cpp	232
CGOClass.h	76
CGOClass.cpp	435
CGOAttribute.h	57
CGOAttribute.cpp	112
CGOOperation.h	68
CGOOperation.cpp	242
CGOParameter.h	50
CGOParameter.cpp	58
CGOAssociation.h	59
CGOAssociation.cpp	143
CGORole.h	67
CGORole.cpp	144
Summe	2953

Tabelle 4.5-1: LOC nach Dateien

Darüber hinaus sind noch Ergänzungen und Änderungen in den vorhandenen Quellcode-Dateien im Gesamtumfang von ca. 230 LOC erfolgt.

Diese für die kurze Zeit hohe Anzahl von LOC muss jedoch relativiert werden, da einerseits Rümpfe bereits durch Rational Rose generiert wurden und darüber hinaus viele Schnittstellen-Operationen ähnlich implementiert werden konnten (***Copy & Paste***).

Die nachfolgende Tabelle gibt die Verteilung des Arbeitsaufwands auf unterschiedliche Tätigkeiten wieder:

Tätigkeit	***Arbeitsaufwand***	
	in Stunden	***in %***
Planung (incl. Besprechungen mit oTRIs, Einarbeitung in COM und ATL)	95	27%
Schnittstellen-Entwurf	50	14%
Schnittstellen-Implementierung	100	29%
Testen	25	7%
Schreiben der Arbeit	80	23%
Summe	350	100%

Tabelle 4.5-2: Arbeitsaufwand nach Tätigkeiten

Bezogen auf den Arbeitsaufwand von 44 Mitarbeitertagen ergibt sich LOC/MT zu 72.
Zieht man die Zeit für das Schreiben der Studienarbeit ab, so reduzieren sich die Mitarbeitertage auf 34. Rein rechnerisch ergibt sich LOC/MT somit zu 94.

6 Ausblick

Die in dieser Studienarbeit identifizierten COM-Komponenten und COM-Schnittstellen wurden auch implementiert.
Im Rahmen der verfügbaren Zeit konnten jedoch nicht alle Probleme gelöst werden.
So sind einige Schnittstellen-Operationen vorgesehen, aber nur leer implementiert worden.

```
CGOAttribute::get_Static()
CGOAttribute::put_Static()
CGOAttribute::get_Derived()
CGOAttribute::put_Derived()
```

Der Zugriff auf die Eigenschaften »Static« und »Derived« konnte nicht implementiert werden, da Klassenattribute und abgeleitete Attribute von GO momentan noch nicht unterstützt werden.

```
CGOElement::get_Stereotype(...)
CGOElement::put_Stereotype(...)
```

Stereotypes werden von GO grundsätzlich noch nicht unterstützt.

```
CGOElement::get_Property(...)
CGOElement::put_Property(...)
```

Tagged Values sind zwar im Fachkonzept hinzugefügt worden, jedoch existieren noch keine Zugriffsklassen (***Action***), um die Eigenschaften schreiben zu können. Weiterhin fehlt noch eine Möglichkeit, die relevanten JANUS-Eigenschaften in GO einzulesen und den einzelnen Modellelementen (Klasse, Attribut, Assoziation, ...) zuzuordnen.

```
CGOOperation::AddParameter(...)
```

Beim Hinzufügen von Parametern wird die Positionsangabe noch nicht unterstützt. Dies liegt daran, dass die Zugriffsklassen (***Action***) des GO eine solche Positionsangabe nicht verwenden.

Darüber hinaus wird das GO ständig weiterentwickelt werden, weitere Diagramme und Funktionalitäten werden hinzukommen. Andere Diagramme als das Klassendiagramm und weitere Funktionen sollen in Zukunft eventuell ebenfalls über COM zur Verfügung gestellt werden, um eine vollständige Fernsteuerung der Anwendung zu ermöglichen. Wenn man dies erwägt, muss intensiv überlegt werden, ob die in dieser Arbeit vorgestellte Lösung, bei der COM-Komponenten das Fachkonzept nur kapseln das Mittel der Wahl ist. Aus Effizienzgründen (momentan wird ein Modell von COM-Objekten zusätzlich zum Fachkonzeptmodell erzeugt) sollte eine Umstellung des Fachkonzeptes auf COM-Komponenten erwogen werden. Der dadurch entstehende enorme Arbeitsaufwand muss natürlich gegen die Effizienzsteigerung abgewogen werden.

Auch ist im Hinblick auf eine einfachere Benutzung der Schnittstellen in Visual Basic die Benutzung der Wiederverwendungs-Mechanismen Einschließung und Aggregation zu prüfen, die im Rahmen dieser Arbeit nicht zum Einsatz gekommen sind.

Anhang A – Lastenheft

1 Zielbestimmung

Das am Lehrstuhl für Software-Technik entwickelte Software-Werkzeug GO (Generating Objects) soll zu einem Automatisierungsserver erweitert werden.
UML-Klassenmodelle, die mit GO erstellt wurden, sollen über die Automatisierungsschnittstelle von anderen Software-Komponenten ausgelesen, bearbeitet und erweitert werden können.

2 Produkteinsatz

Ziel ist es, die Kommunikation zwischen GO und dem JANUS-Specifier des Software-Generierungswerkzeuges JANUS von der Firma oTRIs über COM zu ermöglichen.
Zielgruppe: Anwendungsentwickler

3 Produktfunktionen

/LF 10/ Alle für die Kommunikation notwendigen Klassen müssen als COM-Komponenten mit den notwendigen Schnittstellen zur Verfügung gestellt werden.

/LF 20/ UML-Klassenmodelle müssen in Ihrer Struktur über COM ausgelesen werden können.

/LF 30/ Die für den JANUS-Specifier relevanten Modelleigenschaften müssen über COM ausgelesen und evtl. bearbeitet werden können.

/LF 40/ Für einige spezielle JANUS-Modelleigenschaften muss GO um eine Datenhaltung für »Tagged Values« erweitert werden.

/LF 50/ Diese speziellen JANUS-Modelleigenschaften sollen in GO angezeigt und bearbeitet werden können.

4 Produktdaten

/LD 10/ Es sind die relevanten Daten zu den JANUS-Modelleigenschaften als »Tagged Values« zu speichern.

5 Produktleistungen

/LL 10/ Die COM-Automatisierungsschnittstellen müssen vom JANUS-Specifier über Visual Basic 6 angesprochen werden können.

6 Qualitätsanforderungen

Produktqualität	*sehr gut*	*gut*	*normal*	*nicht relevant*
Funktionalität		X		
Zuverlässigkeit		X		
Benutzbarkeit		X		
Effizienz			X	
Änderbarkeit			X	
Übertragbarkeit		X		

7 Ergänzungen

Die speziellen JANUS-Modelleigenschaften werden nur mit einer Eigenschaftsbezeichnung und einem Wert beschrieben.

Anhang B – Pflichtenheft

1 Zielbestimmung

Das am Lehrstuhl für Software-Technik entwickelte Software-Werkzeug GO (Generating Objects) soll zu einem Automatisierungsserver erweitert werden.
UML-Klassenmodelle, die mit GO erstellt wurden, sollen über die Automatisierungsschnittstelle von anderen Software-Komponenten ausgelesen, bearbeitet und erweitert werden können.

1.1 Musskriterien

- Erstellung der COM-Komponenten und Schnittstellen zum Auslesen der UML-Modelle aus GO.
- Die Komponente muss unter Windows mit Visual Basic 6 verwendbar sein.

1.2 Wunschkriterien

- Die für JANUS relevanten zusätzlichen Modelleigenschaften sollen über ein noch festzulegendes Format in GO eingelesen werden.
- Diese zusätzlichen Modelleigenschaften sollen in GO angezeigt und editiert werden können.

1.3 Abgrenzungskriterien

- Es sollen nur die für den JANUS-Specifier relevanten Daten nach außen zur Verfügung gestellt werden. Dies beinhaltet insbesondere auch nur das Klassenmodell von GO, nicht jedoch z.B. das Prozessmodell.

2 Produkteinsatz

Die COM-Schnittstellen werden für die Firma oTRIs erstellt.

2.1 Anwendungsbereiche

Ziel ist es, die Kommunikation zwischen GO und dem JANUS-Specifier des Software-Generierungswerkzeuges JANUS von der Firma oTRIs über COM zu ermöglichen.

2.2 Zielgruppen

Primär besteht die Zielgruppe aus Anwendungsentwicklern der Firma oTRIs, die das GO als Analysewerkzeug für den JANUS-Generator einsetzen wollen. Die COM-Schnittstelle des GO wird dabei von einer Visual-Basic-Komponente angesprochen, die von oTRIs erstellt wird.
Als potentielle Zielgruppe anzusehen sind jedoch auch alle anderen Anwendungsentwickler, da offene Schnittstellen implementiert werden, die jeder Anwendungsentwickler nutzen kann.

2.3 Betriebsbedingungen

Entwicklungsumgebung, Büroumgebung

3 Produktumgebung

Das Produkt läuft auf einem Arbeitsplatz-PC als Einzelplatzversion.

3.1 Software

- Betriebssystem: Microsoft Windows 95/98/2000/NT 4
- Der Anwendungsentwickler benötigt eine der folgenden Programmiersprachen: Microsoft Visual C++ 6, Microsoft Basic 6 oder ähnliche Programmiersprachen, die auf COM-Objekte über ihre Schnittstellen zugreifen können.

3.2 Hardware

Handelsüblicher PC

3.3 Orgware

keine

3.4 Produkt-Schnittstellen

Über die COM-Schnittstellen soll der JANUS-Specifier auf das UML-Klassenmodell zugreifen können, wie es mit dem Software-Analysewerkzeug Rose der Firma Rational bereits möglich ist. Die Verwendung der Schnittstellen soll möglichst analog erfolgen können.

4 Produktfunktionen

/F 10/ Es ist ein COM-Automatisierungsserver zu erstellen.

/F 20/ Um Introspektion zu ermöglichen, ist eine Typbibliothek zu erstellen.

/F 30/ Um UML-Klassenmodelle auslesen zu können, müssen die folgenden Fachkonzeptklassen durch COM-Komponenten repräsentiert werden:

- CApplication
- MModelElement
- MPackage
- MModel
- MClass
- MAttribute
- MOperation
- MParameter
- MAssociation
- MAssociationEnd

Die zugehörigen COM-Komponenten sollen folgendermaßen bezeichnet werden:

- CGOApplication
- CGOElement
- CGOPackage
- CGOModel
- CGOClass
- CGOAttribute
- CGOOperation
- CGOParameter
- CGOAssociation
- CGORole

/F 40/ Für die Bearbeitung einer Gruppe von Objekten muss die COM-Komponente CGOCollection implementiert werden.

/F 50/ CGOApplication muss einen Zugriff auf das aktuelle Modell ermöglichen.

/F 60/ CGOElement repräsentiert nicht weiter spezifizierte Modell-Elemente. CGOElement soll lesenden und schreibenden Zugriff auf folgende Eigenschaften ermöglichen:

- ***Name***
- ***Documentation***
- ***Stereotype***

/F 70/ CGOElement soll eine eindeutige ID zurückliefern können.

/F 80W/ CGOElement soll lesenden und schreibenden Zugriff auf »***Tagged Values***« ermöglichen.

/F 90/ CGOPackage soll den Zugriff auf die enthaltenen Unterpackages und Klassen ermöglichen.

/F 100/ CGOPackage soll das Hinzufügen von Unterpackages unterstützen.

/F 110/ CGOModel soll die logische Sicht eines UML-Modell zurückliefern können.

/F 120/ CGOModel soll alle Modellelemente anhand ihrer ID erkennen und zurückgeben können.

/F 130/ CGOClass soll lesenden und schreibenden Zugriff auf die Eigenschaft ***Abstract*** ermöglichen.

/F 140/ CGOClass soll den Zugriff auf die enthaltenen Attribute und Operationen ermöglichen. Außerdem sollen Attribute hinzugefügt und wieder gelöscht werden können.

/F 150/ CGOClass soll Zugriff auf alle Assoziationen und Oberklassen ermöglichen.

/F 160/ CGOAttribute soll lesenden und schreibenden Zugriff auf folgende Eigenschaften ermöglichen:

- ***Type***
- ***Static***
- ***Derived***
- ***InitValue***

/F 170/ CGOOperation soll schreibenden und lesenden Zugriff auf die Eigenschaft ***ReturnType*** erlauben.

/F 180/ CGOOperation soll den Zugriff auf die Parameter der Operation ermöglichen. Außerdem sollen alle Parameter gelöscht und einzelne Parameter hinzugefügt werden können.

/F 190W/ Beim Hinzufügen der Parameter soll eine Einfüge-Position angegeben werden können.

/F 200/ CGOParameter soll schreibenden und lesenden Zugriff auf die Eigenschaft ***Type*** ermöglichen.

/F 210/ CGOAssociation soll den Zugriff auf die beiden Assoziationsenden ermöglichen.

/F 220/ CGORole soll lesenden und schreibenden Zugriff auf die folgenden Eigenschaften ermöglichen:

- ***Cardinality***
- ***Navigable***

/F 230/ CGORole soll den Zugriff auf das zugehörige Klassen-Objekt und das jeweils andere Assoziationsende erlauben.

/F 240/ CGOPackage, CGOClass, CGOAttribute, CGOOperation, CGOParameter, CGOAssociation, CGORole und CGOModel sollen auch die Eigenschaften von CGOElement besitzen (/F 60/,/F 70/,/F 80W/).

/F 250/ CGOCollection repräsentiert eine Gruppe von CGOElement-Objekten. CGOCollection soll den ***nur*** lesenden Zugriff auf die Anzahl der Objekte ermöglichen. Außerdem soll der Zugriff auf die einzelnen Objekte über einen Index möglich sein.

/F 260W/ Diese speziellen JANUS-Modelleigenschaften sollen in GO angezeigt und bearbeitet werden können. /LF 50/

5 Produktdaten

/D 10/ »Tagged Values« sind als Eigenschaftsname und Wert zu speichern. Welche Eigenschaften im Einzelnen zu speichern sind, ist daher nicht relevant. /LD 10/

6 Produktleistungen

/L 10/ Die COM-Automatisierungsschnittstellen müssen vom JANUS-Specifier über Visual Basic 6 angesprochen werden können. / LL 10/

7 Benutzungsschnittstelle

/B 10/ »Rückgängig/Wiederherstellen« soll auch für Änderungen, die über COM am Modell gemacht wurden, unterstützt werden.

/B 20W/ JANUS-spezifische Modelleigenschaften sollen als »Tagged Values« angezeigt und bearbeitet werden können. Dazu ist eine Tabellendarstellung zu wählen.

8 Qualitätsbestimmung

Produktqualität	*sehr gut*	*gut*	*normal*	*Nicht relevant*
Funktionalität				
Angemessenheit		X		
Richtigkeit		X		
Interoperabilität		X		
Ordnungsmäßigkeit		X		
Sicherheit		X		
Zuverlässigkeit				
Reife			X	
Fehlertoleranz		X		
Wiederherstellbarkeit		X		
Benutzbarkeit				
Verständlichkeit		X		
Erlernbarkeit		X		
Bedienbarkeit			X	
Effizienz				
Zeitverhalten			X	
Verbrauchsverhalten			X	
Änderbarkeit				
Analysierbarkeit		X		

Modifizierbarkeit		X	
Stabilität		X	
Prüfbarkeit			X
Übertragbarkeit			
Anpassbarkeit	X		
Installierbarkeit		X	
Konformität			X
Austauschbarkeit		X	

9 Globale Testszenarien/Testfälle

/T 10/ Zum Testen der Automatisierungsschnittstelle ist ein ***Dummy-Specifier*** in Visual Basic zu implementieren.
/T 20/ Allgemeine Kommunikation zwischen GO und ***Dummy-Specifier*** testen.
/T 30/ Auslesen eines GO-Modells über mehrere Hierarchieebenen (***Packages***) hinweg.
/T 40/ Bearbeiten/Verändern von Modelleigenschaften.
/T 50/ Hinzufügen von Modelleigenschaften.
/T 60/ Löschen von Modelleigenschaften.
/T 70/ Überprüfen der Referenzzählung, Löschen von COM-Objekten.

10 Entwicklungsumgebung

10.1 Software

Betriebssystem: Windows 9x/NT
Werkzeuge:

- Microsoft Visual C++ 6.0
- Rational Rose 2000
- Microsoft Visual Basic 6.0 (als Testumgebung)

10.2 Hardware

Handelsüblicher PC

10.3 Orgware

- keine erforderlich -

10.4 Entwicklungs-Schnittstellen

Nicht relevant, da Entwicklungsplattform gleich der Zielplattform ist.

11 Ergänzungen

Anhang C – OOA-Modell

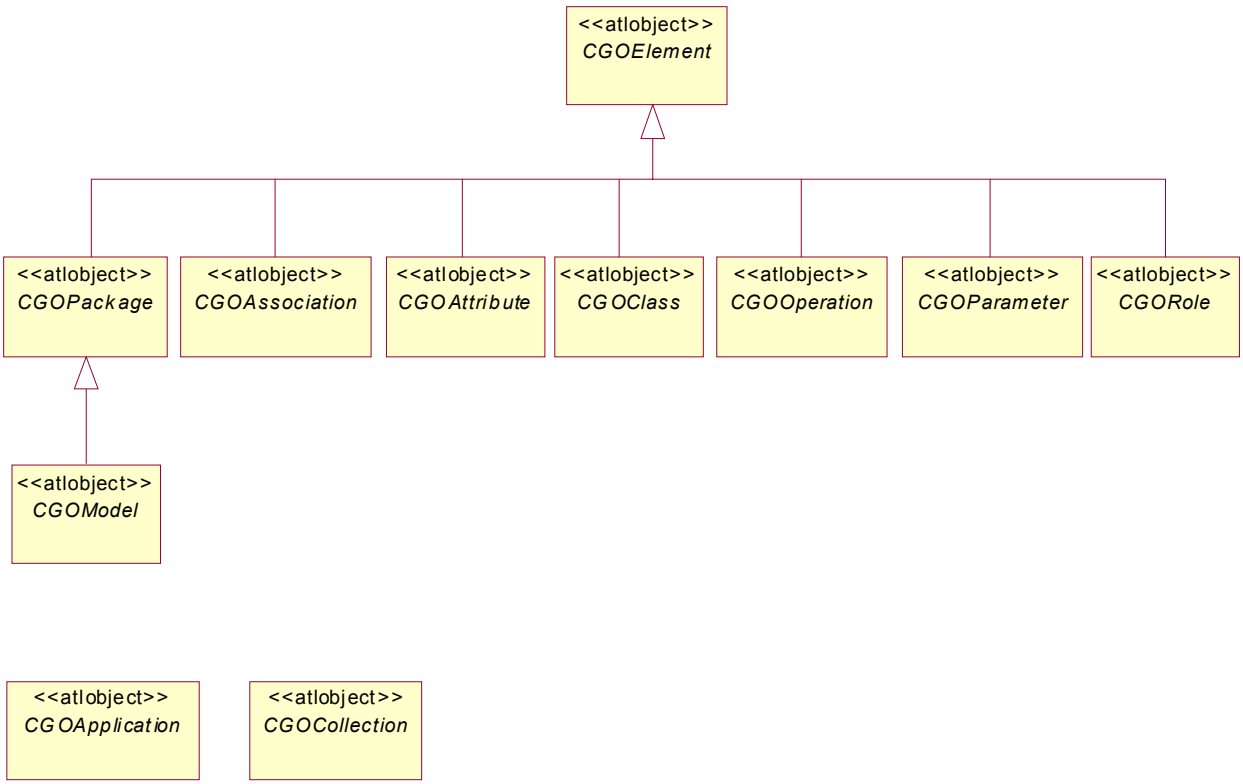

Abb. C-1: UML-Klassendiagramm – Gesamtübersicht (Vererbungshierarchie)

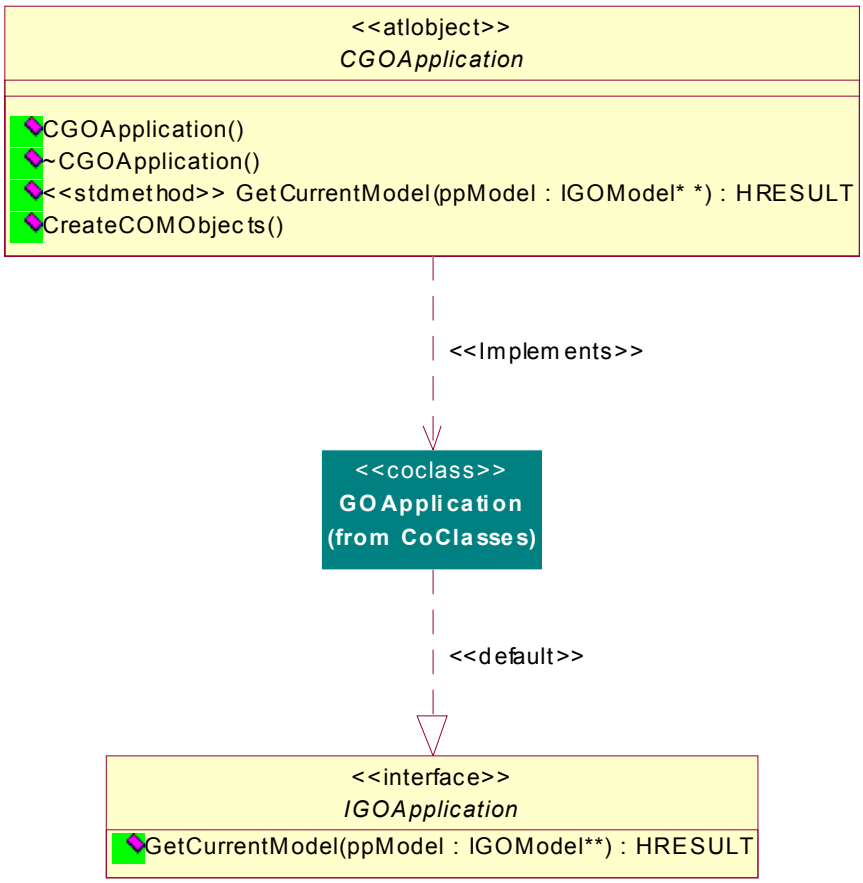

Abb. C-2: UML-Klassendiagramm – GOApplication

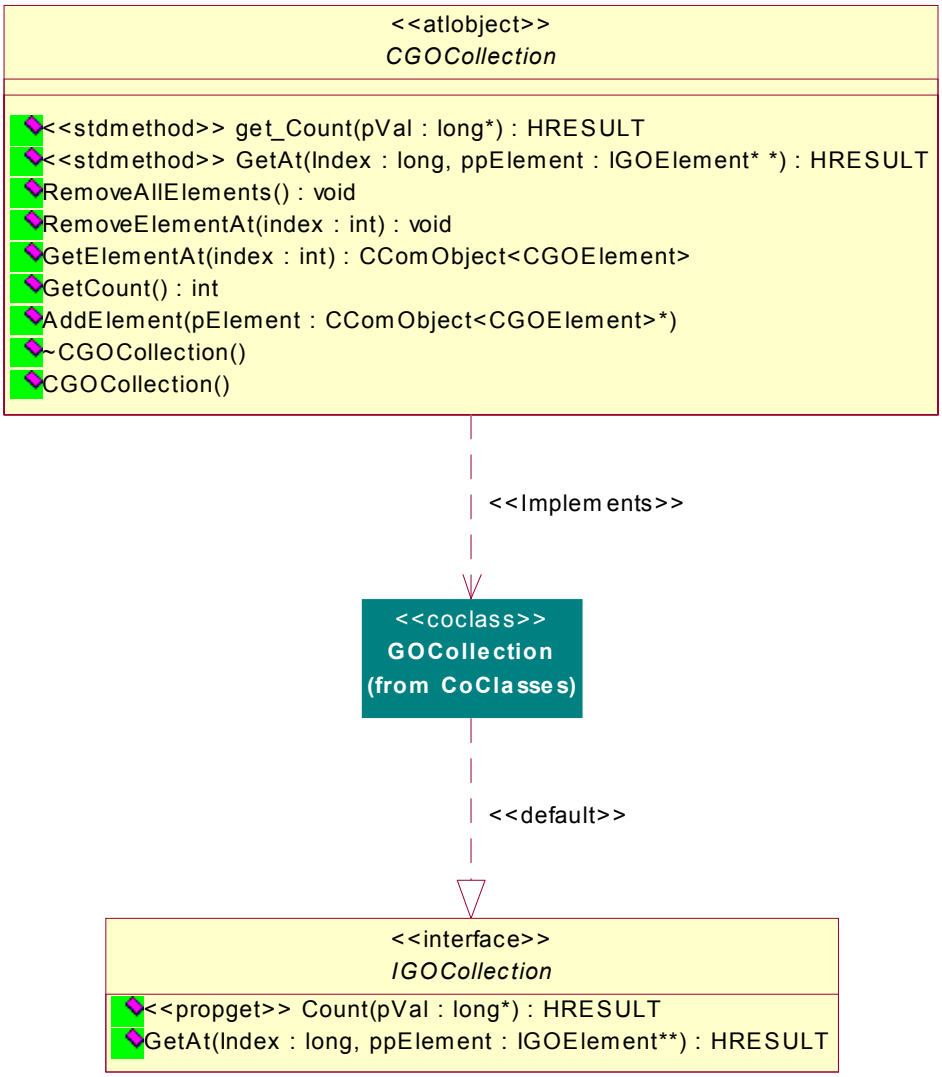

Abb. C-3: UML-Klassendiagramm – GOCollection

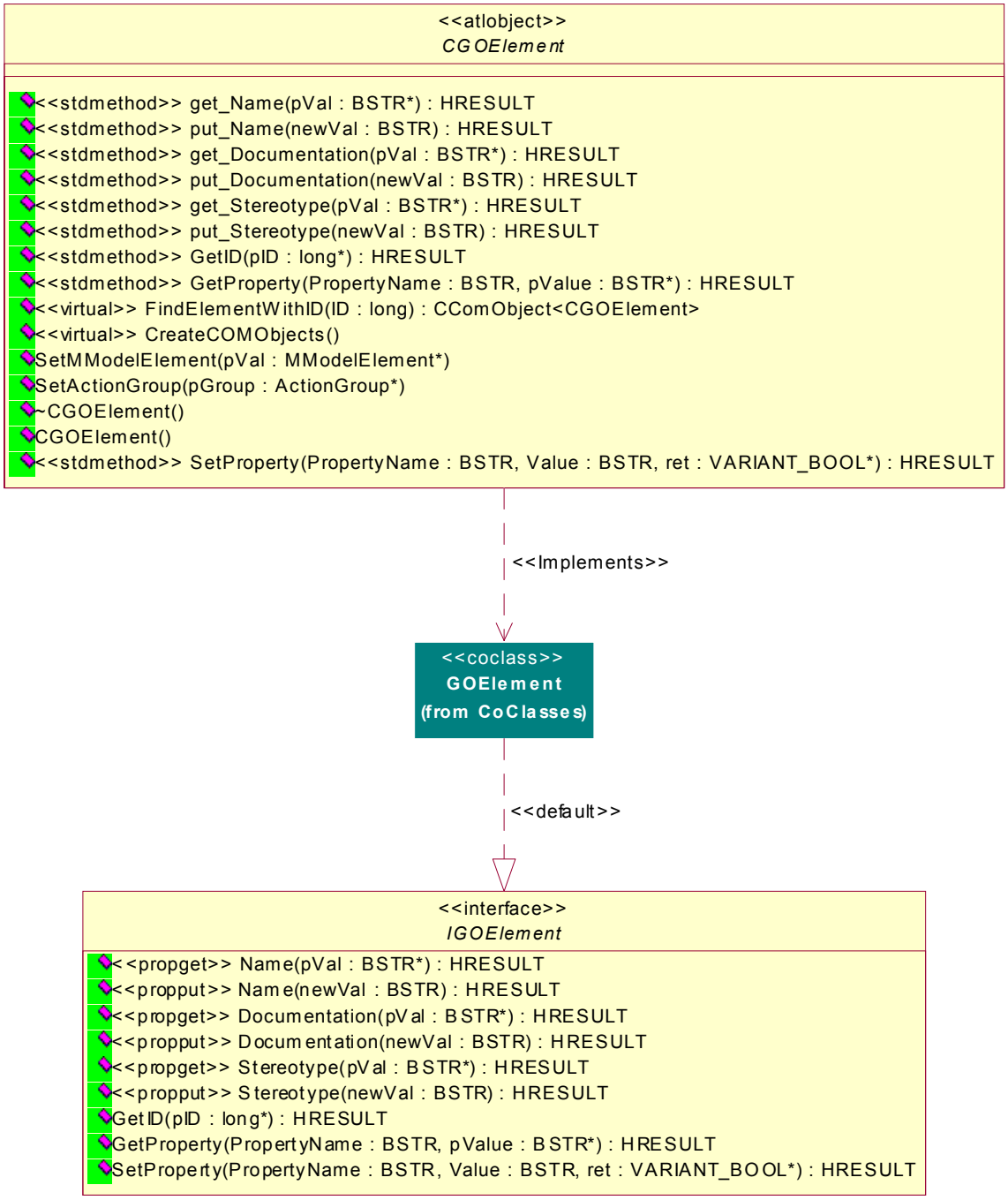

Abb. C-4: UML-Klassendiagramm – GOElement

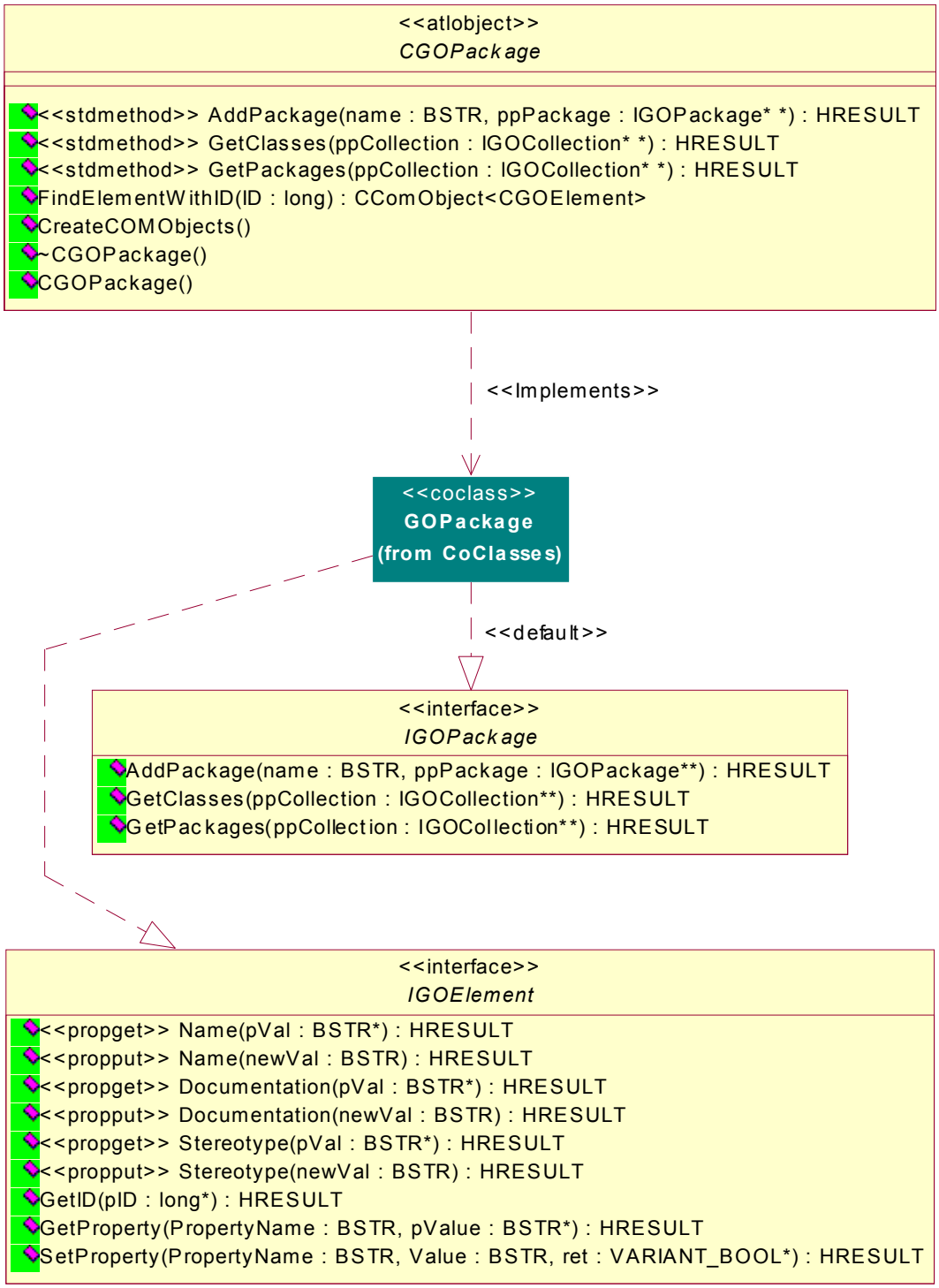

Abb. C-5: UML-Klassendiagramm – GOPackage

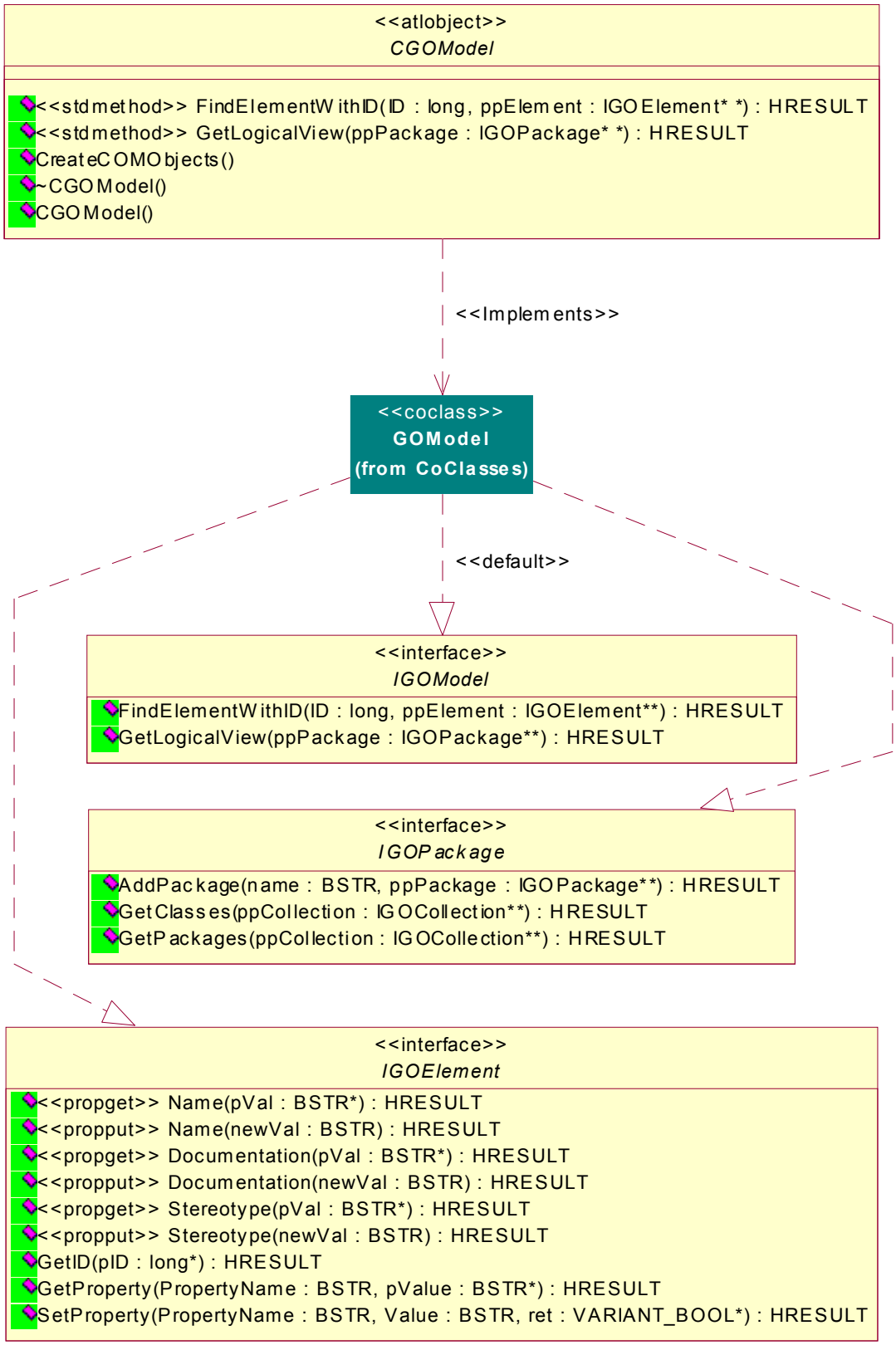

Abb. C-6: UML-Klassendiagramm – GOModel

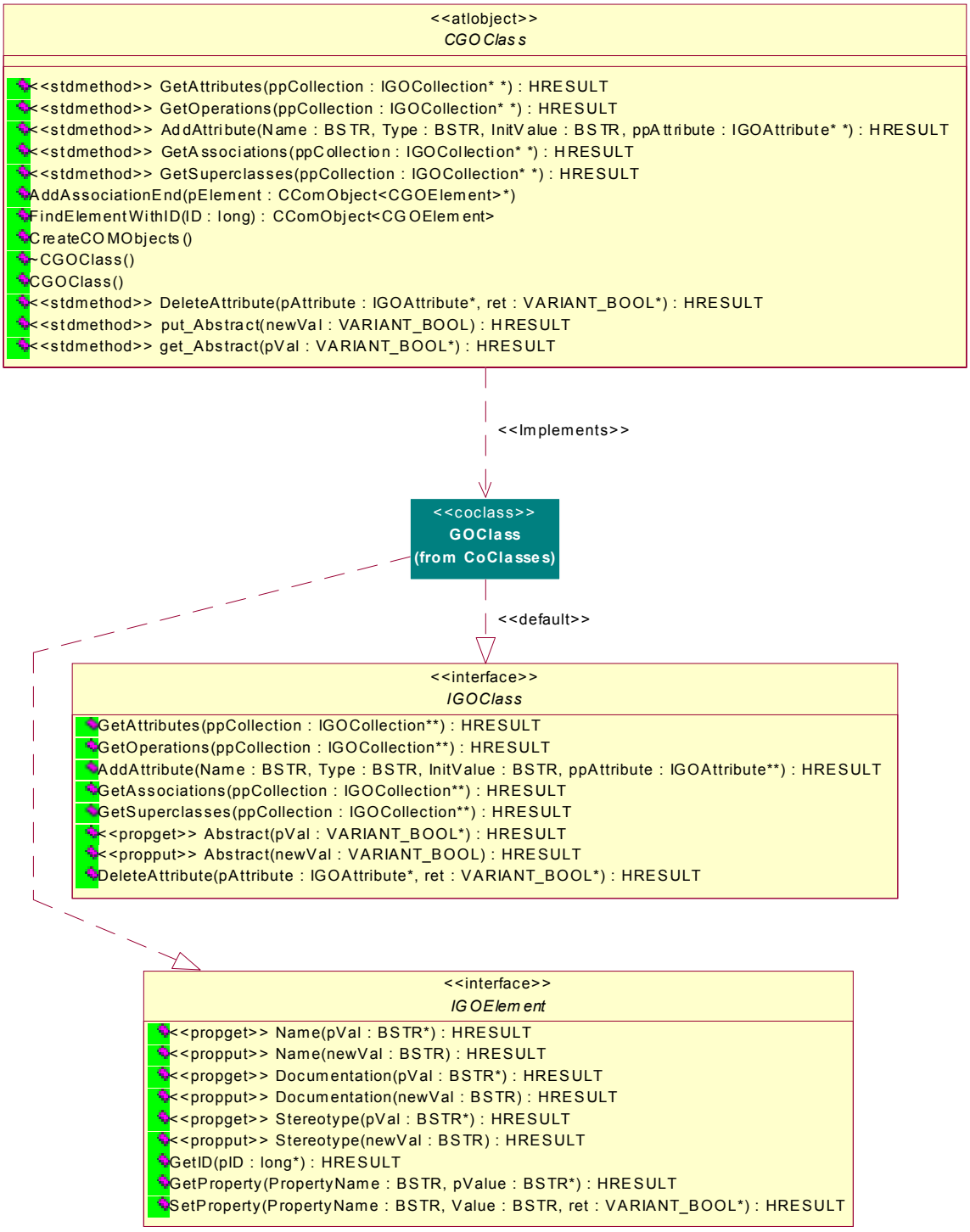

Abb. C-7: UML-Klassendiagramm – GOClass

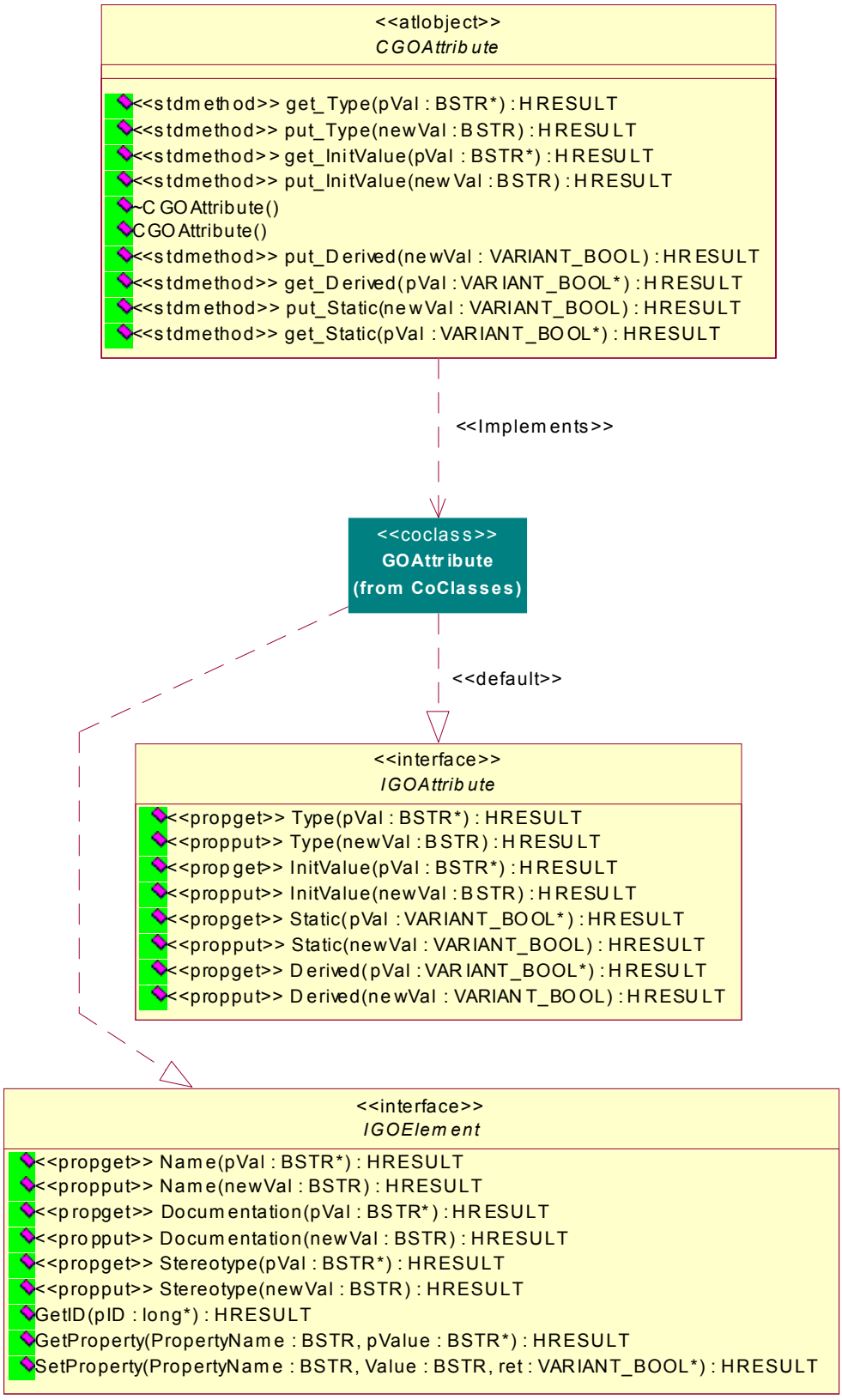

Abb. C-8: UML-Klassendiagramm – GOAttribute

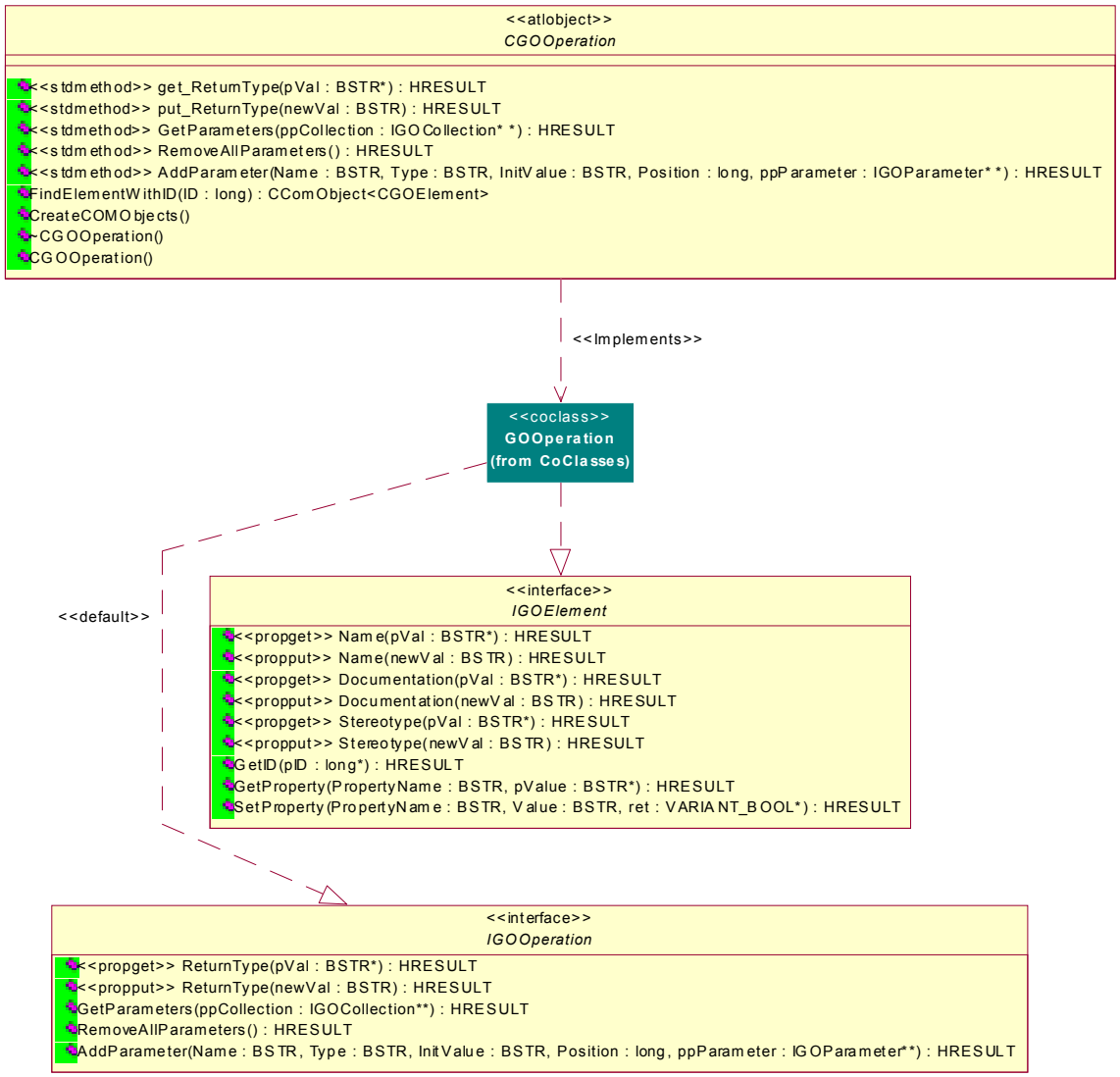

Abb. C-9: UML-Klassendiagramm – GOOperation

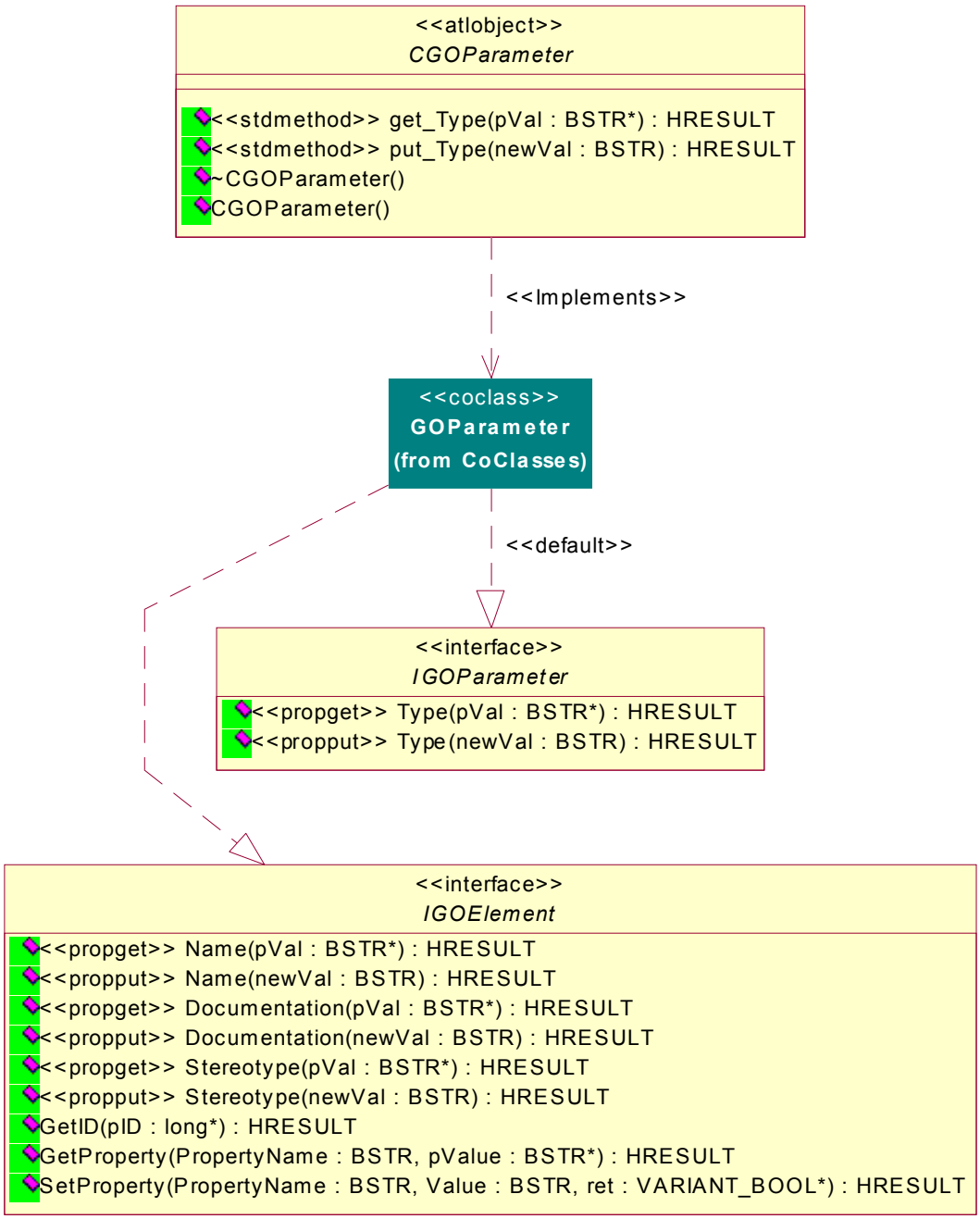

Abb. C-10: UML-Klassendiagramm – GOParameter

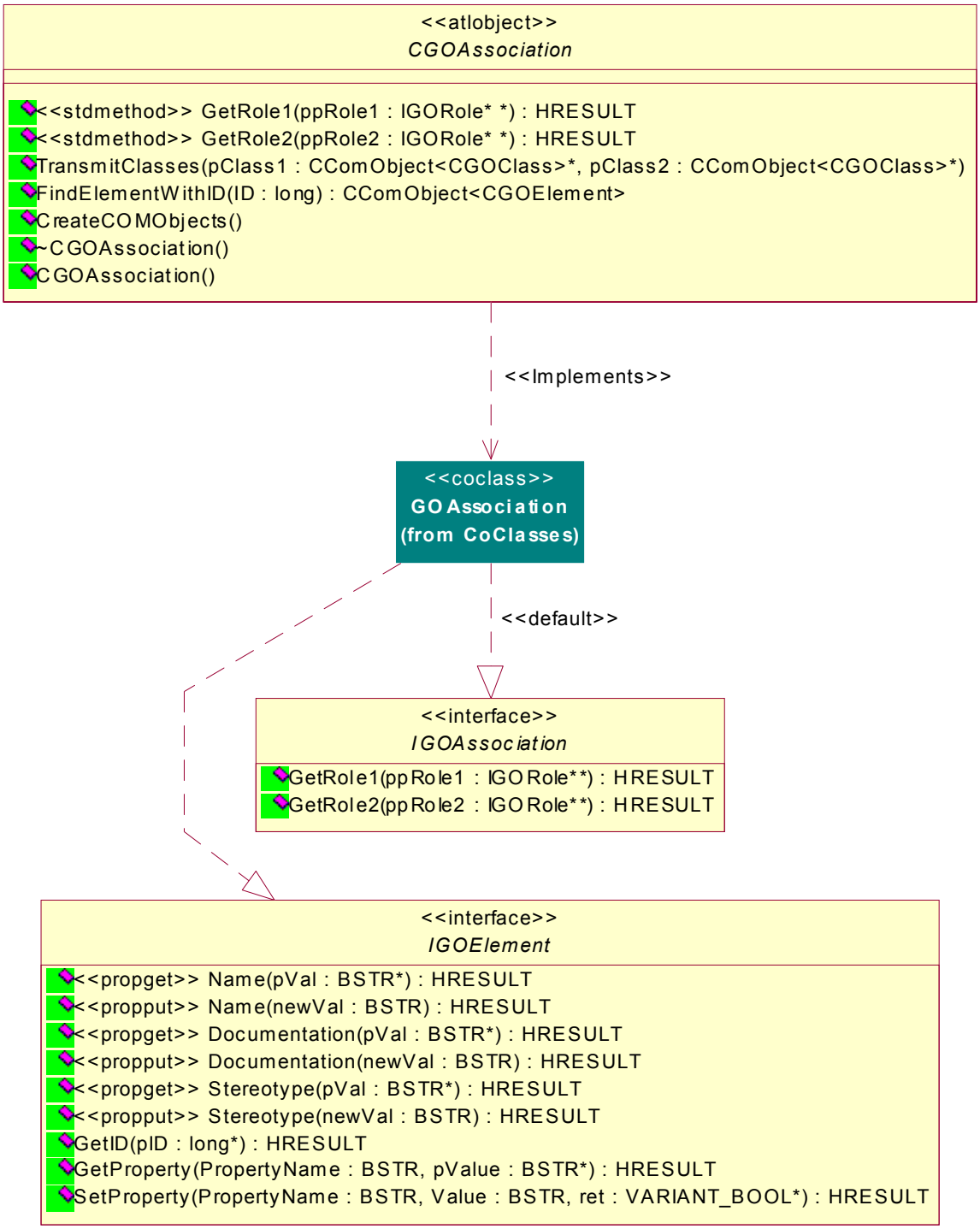

Abb. C-11: UML-Klassendiagramm – GOAssociation

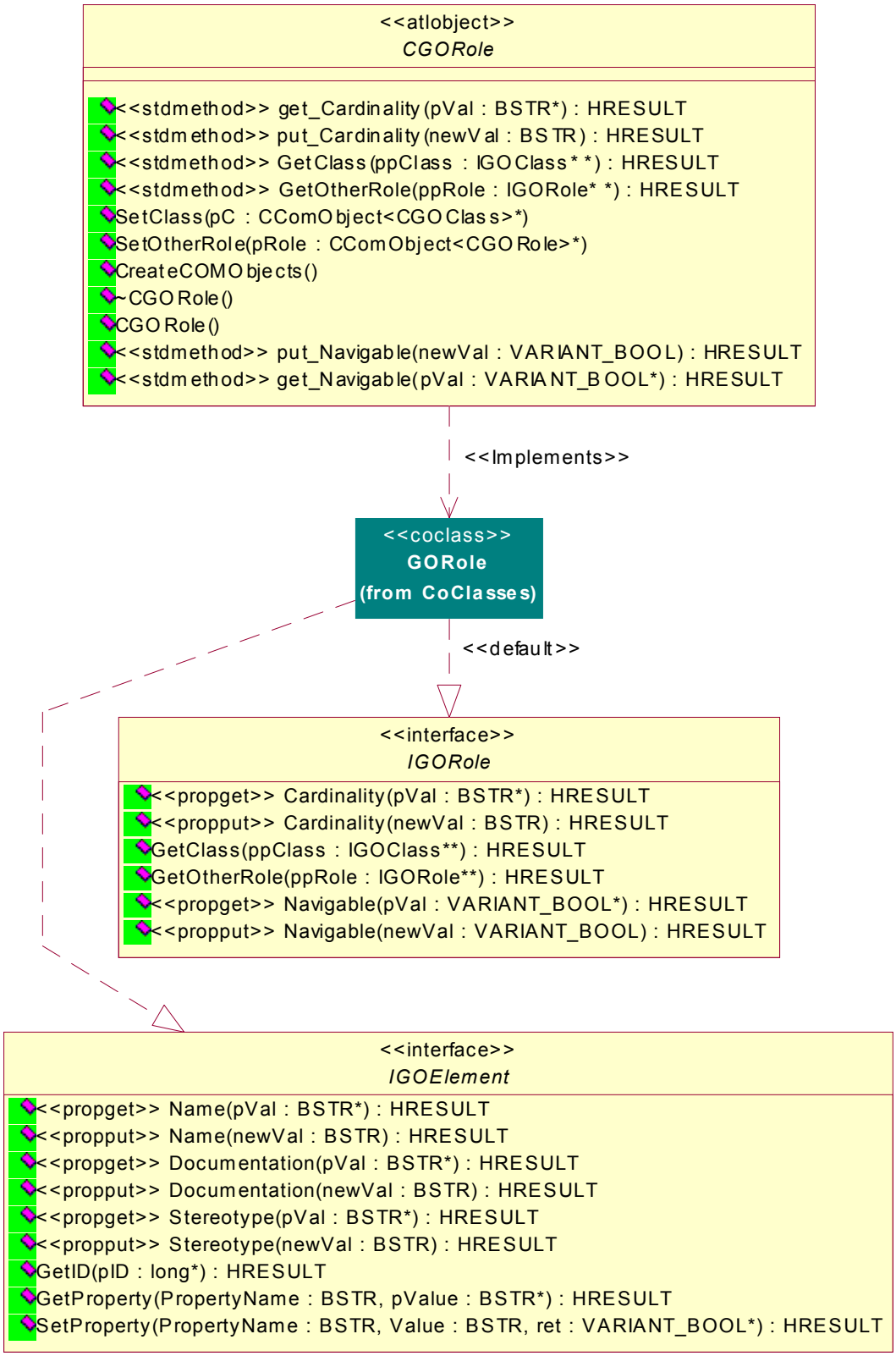

Abb. C-12: UML-Klassendiagramm – GORole

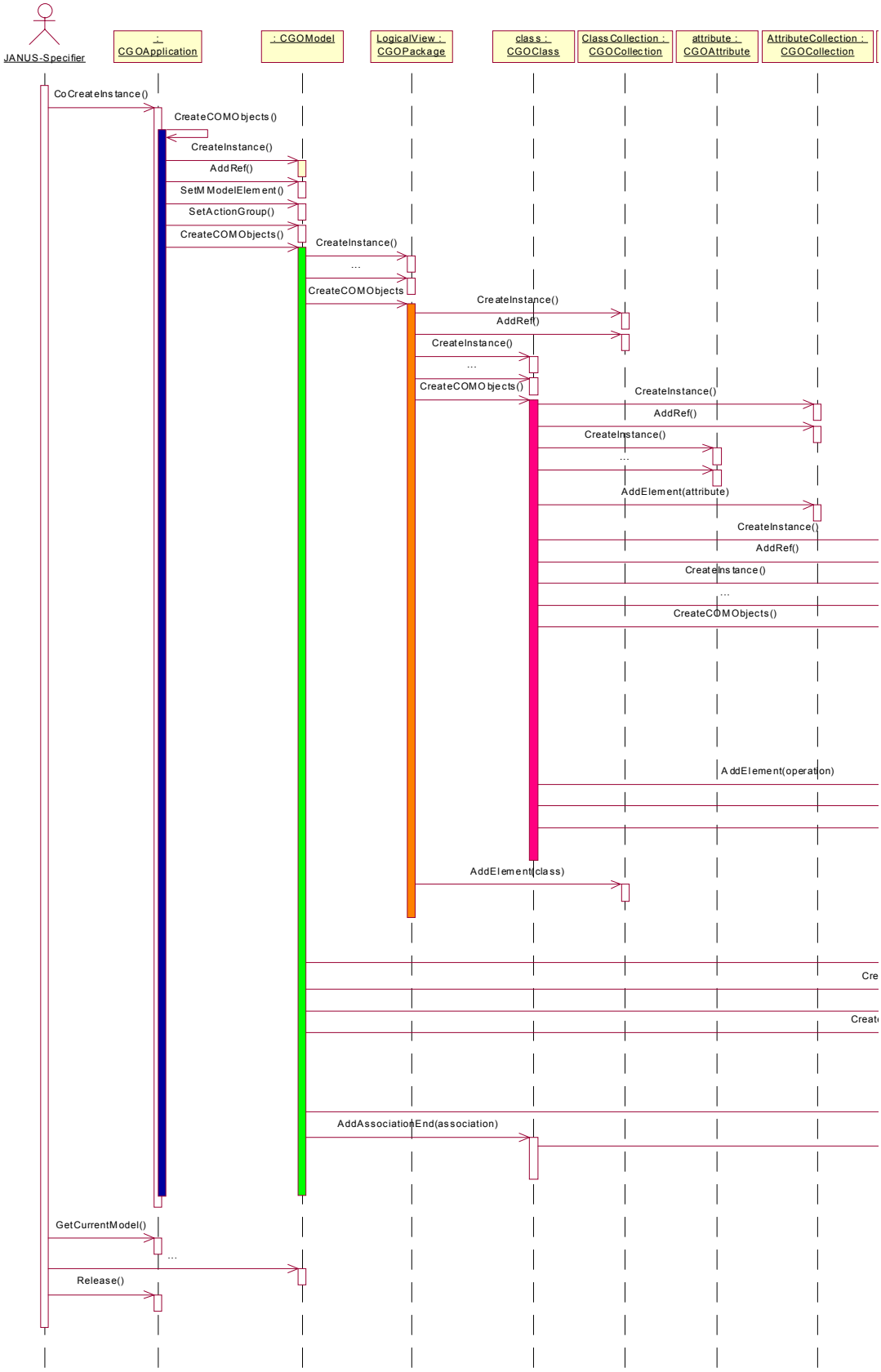

Abb. C-13: UML-Sequenzdiagramm – Initialisierung des Schnittstellenmodells (1. Teil)

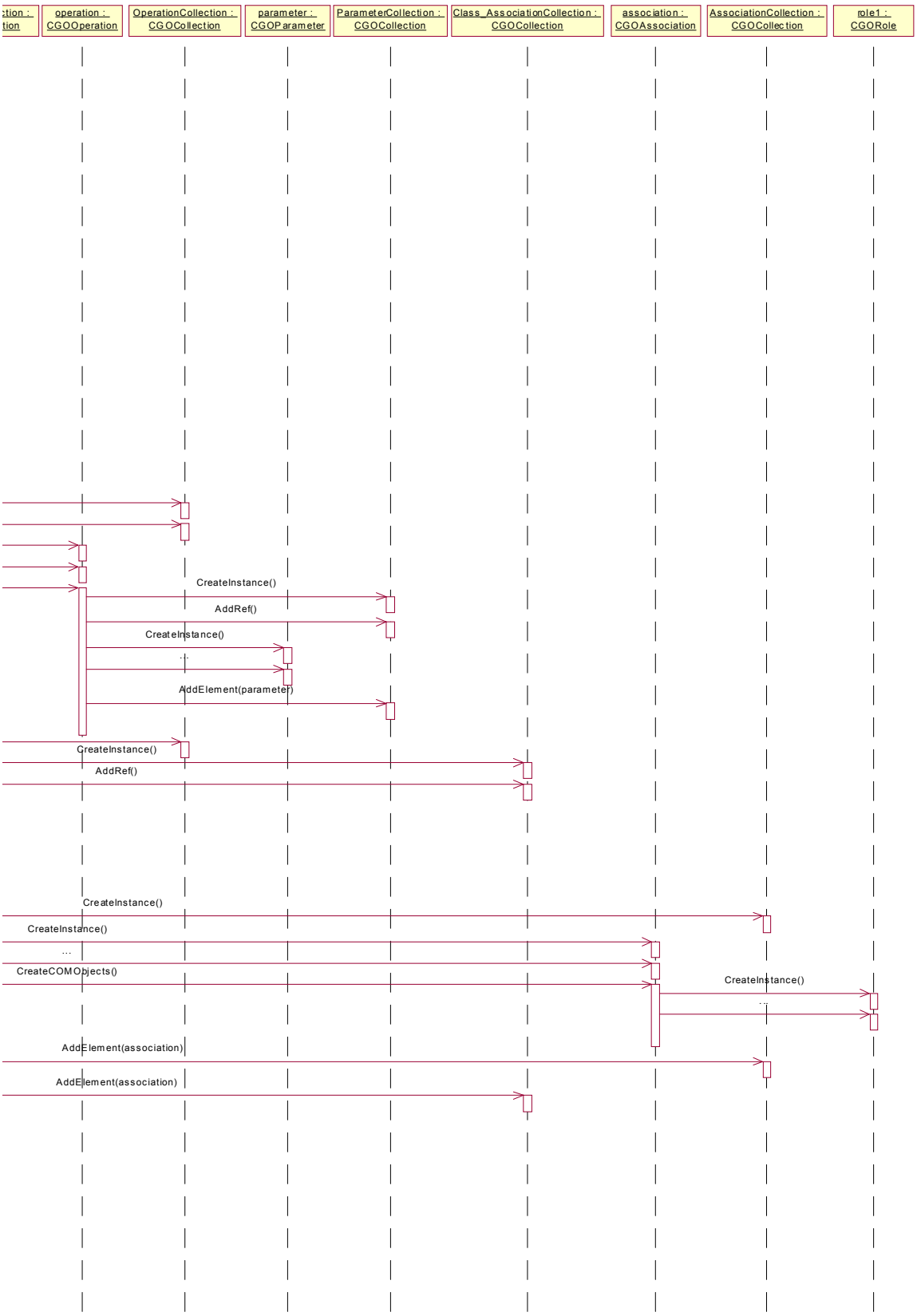

Abb. C-14: UML-Sequenzdiagramm – Initialisierung des Schnittstellenmodells (2. Teil)

Anhang D – Glossar

ATL ***Active Template Library***

Client Ein Client nimmt Dienste eines ***Servers*** (Dienstanbieters) in Anspruch.

COM ***Component Object Model***

COM-Komponente
COM-Komponenten werden in der Literatur in der Regel als »COM-Objekte**«** bezeichnet. (s.COM-Objekt)

COM-Objekt
In der Literatur wird der Begriff »COM-Objekt« synonym zu »COM-Komponente« verwendet. In dieser Studienarbeit wird die Terminologie »COM-Objekt« für ein von einer COM-Komponente erzeugtes Objekt verwendet.

DLL ***Dynamic Link Library***

GUID ***Global Unique Identifier***
GUIDs können mit Hilfe des Programmes »GUIDGEN.exe« erzeugt werden.

IID ***Interface Identifier***

Introspektion
Fähigkeit einer Komponente, Informationen über sich selbst für ihre ***Clients*** zugänglich zu machen.

MFC ***Microsoft Foundation Class Library***

MSVC ***Microsoft Visual C++***

OLE ***object linking and embedding***

RMI ***Remote Method Invocation***

Server Dienstanbieter (s. ***Client***)

Stellvertreter (proxies)
Außerhalb des COM-Kontextes als Stummel (***stubs***) bezeichnet. (s. Stummel)

STL ***Standard Template Library***

Stummel (stubs)
Üblicherweise werden außerhalb des COM-Kontextes Stellvertreter als Stummel (stubs) und Stummel als Skelette (skeletons) bezeichnet.

Tagged-Value
Wert (***Value***), der mit einem bestimmten Kennzeichen (***Tag***) zusammen gespeichert wird.

Anhang D – Glossar

UML ***Unified Modelling Language***

Anhang E – Quelltexte

E.1 Die Schnittstellen-Definition »GO.idl«

```
/* GO.idl

    IDL-Quellcode für GO.exe

    Diese Datei wird von dem Programm MIDL verarbeitet, um
    die Typbibliothek (GO.tlb) und Marshalling-Code zu erstellen.

    Der MIDL-Compiler erzeugt dabei auch 'GO_i.h' und 'GO_i.c'

    Autor:       Carsten Schu
    Version      1.0
    Erstellt am: 02.01.2001

    Copyright (c) 2000 / 2001 by Carsten Schu
*/

import "oaidl.idl";
import "ocidl.idl";

// ------------------------------------------------------------------------------

interface IGOApplication;

interface IGOCollection;

interface IGOElement;

interface IGOPackage;
interface IGOModel;

interface IGOClass;
interface IGOAttribute;
interface IGOOperation;
interface IGOParameter;

interface IGOAssociation;
interface IGORole;

// ------------------------------------------------------------------------------

[
    uuid(CF18DFE2-C6D3-11D4-81C7-00E09882CA77),
    version(1.0),
    helpstring("GO 1.0 Typbibliothek")
]
library GOLib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    // ------------------------------------------------------------------------
    // GOApplication

    [
        uuid(5EE61D40-C6E0-11D4-81C7-00E09882CA77),
        helpstring("GOApplication Class")
    ]
    coclass GOApplication
    {
        [default] interface IGOApplication;
    };

    [
        uuid(5EE61D41-C6E0-11D4-81C7-00E09882CA77),
        helpstring("IGOApplication Interface"),
        dual,
        object,
```

```
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOApplication : IDispatch
    {
        [helpstring("method GetCurrentModel"), id(1)] HRESULT GetCurrentModel([out, retval]
IGOModel **ppModel);
    };

    // ---------------------------------------------------------------------------
    // GOCollection

    [
        uuid(715C16E5-CB80-11D4-81C7-00E09882CA77),
        helpstring("GOCollection Class")
    ]
    coclass GOCollection
    {
        [default] interface IGOCollection;
    };

    [
        uuid(715C16E6-CB80-11D4-81C7-00E09882CA77),
        helpstring("IGOCollection Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOCollection : IDispatch
    {
        [propget, helpstring("property Count"), id(1)] HRESULT Count([out, retval] long *pVal);
        [helpstring("method GetAt"), id(2)] HRESULT GetAt([in] long Index, [out, retval]
IGOElement **ppElement);
    };

    // ---------------------------------------------------------------------------
    // GOElement

    [
        uuid(CA9476D1-CAAC-11D4-81C7-00E09882CA77),
        helpstring("GOElement Class")
    ]
    coclass GOElement
    {
        [default] interface IGOElement;
    };

    [
        uuid(CA9476D2-CAAC-11D4-81C7-00E09882CA77),
        helpstring("IGOElement Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOElement : IDispatch
    {
        [propget, helpstring("property Name"), id(1)] HRESULT Name([out, retval] BSTR *pVal);
        [propput, helpstring("property Name"), id(1)] HRESULT Name([in] BSTR newVal);
        [propget, helpstring("property Documentation"), id(2)] HRESULT Documentation([out,
retval] BSTR *pVal);
        [propput, helpstring("property Documentation"), id(2)] HRESULT Documentation([in] BSTR
newVal);
        [propget, helpstring("property Stereotype"), id(3)] HRESULT Stereotype([out, retval]
BSTR *pVal);
        [propput, helpstring("property Stereotype"), id(3)] HRESULT Stereotype([in] BSTR
newVal);
        [helpstring("method GetID"), id(4)] HRESULT GetID([out, retval] long *pID);
        [helpstring("method GetProperty"), id(5)] HRESULT GetProperty([in] BSTR PropertyName,
[out, retval] BSTR *pValue);
        [helpstring("method SetProperty"), id(6)] HRESULT SetProperty([in] BSTR PropertyName,
[in] BSTR Value, [out, retval] VARIANT_BOOL *ret);
    };
```

```
    // ------------------------------------------------------------------------
    // GOPackage

    [
        uuid(CA9476CB-CAAC-11D4-81C7-00E09882CA77),
        helpstring("GOPackage Class")
    ]
    coclass GOPackage
    {
        [default] interface IGOPackage;
        interface IGOElement;
    };

    [
        uuid(CA9476CC-CAAC-11D4-81C7-00E09882CA77),
        helpstring("IGOPackage Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOPackage : IDispatch
    {
        [helpstring("method AddPackage"), id(1)] HRESULT AddPackage([in] BSTR name, [out,
retval] IGOPackage **ppPackage);
        [helpstring("method GetAllClasses"), id(2)] HRESULT GetClasses([out, retval]
IGOCollection **ppCollection);
        [helpstring("method GetPackages"), id(3)] HRESULT GetPackages([out, retval]
IGOCollection **ppCollection);
    };

    // ------------------------------------------------------------------------
    // GOModel

    [
        uuid(CA9476C8-CAAC-11D4-81C7-00E09882CA77),
        helpstring("GOModel Class")
    ]
    coclass GOModel
    {
        [default] interface IGOModel;
        interface IGOPackage;
        interface IGOElement;
    };

    [
        uuid(CA9476C9-CAAC-11D4-81C7-00E09882CA77),
        helpstring("IGOModel Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOModel : IDispatch
    {
        [helpstring("method FindElementWithID"), id(1)] HRESULT FindElementWithID([in] long ID,
[out, retval] IGOElement **ppElement);
        [helpstring("method GetLogicalView"), id(2)] HRESULT GetLogicalView([out, retval]
IGOPackage **ppPackage);
    };

    // ------------------------------------------------------------------------
    // GOClass

    [
        uuid(5A1FC5A3-CB79-11D4-81C7-00E09882CA77),
        helpstring("GOClass Class")
    ]
    coclass GOClass
    {
        [default] interface IGOClass;
        interface IGOElement;
    };

    [
```

```
        uuid(5A1FC5A4-CB79-11D4-81C7-00E09882CA77),
        helpstring("IGOClass Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOClass : IDispatch
    {
        [propget, helpstring("property Abstract"), id(1)] HRESULT Abstract([out, retval]
VARIANT_BOOL *pVal);
        [propput, helpstring("property Abstract"), id(1)] HRESULT Abstract([in] VARIANT_BOOL
newVal);
        [helpstring("method GetAllAttributes"), id(2)] HRESULT GetAttributes([out, retval]
IGOCollection **ppCollection);
        [helpstring("method GetAllOperations"), id(3)] HRESULT GetOperations([out, retval]
IGOCollection **ppCollection);
        [helpstring("method AddAttribute"), id(4)] HRESULT AddAttribute([in] BSTR Name, [in]
BSTR Type, [in] BSTR InitValue, [out, retval] IGOAttribute **ppAttribute);
        [helpstring("method DeleteAttribute"), id(5)] HRESULT DeleteAttribute([in] IGOAttribute
*pAttribute, [out, retval] VARIANT_BOOL *ret);
        [helpstring("method GetAllAssociations"), id(6)] HRESULT GetAssociations([out, retval]
IGOCollection **ppCollection);
        [helpstring("method GetAllSuperclasses"), id(7)] HRESULT GetSuperclasses([out, retval]
IGOCollection **ppCollection);
    };

    // ------------------------------------------------------------------------
    // GOAttribute

    [
        uuid(5A1FC5A5-CB79-11D4-81C7-00E09882CA77),
        helpstring("GOAttribute Class")
    ]
    coclass GOAttribute
    {
        [default] interface IGOAttribute;
        interface IGOElement;
    };

    [
        uuid(5A1FC5A6-CB79-11D4-81C7-00E09882CA77),
        helpstring("IGOAttribute Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOAttribute : IDispatch
    {
        [propget, helpstring("property Type"), id(1)] HRESULT Type([out, retval] BSTR *pVal);
        [propput, helpstring("property Type"), id(1)] HRESULT Type([in] BSTR newVal);
        [propget, helpstring("property Static"), id(2)] HRESULT Static([out, retval]
VARIANT_BOOL *pVal);
        [propput, helpstring("property Static"), id(2)] HRESULT Static([in] VARIANT_BOOL
newVal);
        [propget, helpstring("property Derived"), id(3)] HRESULT Derived([out, retval]
VARIANT_BOOL *pVal);
        [propput, helpstring("property Derived"), id(3)] HRESULT Derived([in] VARIANT_BOOL
newVal);
        [propget, helpstring("property InitValue"), id(4)] HRESULT InitValue([out, retval] BSTR
*pVal);
        [propput, helpstring("property InitValue"), id(4)] HRESULT InitValue([in] BSTR newVal);
    };

    // ------------------------------------------------------------------------
    // GOOperation

    [
        uuid(5A1FC5AB-CB79-11D4-81C7-00E09882CA77),
        helpstring("GOOperation Class")
    ]
    coclass GOOperation
    {
        [default] interface IGOOperation;
```

```
        interface IGOElement;
    };

    [
        uuid(5A1FC5AC-CB79-11D4-81C7-00E09882CA77),
        helpstring("IGOOperation Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOOperation : IDispatch
    {
        [propget, helpstring("property ReturnType"), id(1)] HRESULT ReturnType([out, retval]
BSTR *pVal);
        [propput, helpstring("property ReturnType"), id(1)] HRESULT ReturnType([in] BSTR
newVal);
        [helpstring("method GetParameters"), id(2)] HRESULT GetParameters([out, retval]
IGOCollection **ppCollection);
        [helpstring("method RemoveAllParameters"), id(3)] HRESULT RemoveAllParameters();
        [helpstring("method AddParameter"), id(4)] HRESULT AddParameter([in] BSTR Name, [in]
BSTR Type, [in] BSTR InitValue, [in] long Position, [out, retval] IGOParameter **ppParameter);
    };

    // ------------------------------------------------------------------------
    // GOParameter

    [
        uuid(5A1FC5AD-CB79-11D4-81C7-00E09882CA77),
        helpstring("GOParameter Class")
    ]
    coclass GOParameter
    {
        [default] interface IGOParameter;
        interface IGOElement;
    };

    [
        uuid(5A1FC5AE-CB79-11D4-81C7-00E09882CA77),
        helpstring("IGOParameter Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOParameter : IDispatch
    {
        [propget, helpstring("property Type"), id(1)] HRESULT Type([out, retval] BSTR *pVal);
        [propput, helpstring("property Type"), id(1)] HRESULT Type([in] BSTR newVal);
    };

    // ------------------------------------------------------------------------
    // GOAssociation

    [
        uuid(5A1FC5A7-CB79-11D4-81C7-00E09882CA77),
        helpstring("GOAssociation Class")
    ]
    coclass GOAssociation
    {
        [default] interface IGOAssociation;
        interface IGOElement;
    };

    [
        uuid(5A1FC5A8-CB79-11D4-81C7-00E09882CA77),
        helpstring("IGOAssociation Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGOAssociation : IDispatch
    {
```

```
        [helpstring("method GetRole1"), id(1)] HRESULT GetRole1([out, retval] IGORole
**ppRole1);
        [helpstring("method GetRole2"), id(2)] HRESULT GetRole2([out, retval] IGORole
**ppRole2);
    };

    // ------------------------------------------------------------------------
    // GORole

    [
        uuid(5A1FC5A9-CB79-11D4-81C7-00E09882CA77),
        helpstring("GORole Class")
    ]
    coclass GORole
    {
        [default] interface IGORole;
        interface IGOElement;
    };

    [
        uuid(5A1FC5AA-CB79-11D4-81C7-00E09882CA77),
        helpstring("IGORole Interface"),
        dual,
        object,
        oleautomation,
        pointer_default(unique)
    ]
    interface IGORole : IDispatch
    {
        [propget, helpstring("property Cardinality"), id(1)] HRESULT Cardinality([out, retval]
BSTR *pVal);
        [propput, helpstring("property Cardinality"), id(1)] HRESULT Cardinality([in] BSTR
newVal);
        [propget, helpstring("property Navigable"), id(2)] HRESULT Navigable([out, retval]
VARIANT_BOOL *pVal);
        [propput, helpstring("property Navigable"), id(2)] HRESULT Navigable([in] VARIANT_BOOL
newVal);
        [helpstring("method GetClass"), id(3)] HRESULT GetClass([out, retval] IGOClass
**ppClass);
        [helpstring("method GetOtherRole"), id(4)] HRESULT GetOtherRole([out, retval] IGORole
**ppRole);
    };

    // ------------------------------------------------------------------------
};
```

E.2 ATL-Unterstützung in der Anwendung

E.2.1 »stdafx.h«

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#define VC_EXTRALEAN        // Selten verwendete Teile der Windows-Header nicht einbinden

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions

#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows 95 Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxtempl.h>
#include <fstream.h>

#include "id.h"
#include "TreeArchive.h"

// Includes zum Aufbau des DOM-Baums für den XMI-Export
#include <util/PlatformUtils.hpp>
#include <util/XMLString.hpp>
#include <dom/DOM.hpp>
#include <parsers/DOMParser.hpp>
#include <dom/DOM_Node.hpp>
#include <dom/DOM_NamedNodeMap.hpp>

//////////////////////////////////////////////////////////////////////
// cs001130 : Der folgende Code dient der ATL-Unterstützung für COM

#ifndef C_GO_MODULE
#define C_GO_MODULE

#define _ATL_APARTMENT_THREADED

#define _ATL_DEBUG_INTERFACES // cs: Debug-Informationen ausgeben

#include <atlbase.h>
// Sie können eine Klasse von CComModule ableiten und dies verwenden, wenn Sie
// etwas überladen wollen. Ändern Sie aber nicht den Namen von _Module
class CGOModule : public CComModule
{
public:
     LONG Unlock();
     LONG Lock();
     LPCTSTR FindOneOf(LPCTSTR p1, LPCTSTR p2);
     DWORD dwThreadID;
};
extern CGOModule _Module;
#include <atlcom.h>

#endif //C_GO_MODULE
```

E.2.2 »CApplication.cpp«

```
// Application.cpp : Defines the class behaviors for the application.

#include "stdafx.h"

... // weiterer Code, der für diese Studienarbeit nicht von Bedeutung ist

// csOO1212: Includes für die COM-Schnittstelle
// ------------------------------------------
#include "CGOApplication.h"
#include "CGOAssociation.h"
#include "CGOAttribute.h"
#include "CGOClass.h"
#include "CGOCollection.h"

#include "CGOElement.h"
#include "CGOPackage.h"

#include "CGOModel.h" // includes CGOPackage.h & CGOElement.h
#include "CGOOperation.h"
#include "CGOParameter.h"
#include "CGORole.h"

#include "GO_i.c"
// ------------------------------------------

... // weiterer Code, der für diese Studienarbeit nicht von Bedeutung ist

/////////////////////////////////////////////////////////////////////////////
// The one and only CApplication object
CApplication theApp;

/////////////////////////////////////////////////////////////////////////////
// CObjectCApp initialization

BOOL CApplication::InitInstance()
{
    if (!InitATL())
        return FALSE;

    CCommandLineInfo cmdInfo;

    ParseCommandLine(cmdInfo);

    if (cmdInfo.m_bRunEmbedded || cmdInfo.m_bRunAutomated)
    {
        return TRUE;
    }

    CSplashWnd::EnableSplashScreen(cmdInfo.m_bShowSplash);

    GetPrinterDim();

#ifdef _AFXDLL
    Enable3dControls();        // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

    SetRegistryKey(_T("SWT"));
    LoadStdProfileSettings();  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    m_pDocTemplate = new CMultiDocTemplate(
        IDR_OBJECTTYPE,
        RUNTIME_CLASS(CDocProject),
        RUNTIME_CLASS(CChildFrame), // custom MDI child frame
        RUNTIME_CLASS(CViewBase));

    // Dokumentvorlage in Liste aufnehmen
```

```
    AddDocTemplate(m_pDocTemplate);

    // create main MDI Frame window
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;

    // Neuer Standard: Applikation ohne neuen File öffnen
    if(cmdInfo.m_nShellCommand == CCommandLineInfo::FileNew)
        cmdInfo.m_nShellCommand = CCommandLineInfo::FileNothing;
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The main window has been initialized, so show and update it.
    pMainFrame->ShowWindow(SW_SHOWMAXIMIZED/*m_nCmdShow*/);
    pMainFrame->UpdateWindow();

    // CG: The following block was inserted by 'Document Registration' component.'
    {
        // Enable DDE Execute open
        EnableShellOpen();
        RegisterShellFileTypes(TRUE);

        // TODO: You may remove the #ifndef block below if you already have an
        //  #ifdef _MAC block around a DragAcceptFiles() above.
        //  Just remove the #ifdef _MAC around it and the #ifndef that follows.
#ifndef _MAC
        m_pMainWnd->DragAcceptFiles();
#endif
    }
    return TRUE;
}

... // weiterer Code, der für diese Studienarbeit nicht von Bedeutung ist

/////////////////////////////////////////////////////////////////////////////

// csOO1130 : Der nachfolgende Code dient ausschließlich der ATL-Unterstützung für COM

CGOModule _Module;

//csOO1212: ObjectMap von Hand gefüllt!

BEGIN_OBJECT_MAP(ObjectMap)
OBJECT_ENTRY(CLSID_GOApplication, CGOApplication)
OBJECT_ENTRY(CLSID_GOCollection, CGOCollection)
OBJECT_ENTRY(CLSID_GOElement, CGOElement)
OBJECT_ENTRY(CLSID_GOPackage, CGOPackage)
OBJECT_ENTRY(CLSID_GOModel, CGOModel)
OBJECT_ENTRY(CLSID_GOClass, CGOClass)
OBJECT_ENTRY(CLSID_GOAttribute, CGOAttribute)
OBJECT_ENTRY(CLSID_GOOperation, CGOOperation)
OBJECT_ENTRY(CLSID_GOParameter, CGOParameter)
OBJECT_ENTRY(CLSID_GOAssociation, CGOAssociation)
OBJECT_ENTRY(CLSID_GORole, CGORole)
END_OBJECT_MAP()

LONG CGOModule::Unlock()
{
    AfxOleUnlockApp();
    return 0;
}

LONG CGOModule::Lock()
{
    AfxOleLockApp();
    return 1;
}

LPCTSTR CGOModule::FindOneOf(LPCTSTR p1, LPCTSTR p2)
{
    while (*p1 != NULL)
    {
        LPCTSTR p = p2;
```

```
        while (*p != NULL)
        {
            if (*p1 == *p)
                return CharNext(p1);
            p = CharNext(p);
        }
        p1++;
    }
    return NULL;
}

int CApplication::ExitInstance()
{
    if (m_bATLInited)
    {
        _Module.RevokeClassObjects();
        _Module.Term();
        CoUninitialize();
    }

    return CWinApp::ExitInstance();
}

BOOL CApplication::InitATL()
{
    m_bATLInited = TRUE;

#if _WIN32_WINNT >= 0x0400
    HRESULT hRes = CoInitializeEx(NULL, COINIT_MULTITHREADED);
#else
    HRESULT hRes = CoInitialize(NULL);
#endif

    if (FAILED(hRes))
    {
        m_bATLInited = FALSE;
        return FALSE;
    }

    _Module.Init(ObjectMap, AfxGetInstanceHandle());
    _Module.dwThreadID = GetCurrentThreadId();

    LPTSTR lpCmdLine = GetCommandLine(); //this line necessary for _ATL_MIN_CRT
    TCHAR szTokens[] = _T("-/");

    BOOL bRun = TRUE;
    LPCTSTR lpszToken = _Module.FindOneOf(lpCmdLine, szTokens);
    while (lpszToken != NULL)
    {
        if (lstrcmpi(lpszToken, _T("UnregServer"))==0)
        {
            _Module.UpdateRegistryFromResource(IDR_GO, FALSE);
            _Module.UnregisterServer(TRUE); //TRUE zeigt an, daß die Typbibliothek nicht
registriert ist
            bRun = FALSE;
            break;
        }
        if (lstrcmpi(lpszToken, _T("RegServer"))==0)
        {
            _Module.UpdateRegistryFromResource(IDR_GO, TRUE);
            _Module.RegisterServer(TRUE);
            bRun = FALSE;
            break;
        }
        lpszToken = _Module.FindOneOf(lpszToken, szTokens);
    }

    if (!bRun)
    {
        m_bATLInited = FALSE;
        _Module.Term();
        CoUninitialize();
        return FALSE;
    }
```

```
    hRes = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER,
        REGCLS_MULTIPLEUSE);
    if (FAILED(hRes))
    {
        m_bATLInited = FALSE;
        CoUninitialize();
        return FALSE;
    }

    return TRUE;
}
```

E.3 Registry-Dateien

E.3.1 »GO.rgs«

```
HKCR
{
    NoRemove AppID
    {
        {CF18DFE3-C6D3-11D4-81C7-00E09882CA77} = s 'GO'
        'GO.EXE'
        {
            val AppID = s {CF18DFE3-C6D3-11D4-81C7-00E09882CA77}
        }
    }
}
```

E.3.2 »GOApplication.rgs«

```
HKCR
{
    GO.GOApplication.1 = s 'GOApplication Class'
    {
        CLSID = s '{5EE61D40-C6E0-11D4-81C7-00E09882CA77}'
    }
    GO.GOApplication = s 'GOApplication Class'
    {
        CLSID = s '{5EE61D40-C6E0-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOApplication.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5EE61D40-C6E0-11D4-81C7-00E09882CA77} = s 'GOApplication Class'
        {
            ProgID = s 'GO.GOApplication.1'
            VersionIndependentProgID = s 'GO.GOApplication'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.3 »GOCollection.rgs«

```
HKCR
{
    GO.GOCollection.1 = s 'GOCollection Class'
    {
        CLSID = s '{715C16E5-CB80-11D4-81C7-00E09882CA77}'
    }
    GO.GOCollection = s 'GOCollection Class'
    {
        CLSID = s '{715C16E5-CB80-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOCollection.1'
    }
    NoRemove CLSID
    {
        ForceRemove {715C16E5-CB80-11D4-81C7-00E09882CA77} = s 'GOCollection Class'
        {
            ProgID = s 'GO.GOCollection.1'
            VersionIndependentProgID = s 'GO.GOCollection'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.4 »GOElement.rgs«

```
HKCR
{
    GO.GOElement.1 = s 'GOElement Class'
    {
        CLSID = s '{CA9476D1-CAAC-11D4-81C7-00E09882CA77}'
    }
    GO.GOElement = s 'GOElement Class'
    {
        CLSID = s '{CA9476D1-CAAC-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOElement.1'
    }
    NoRemove CLSID
    {
        ForceRemove {CA9476D1-CAAC-11D4-81C7-00E09882CA77} = s 'GOElement Class'
        {
            ProgID = s 'GO.GOElement.1'
            VersionIndependentProgID = s 'GO.GOElement'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.5 »GOPackage.rgs«

```
HKCR
{
    GO.GOPackage.1 = s 'GOPackage Class'
    {
        CLSID = s '{CA9476CB-CAAC-11D4-81C7-00E09882CA77}'
    }
    GO.GOPackage = s 'GOPackage Class'
    {
        CLSID = s '{CA9476CB-CAAC-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOPackage.1'
    }
    NoRemove CLSID
    {
        ForceRemove {CA9476CB-CAAC-11D4-81C7-00E09882CA77} = s 'GOPackage Class'
        {
            ProgID = s 'GO.GOPackage.1'
            VersionIndependentProgID = s 'GO.GOPackage'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.6 »GOModel.rgs«

```
HKCR
{
    GO.GOModel.1 = s 'GOModel Class'
    {
        CLSID = s '{C4B914D7-CAA7-11D4-81C7-00E09882CA77}'
    }
    GO.GOModel = s 'GOModel Class'
    {
        CLSID = s '{C4B914D7-CAA7-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOModel.1'
    }
    NoRemove CLSID
    {
        ForceRemove {C4B914D7-CAA7-11D4-81C7-00E09882CA77} = s 'GOModel Class'
        {
            ProgID = s 'GO.GOModel.1'
            VersionIndependentProgID = s 'GO.GOModel'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.7 »GOClass.rgs«

```
HKCR
{
    GO.GOClass.1 = s 'GOClass Class'
    {
        CLSID = s '{5A1FC5A3-CB79-11D4-81C7-00E09882CA77}'
    }
    GO.GOClass = s 'GOClass Class'
    {
        CLSID = s '{5A1FC5A3-CB79-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOClass.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5A1FC5A3-CB79-11D4-81C7-00E09882CA77} = s 'GOClass Class'
        {
            ProgID = s 'GO.GOClass.1'
            VersionIndependentProgID = s 'GO.GOClass'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.8 »GOAttribute.rgs«

```
HKCR
{
    GO.GOAttribute.1 = s 'GOAttribute Class'
    {
        CLSID = s '{5A1FC5A5-CB79-11D4-81C7-00E09882CA77}'
    }
    GO.GOAttribute = s 'GOAttribute Class'
    {
        CLSID = s '{5A1FC5A5-CB79-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOAttribute.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5A1FC5A5-CB79-11D4-81C7-00E09882CA77} = s 'GOAttribute Class'
        {
            ProgID = s 'GO.GOAttribute.1'
            VersionIndependentProgID = s 'GO.GOAttribute'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.9 »GOOperation.rgs«

```
HKCR
{
    GO.GOOperation.1 = s 'GOOperation Class'
    {
        CLSID = s '{5A1FC5AB-CB79-11D4-81C7-00E09882CA77}'
    }
    GO.GOOperation = s 'GOOperation Class'
    {
        CLSID = s '{5A1FC5AB-CB79-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOOperation.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5A1FC5AB-CB79-11D4-81C7-00E09882CA77} = s 'GOOperation Class'
        {
            ProgID = s 'GO.GOOperation.1'
            VersionIndependentProgID = s 'GO.GOOperation'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.10 »GOParameter.rgs«

```
HKCR
{
    GO.GOParameter.1 = s 'GOParameter Class'
    {
        CLSID = s '{5A1FC5AD-CB79-11D4-81C7-00E09882CA77}'
    }
    GO.GOParameter = s 'GOParameter Class'
    {
        CLSID = s '{5A1FC5AD-CB79-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOParameter.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5A1FC5AD-CB79-11D4-81C7-00E09882CA77} = s 'GOParameter Class'
        {
            ProgID = s 'GO.GOParameter.1'
            VersionIndependentProgID = s 'GO.GOParameter'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.11 »GOAssociation.rgs«

```
HKCR
{
    GO.GOAssociation.1 = s 'GOAssociation Class'
    {
        CLSID = s '{5A1FC5A7-CB79-11D4-81C7-00E09882CA77}'
    }
    GO.GOAssociation = s 'GOAssociation Class'
    {
        CLSID = s '{5A1FC5A7-CB79-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GOAssociation.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5A1FC5A7-CB79-11D4-81C7-00E09882CA77} = s 'GOAssociation Class'
        {
            ProgID = s 'GO.GOAssociation.1'
            VersionIndependentProgID = s 'GO.GOAssociation'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.3.12 »GORole.rgs«

```
HKCR
{
    GO.GORole.1 = s 'GORole Class'
    {
        CLSID = s '{5A1FC5A9-CB79-11D4-81C7-00E09882CA77}'
    }
    GO.GORole = s 'GORole Class'
    {
        CLSID = s '{5A1FC5A9-CB79-11D4-81C7-00E09882CA77}'
        CurVer = s 'GO.GORole.1'
    }
    NoRemove CLSID
    {
        ForceRemove {5A1FC5A9-CB79-11D4-81C7-00E09882CA77} = s 'GORole Class'
        {
            ProgID = s 'GO.GORole.1'
            VersionIndependentProgID = s 'GO.GORole'
            ForceRemove 'Programmable'
            LocalServer32 = s '%MODULE%'
            val AppID = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
            'TypeLib' = s '{CF18DFE2-C6D3-11D4-81C7-00E09882CA77}'
        }
    }
}
```

E.4 Schnittstellenimplementierungen

E.4.1 »CGOApplication«

E.4.1.1 »CGOApplication.h«

```
/* CGOApplication.h

    Die Klasse CGOApplication implementiert die COM-Schnittstelle IGOApplication.
    Sie repräsentiert das Anwendungsobjekt und bietet die Möglichkeit, auf das
    aktuelle GO-Model zuzugreifen.

    Autor:       Carsten Schu
    Version      1.0
    Erstellt am: 31.12.2000

    Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOAPPLICATION_H_
#define _INC_CGOAPPLICATION_H_

#include "resource.h"
#include "GO_i.h"

#include "ActionManager.h"
#include "ActionGroup.h"

class CGOModel;

class ATL_NO_VTABLE CGOApplication
: public CComObjectRootEx<CComSingleThreadModel>
, public CComCoClass<CGOApplication,&CLSID_GOApplication>
, public IDispatchImpl<IGOApplication,&IID_IGOApplication,&LIBID_GOLib>
{
private:
    CComObject<CGOModel> *pGOModel;    // Pointer auf das Model

    ActionManager* pActionManager;
    ActionGroup* pActionGroup;

public:
    CGOApplication();
    ~CGOApplication();

    CreateCOMObjects();

public:
    STDMETHOD (GetCurrentModel)(IGOModel** ppModel);


DECLARE_REGISTRY_RESOURCEID(IDR_GOAPPLICATION)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOApplication)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(IGOApplication)
END_COM_MAP()

};

#endif // _INC_CGOAPPLICATION_H_
```

E.4.1.2 »CGOApplication.cpp«

```
// CGOApplication.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOApplication.h"
#include "CGOModel.h"

#include "Application.h"
#include "docproject.h"
extern CApplication theApp;


CGOApplication::CGOApplication()
{
     if (theApp.GetDocument() == NULL)
     {
          //AfxMessageBox("GO muss gestartet sein und ein Modell muss existieren!");
          return;
     }

     pGOModel = NULL;

     pActionManager = theApp.GetDocument()->GetActionManager();
     pActionGroup = new ActionGroup("Änderungen über COM");

     CreateCOMObjects();
}

CGOApplication::~CGOApplication()
{
     pActionManager->PerformAction(pActionGroup);// Action-Group zur Sicherheit nochmal ausführen

     if (pGOModel)
          pGOModel->Release();
}

CGOApplication::CreateCOMObjects()
{
     HRESULT hr = CComObject<CGOModel>::CreateInstance(&pGOModel);

     if (SUCCEEDED(hr))
     {
          pGOModel->AddRef();
          // Das MModelElement initialisieren
          pGOModel->SetMModelElement((MModelElement*)theApp.GetDocument()->GetMModel());
          pGOModel->SetActionGroup(pActionGroup);
          pGOModel->CreateCOMObjects();
     }
     else
     {
          AfxMessageBox("Konnte in 'CGOApplication' kein GOModel anlegen.");
          pGOModel = NULL;
     }
}


STDMETHODIMP CGOApplication::GetCurrentModel(IGOModel** ppModel)
{
     CComQIPtr<IGOModel> d_pGOModel (pGOModel);  // Calls QI

     if (d_pGOModel)
     {
          return d_pGOModel.CopyTo(ppModel); // Calls AddRef !
     }    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück

     return E_FAIL;
}
```

E.4.2 »CGOCollection«

E.4.2.1 »CGOCollection.h«

```
/* CGOCollection.h

    Die Klasse CGOCollection implementiert die COM-Schnittstelle IGOCollection.
    Sie repräsentiert eine Liste von Objekten für die COM-Schnittstelle.
    Sie beinhaltet eine Liste aller CComObjects, die sie über die
    COM-Schnittstelle nach außen zur Verfügung stellt.

    Autor:       Carsten Schu
    Version      1.0
    Erstellt am: 02.01.2001

    Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOCOLLECTION_3A2E6B4E02BC_INCLUDED
#define _INC_CGOCOLLECTION_3A2E6B4E02BC_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class ATL_NO_VTABLE CGOCollection
: public CComObjectRootEx<CComSingleThreadModel>
, public CComCoClass<CGOCollection,&CLSID_GOCollection>
, public IDispatchImpl<IGOCollection,&IID_IGOCollection,&LIBID_GOLib>
{
private:
    // eine Liste von CComObject-Pointern des Typs CGOElement
    CTypedPtrList<CPtrList, CComObject<CGOElement>*> *pElementCollection;

public:
    CGOCollection();
    ~CGOCollection();

    AddElement(CComObject<CGOElement>* pElement);

    int GetCount();
    CComObject<CGOElement>*   GetElementAt(int index);
    void RemoveElementAt(int index);
    void RemoveAllElements();

public:
    STDMETHOD (get_Count)(long* pVal);
    STDMETHOD (GetAt)(long Index, IGOElement** ppElement);


DECLARE_REGISTRY_RESOURCEID(IDR_GOCOLLECTION)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOCollection)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(IGOCollection)
END_COM_MAP()

};

#endif /* _INC_CGOCOLLECTION_3A2E6B4E02BC_INCLUDED */
```

E.4.2.2 »CGOCollection.cpp«

```
// CGOCollection.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOCollection.h"

CGOCollection::CGOCollection()
{
    pElementCollection = new CTypedPtrList<CPtrList, CComObject<CGOElement>*> ();
}

CGOCollection::~CGOCollection()
{
    if (pElementCollection != NULL)
    {
        CComObject<CGOElement>* obj = NULL;

        while (!pElementCollection->IsEmpty())
        {
            obj = pElementCollection->RemoveHead();
            obj->Release();
        }
        delete pElementCollection;
    }
}


CGOCollection::AddElement(CComObject<CGOElement>* pElement)
{
    pElement->AddRef();
    pElementCollection->AddTail(pElement);
}

int CGOCollection::GetCount()
{
    return pElementCollection->GetCount();
}

CComObject<CGOElement>*    CGOCollection::GetElementAt(int index)
{
    POSITION pos = pElementCollection->FindIndex(index);
    return pElementCollection->GetAt(pos);
}

void CGOCollection::RemoveElementAt(int index)
{
    POSITION pos = pElementCollection->FindIndex(index);

    CComObject<CGOElement>*    pElement = pElementCollection->GetAt(pos);

    pElementCollection->RemoveAt(pos);
    pElement->Release();
}

void CGOCollection::RemoveAllElements()
{
    if (pElementCollection != NULL)
    {
        CComObject<CGOElement>* obj = NULL;

        while (!pElementCollection->IsEmpty())
        {
            obj = pElementCollection->RemoveHead();
            obj->Release();
        }
    }
}


STDMETHODIMP CGOCollection::get_Count(long* pVal)
{
```

```
    if (pVal)
    {
        *pVal = pElementCollection->GetCount();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGOCollection::GetAt(long Index, IGOElement** ppElement)
{
    POSITION pos = NULL;

    if(pElementCollection->GetCount() > Index)
    {
        pos = pElementCollection->FindIndex(Index);

        CComQIPtr<IGOElement> d_pElement(pElementCollection->GetAt(pos)); // Calls QI

        if (d_pElement)
        {
            return d_pElement.CopyTo(ppElement);    // Calls AddRef !
        }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    }
    return E_FAIL;
}
```

E.4.3 »CGOElement«

E.4.3.1 »CGOElement.h«

```
/* CGOElement.h

     Die Klasse CGOElement implementiert die COM-Schnittstelle IGOElement.
     CGOElement ist die Oberklasse zu CGOAssociation, CGOAttribute, CGOClass,
     CGOPackage, (CGOModel über CGOPackage), CGOOperation, CGOParameter und
     CGORole.
     Sie stellt allgemeine Zugriffs-Funktionen auf alle Unterklassen-Objekte
     zur Verfügung, so wie Zugriffe auf Name und Dokumentation eines ModelElements.
     Fachkonzept-Objekt ist MModelElement.

     Autor:       Carsten Schu
     Version      1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOELEMENT_3A2CE75700B4_INCLUDED
#define _INC_CGOELEMENT_3A2CE75700B4_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "ActionGroup.h"

#include "MModelElement.h"

class ATL_NO_VTABLE CGOElement
: public CComObjectRootEx<CComSingleThreadModel>
, public CComCoClass<CGOElement,&CLSID_GOElement>
, public IDispatchImpl<IGOElement,&IID_IGOElement,&LIBID_GOLib>
{
protected:
     ActionGroup* pActionGroup;

public:
     MModelElement* theMModelElement;

     CGOElement();
     ~CGOElement();

     SetActionGroup(ActionGroup* pGroup);
     SetMModelElement(MModelElement* pVal);

     virtual CreateCOMObjects();
     virtual CComObject<CGOElement>* FindElementWithID(long ID);

public:
     STDMETHOD (get_Name)(BSTR* pVal);
     STDMETHOD (put_Name)(BSTR newVal);

     STDMETHOD (get_Documentation)(BSTR* pVal);
     STDMETHOD (put_Documentation)(BSTR newVal);

     STDMETHOD (get_Stereotype)(BSTR* pVal);
     STDMETHOD (put_Stereotype)(BSTR newVal);

     STDMETHOD (GetID)(long* pID);

     STDMETHOD (GetProperty)(BSTR PropertyName, BSTR* pValue);
     STDMETHOD (SetProperty)(BSTR PropertyName, BSTR Value, VARIANT_BOOL* ret);


DECLARE_REGISTRY_RESOURCEID(IDR_GOELEMENT)
```

```
DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOElement)
     COM_INTERFACE_ENTRY(IDispatch)
     COM_INTERFACE_ENTRY(IGOElement)
END_COM_MAP()

};

#endif /* _INC_CGOELEMENT_3A2CE75700B4_INCLUDED */
```

E.4.3.2 »CGOElement.cpp«

```
// CGOElement.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOElement.h"

#include "ActionEditMModelElement.h"


CGOElement::CGOElement()
{
}

CGOElement::~CGOElement()
{
}


CGOElement::SetActionGroup(ActionGroup* pGroup)
{
    pActionGroup = pGroup;
}

CGOElement::SetMModelElement(MModelElement* pVal)
{
    theMModelElement = pVal;
}


/*virtual*/
CGOElement::CreateCOMObjects()
{   // standardmäßig leere Implementation!
}   // Unterklassen können diese Funktion überschreiben

/*virtual*/
CComObject<CGOElement>* CGOElement::FindElementWithID(long ID)
{
    // Unterklassen können diese Funktion überschreiben
    return NULL; // nichts gefunden
}


STDMETHODIMP CGOElement::GetID(long* pID)
{
    if (pID)
    {
        *pID = theMModelElement->getID();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}


STDMETHODIMP CGOElement::get_Name(BSTR* pVal)
{
    if (pVal)
    {
        CString text = theMModelElement->getname();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGOElement::put_Name(BSTR newVal)
```

```
{
    CString text = CString(newVal);

    ActionEditMModelElement* pEdit = new ActionEditMModelElement(theMModelElement);
    pActionGroup->AddSubAction(pEdit);

    pEdit->SetChangedName(text);
    pEdit->Do();

    return S_OK;
}


STDMETHODIMP CGOElement::get_Documentation(BSTR* pVal)
{
    if (pVal)
    {
        CString text = theMModelElement->getdescription();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGOElement::put_Documentation(BSTR newVal)
{
    CString text = CString(newVal);

    ActionEditMModelElement* pEdit = new ActionEditMModelElement(theMModelElement);
    pActionGroup->AddSubAction(pEdit);

    pEdit->SetChangedDocumentation(text);
    pEdit->Do();

    return S_OK;
}


// Stereotypes werden von GO momentan noch nicht unterstützt
STDMETHODIMP CGOElement::get_Stereotype(BSTR* pVal)
{
    return E_NOTIMPL; // MStereotype noch nicht fertig!
}

STDMETHODIMP CGOElement::put_Stereotype(BSTR newVal)
{
    return E_NOTIMPL; // MStereotype noch nicht fertig!
}

// Daten in/aus Tagged-Values wird von GO momentan noch nicht unterstützt
STDMETHODIMP CGOElement::GetProperty(BSTR PropertyName, BSTR* pValue)
{
    return E_NOTIMPL;
}

STDMETHODIMP CGOElement::SetProperty(BSTR PropertyName, BSTR Value, VARIANT_BOOL* ret)
{
    if (ret)
        *ret = VARIANT_FALSE;

    return E_NOTIMPL;
}
```

E.4.4 »CGOPackage«

E.4.4.1 »CGOPackage.h«

```
/* CGOPackage.h

     Die Klasse CGOPackage implementiert die COM-Schnittstelle IGOPackage.

     CGOPackage enthält Listen aller enthaltenen Unter-Packages und Klassen.
     CGOPackage ist Oberklasse für CGOModel (Fachkonzept-Objekt MPackage).

     Autor:       Carsten Schu
     Version      1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOPACKAGE_3A2CE63001FE_INCLUDED
#define _INC_CGOPACKAGE_3A2CE63001FE_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class CGOCollection;


class ATL_NO_VTABLE CGOPackage
: public CGOElement
, public IDispatchImpl<IGOPackage,&IID_IGOPackage,&LIBID_GOLib>
{
protected:
     CComObject<CGOCollection> *pPackageCollection;
     CComObject<CGOCollection> *pClassCollection;

public:
     CGOPackage();
     ~CGOPackage();

     CreateCOMObjects(); // Überschreibt Funktion aus CGOElement

     CComObject<CGOElement>* FindElementWithID(long ID);

public:
     STDMETHOD (AddPackage)(BSTR name, IGOPackage** ppPackage);
     STDMETHOD (GetClasses)(IGOCollection** ppCollection);
     STDMETHOD (GetPackages)(IGOCollection** ppCollection);


DECLARE_REGISTRY_RESOURCEID(IDR_GOPACKAGE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOPackage)
     COM_INTERFACE_ENTRY(IGOPackage)
     COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
     COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};

#endif /* _INC_CGOPACKAGE_3A2CE63001FE_INCLUDED */
```

E.4.4.2 »CGOPackage.cpp«

```
// CGOPackage.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOPackage.h"

#include "CGOClass.h"
#include "CGOCollection.h"

#include "MPackage.h"

#include "ActionCreateMPackage.h"
#include "ActionEditMPackage.h"


CGOPackage::CGOPackage()
{
     pPackageCollection = NULL;
     pClassCollection = NULL;
}

CGOPackage::~CGOPackage()
{
     if (pPackageCollection)
          pPackageCollection->Release();

     if (pClassCollection)
          pClassCollection->Release();
}


CGOPackage::CreateCOMObjects()
{
     HRESULT hr;

     HRESULT hr1 = CComObject<CGOCollection>::CreateInstance(&pPackageCollection);
     HRESULT hr2 = CComObject<CGOCollection>::CreateInstance(&pClassCollection);

     if (SUCCEEDED(hr1) && SUCCEEDED(hr2))
     {    // Objekte der Collections erzeugen !!!

          pPackageCollection->AddRef();
          pClassCollection->AddRef();

          MModelElement* pME;

          POSITION pos = ((MPackage*)theMModelElement)->ownedElement.GetHeadPosition();

          // Alle Elemente des Packages durchsuchen
          while (pos != NULL)
          {
               pME = ((MPackage*)theMModelElement)->ownedElement.GetNext(pos);

               // ------------------------------------------------------------------
               // PackageCollection
               if (pME->GetSubjectType() == Subject::Package)
               {
                    CComObject<CGOPackage> *obj = NULL;
                    hr = CComObject<CGOPackage>::CreateInstance(&obj);
                    if (SUCCEEDED(hr))
                    {
                         obj->SetMModelElement(pME);
                         obj->SetActionGroup(pActionGroup);
                         obj->CreateCOMObjects();

                         pPackageCollection->AddElement((CComObject<CGOElement>*) obj);
                         // AddElement Calls AddRef für obj
                    }
               }

               // ------------------------------------------------------------------
```

```
            // ClassCollection
            if (pME->GetSubjectType() == Subject::Class)
            {
                CComObject<CGOClass> *obj = NULL;
                hr = CComObject<CGOClass>::CreateInstance(&obj);
                if (SUCCEEDED(hr))
                {
                    obj->SetMModelElement(pME);
                    obj->SetActionGroup(pActionGroup);
                    obj->CreateCOMObjects();

                    pClassCollection->AddElement((CComObject<CGOElement>*) obj);
                    // AddElement Calls AddRef für obj
                }
            }
            // ------------------------------------------------------------------
        }
    }
    else
    {
        AfxMessageBox("Konnte in 'CGOPackage' keine Collections anlegen.");
        pClassCollection = NULL;
        pPackageCollection = NULL;
    }
}


CComObject<CGOElement>* CGOPackage::FindElementWithID(long ID)
{
    long ElementID = 0;
    this->GetID(&ElementID);

    if (ElementID == ID)
        return ((CComObject<CGOElement>*)this);

    // Alle Klassen durchsuchen
    if (pClassCollection)
    {
        long Anzahl = pClassCollection->GetCount();

        for (long index = 0; index< Anzahl; index++)
        {
            CComObject<CGOElement> *theElement = NULL;

            theElement = pClassCollection->GetElementAt(index);

            if (theElement != NULL)
            {
                // Klassen untersuchen ihre Unterelemente selbst
                CComObject<CGOElement> *searchElement = NULL;

                searchElement = ((CComObject<CGOClass>*)theElement)->FindElementWithID(ID);

                if (searchElement != NULL)
                    return searchElement;
            }
        }
    }

    // Alle Unterpackages rekursiv durchsuchen
    if (pPackageCollection)
    {
        long Anzahl = pPackageCollection->GetCount();

        for (long i = 0; i < Anzahl; i++)
        {
            CComObject<CGOElement> *packageElement = pPackageCollection->GetElementAt(i);

            if (packageElement != NULL)
            {
                CComObject<CGOElement> *searchElement = NULL;

                searchElement = ((CComObject<CGOPackage>*)packageElement)-
>FindElementWithID(ID);
```

```
                if (searchElement != NULL)
                    return searchElement;
            }
        }
    }
    return NULL; // nichts gefunden!
}


STDMETHODIMP CGOPackage::GetPackages(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pPackageCollection (pPackageCollection); // Calls QI

    if (d_pPackageCollection)
    {
        return d_pPackageCollection.CopyTo(ppCollection); // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    return E_FAIL;
}

STDMETHODIMP CGOPackage::GetClasses(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pClassCollection (pClassCollection);      // Calls QI

    if (d_pClassCollection)
    {
        return d_pClassCollection.CopyTo(ppCollection); // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    return E_FAIL;
}

STDMETHODIMP CGOPackage::AddPackage(BSTR name, IGOPackage** ppPackage)
{
    // Neues Package anlegen
    ActionCreateMPackage* pCreate = new ActionCreateMPackage((MPackage*)theMModelElement);
    pActionGroup->AddSubAction(pCreate);
    pCreate->Do();

    MPackage* pPackage = pCreate->GetMPackage();

    // Eindeutigkeit des Namens prüfen
    if ((MPackage*)theMModelElement->DoesNameExist(name, Subject::Dummy, pPackage))
    {
        AfxMessageBox("Der Name \"" + CString(name) + "\" wird bereits für ein Member
verwendet.\nBitte wählen Sie einen anderen.");

        pCreate->UnDo();
        pActionGroup->RemoveLastAction();
        delete pCreate;

        return E_FAIL;
    }

    // Package bearbeiten
    ActionEditMPackage* pEdit = new ActionEditMPackage(pPackage);
    pActionGroup->AddSubAction(pEdit);

    pEdit->SetChangedName(name);
    pEdit->Do();

    // ComObject zur Liste hinzufügen
    CComObject<CGOPackage> *obj = NULL;
    HRESULT hr = CComObject<CGOPackage>::CreateInstance(&obj);
    if (SUCCEEDED(hr))
    {
        obj->SetMModelElement(pPackage);
        obj->SetActionGroup(pActionGroup);
        obj->CreateCOMObjects();

        pPackageCollection->AddElement((CComObject<CGOElement>*) obj);
        // AddElement Calls AddRef für obj
    }

    // Return-Value setzen
    CComQIPtr<IGOPackage> d_pPackage(obj);  // Calls QI
```

```
    if (d_pPackage)
    {
        return d_pPackage.CopyTo(ppPackage); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}
```

E.4.5 »CGOModel«

E.4.5.1 »CGOModel.h«

```
/* CGOModel.h

     Die Klasse CGOModel implementiert die COM-Schnittstelle IGOModel.
     CGOModel ist von CGOPackage abgeleitet und damit ebenfalls von CGOElement.

     CGOModel enthält das Package 'LogicalView' und eine Liste aller
     Assoziationen.
     Fachkonzept-Objekt ist MModel.

     Autor:       Carsten Schu
     Version      1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOMODEL_3A2CE12E00DC_INCLUDED
#define _INC_CGOMODEL_3A2CE12E00DC_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOPackage.h"

class ATL_NO_VTABLE CGOModel
: public CGOPackage                      // extends CGOElement
, public IDispatchImpl<IGOModel,&IID_IGOModel,&LIBID_GOLib>
{
private:
     CComObject<CGOPackage> *pLogicalView;
     CComObject<CGOCollection> *pAssociationCollection;   // wird nicht nach aussen zur
                                                          // Verfügung gestellt

public:
     CGOModel();
     ~CGOModel();

     CreateCOMObjects(); // Überschreibt Funktion aus CGOElement

public:
     STDMETHOD (FindElementWithID)(long ID, IGOElement** ppElement);
     STDMETHOD (GetLogicalView)(IGOPackage** ppPackage);


DECLARE_REGISTRY_RESOURCEID(IDR_GOMODEL)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOModel)
     COM_INTERFACE_ENTRY(IGOModel)
     COM_INTERFACE_ENTRY2(IDispatch, IGOPackage)
     COM_INTERFACE_ENTRY_CHAIN(CGOPackage)       // Interface-Map aus CGOPackage übernehmen
END_COM_MAP()

};

#endif /* _INC_CGOMODEL_3A2CE12E00DC_INCLUDED */
```

E.4.5.2 »CGOModel.cpp«

```
// CGOModel.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOModel.h"

#include "MModel.h"
#include "MAssociation.h"

#include "CGOCollection.h"
#include "CGOAssociation.h"
#include "CGOClass.h"


CGOModel::CGOModel()
{
    pLogicalView = NULL;
    pAssociationCollection = NULL;

    pClassCollection = NULL;   // Im Modell können keine Klassen existieren
    pPackageCollection = NULL;
}

CGOModel::~CGOModel()
{
    if (pClassCollection)
        pClassCollection->Release();

    if (pPackageCollection)
        pPackageCollection->Release();

    if (pLogicalView)
        pLogicalView->Release();

    if (pAssociationCollection)
        pAssociationCollection->Release();
}


CGOModel::CreateCOMObjects()
{
    HRESULT hr;

    // Im Model direkt können keine Klassen existieren.
    // Daher wird nur die Collection angelegt, jedoch keine Objekte darin abgelegt!
    hr = CComObject<CGOCollection>::CreateInstance(&pClassCollection);
    if (SUCCEEDED(hr))
    {
        pClassCollection->AddRef();
    }
    else
    {
        AfxMessageBox("Konnte 'ClassCollection' in 'CGOModel' nicht erzeugen.");
        pClassCollection = NULL;
    }

    hr = CComObject<CGOCollection>::CreateInstance(&pPackageCollection);
    if (SUCCEEDED(hr))
    {
        pPackageCollection->AddRef();
    }
    else
    {
        AfxMessageBox("Konnte 'PackageCollection' in 'CGOModel' nicht erzeugen.");
        pPackageCollection = NULL;
    }

    // ------------------------------------------------------------------
    // Logical View erzeugen!!!

    hr = CComObject<CGOPackage>::CreateInstance(&pLogicalView);
```

```
    if (SUCCEEDED(hr))
    {
        pLogicalView->AddRef();

        // Package Logical View suchen
        MModelElement *pMPackage = NULL;

        POSITION pos = ((MModel*)theMModelElement)->ownedElement.GetHeadPosition();
        while (pos != NULL)
        {
            pMPackage = ((MModel*)theMModelElement)->ownedElement.GetNext(pos);
            if (pMPackage->GetSubjectType() == Subject::StructurePackage)
            {
                if (((MPackage*)pMPackage)->GetStructureType() == MPackage::LogicalView)
                {
                    pLogicalView->SetMModelElement(pMPackage);
                    pLogicalView->SetActionGroup(pActionGroup);
                    pLogicalView->CreateCOMObjects();
                }
            }
        }
    }
    else
    {
        AfxMessageBox("Konnte 'LogicalView' in 'CGOModel' nicht erzeugen.");
        pLogicalView = NULL;
    }

    // ------------------------------------------------------------------------
    // Associations hier anlegen !

    hr = CComObject<CGOCollection>::CreateInstance(&pAssociationCollection);
    if (SUCCEEDED(hr))
    {
        pAssociationCollection->AddRef();

        // AssociationList aus dem MModel besorgen
        MModelElementList AssociationList;
        ((MModel*)theMModelElement)->GetObjectList(&AssociationList, Subject::Association);

        MAssociation* pA;
        MAssociationEnd* pAE1;
        MAssociationEnd* pAE2;

        // Eintragen aller Assoziationen
        POSITION pos = AssociationList.GetHeadPosition();
        while (pos != NULL)
        {
            pA = (MAssociation*)AssociationList.GetNext(pos);
            pAE1 = pA->connection.GetHead();
            pAE2 = pA->connection.GetTail();

            CComObject<CGOAssociation> *association = NULL;
            hr = CComObject<CGOAssociation>::CreateInstance(&association);
            if (SUCCEEDED(hr))
            {
                IGOElement* pElement1 = NULL;
                IGOElement* pElement2 = NULL;

                CComObject<CGOClass> *pClass1;
                CComObject<CGOClass> *pClass2;

                association->SetMModelElement((MModelElement*)pA);
                association->SetActionGroup(pActionGroup);
                association->CreateCOMObjects();

                pAssociationCollection->AddElement((CComObject<CGOElement>*) association);
                // AddElement Calls AddRef() für association

                // Verbindungen der Klassen zum Assoziations-Objekt erzeugen
                // Klassen müssen zu diesem Zeitpunkt bereits existieren!!!

                long ID1 = pAE1->gettype()->getID();
                FindElementWithID(ID1, &pElement1);// ruft intern AddRef() auf!
```

```
                    pClass1 = (CComObject<CGOClass>*) pElement1;
                    pClass1->AddAssociationEnd((CComObject<CGOElement>*) association);

                    long ID2 = pAE2->gettype()->getID();
                    FindElementWithID(ID2, &pElement2); // ruft intern AddRef() auf!

                    pClass2 = (CComObject<CGOClass>*) pElement2;
                    pClass2->AddAssociationEnd((CComObject<CGOElement>*) association);

                    association->TransmitClasses(pClass1,pClass2);

                    pElement1->Release();
                    pElement2->Release();
                }
            }
        }
        else
        {
            AfxMessageBox("Konnte in 'CGOModel' keine AssociationCollection anlegen.");
            pAssociationCollection = NULL;
        }
}



STDMETHODIMP CGOModel::GetLogicalView(IGOPackage** ppPackage)
{
    CComQIPtr<IGOPackage> d_pLogicalView (pLogicalView); // Calls QI

    if (d_pLogicalView)
    {
        return d_pLogicalView.CopyTo(ppPackage); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER
    return E_FAIL;
}


STDMETHODIMP CGOModel::FindElementWithID(long ID, IGOElement** ppElement)
{
    CComQIPtr<IGOElement> d_pElement;

    long ElementID = 0;
    this->GetID(&ElementID);

    if (ElementID == ID)  // Element ist das Modell
    {
        d_pElement = (CComObject<CGOElement>*)this;      // Calls QI
        if (d_pElement)
            return d_pElement.CopyTo(ppElement); // Calls AddRef !
    }

    // rekursiv das Modell durchsuchen, mit dem LogicalView beginnen
    d_pElement = pLogicalView->FindElementWithID(ID);    // Calls QI

    if (d_pElement)
        return d_pElement.CopyTo(ppElement); // Calls AddRef !

    // AssociationCollection durchsuchen -> Rollen untersuchen!
    if (pAssociationCollection)
    {
        long Anzahl = pAssociationCollection->GetCount();

        for (long index = 0; index< Anzahl; index++)
        {
            CComObject<CGOElement> *theElement = NULL;
            theElement = pAssociationCollection->GetElementAt(index);

            if (theElement != NULL)
            {
                // Assoziationen untersuchen ihre Rollen selbst

                d_pElement = ((CComObject<CGOAssociation>*)theElement)->FindElementWithID(ID);
```

```
                if (d_pElement)
                    return d_pElement.CopyTo(ppElement); // Calls AddRef !
            }
        }
    }

    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    ppElement = NULL;
    return S_OK;
}
```

E.4.6 »CGOClass«

E.4.6.1 »CGOClass.h«

```
/* CGOClass.h

     Die Klasse CGOClass implementiert die COM-Schnittstelle IGOClass.
     Sie repräsentiert eine Klasse (Fachkonzept-Objekt MClass).
     Sie beinhaltet Listen aller Attribute, aller Operationen, aller Oberklassen
     und aller Assoziationen.

     Autor:       Carsten Schu
     Version      1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOCLASS_3A2E350801E0_INCLUDED
#define _INC_CGOCLASS_3A2E350801E0_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class CGOCollection;


class ATL_NO_VTABLE CGOClass
: public CGOElement
, public IDispatchImpl<IGOClass,&IID_IGOClass,&LIBID_GOLib>
{
private:
     CComObject<CGOCollection> *pAttributeCollection;
     CComObject<CGOCollection> *pOperationCollection;
     CComObject<CGOCollection> *pSuperclassCollection;
     CComObject<CGOCollection> *pAssociationCollection;

public:
     CGOClass();
     ~CGOClass();

     CreateCOMObjects(); // Überschreibt Funktion aus CGOElement
     CComObject<CGOElement>* FindElementWithID(long ID);

     AddAssociationEnd(CComObject<CGOElement>* pElement);

public:
     STDMETHOD (get_Abstract)(VARIANT_BOOL* pVal);
     STDMETHOD (put_Abstract)(VARIANT_BOOL newVal);

     STDMETHOD (GetAttributes)(IGOCollection** ppCollection);

     STDMETHOD (GetOperations)(IGOCollection** ppCollection);

     STDMETHOD (AddAttribute)(BSTR Name, BSTR Type, BSTR InitValue, IGOAttribute** ppAttribute);
     STDMETHOD (DeleteAttribute)(IGOAttribute* pAttribute, VARIANT_BOOL* ret);

     STDMETHOD (GetAssociations)(IGOCollection** ppCollection);

     STDMETHOD (GetSuperclasses)(IGOCollection** ppCollection);


DECLARE_REGISTRY_RESOURCEID(IDR_GOCLASS)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOClass)
```

```
    COM_INTERFACE_ENTRY(IGOClass)
    COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
    COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};

#endif /* _INC_CGOCLASS_3A2E350801E0_INCLUDED */
```

E.4.6.2 »CGOClass.cpp«

```
// CGOClass.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOClass.h"

#include "CGOCollection.h"
#include "CGOAttribute.h"
#include "CGOOperation.h"
#include "CGOAssociation.h"

#include "MClass.h"
#include "MAssociationEnd.h"
#include "MGeneralization.h"

#include "ActionEditMClass.h"
#include "ActionCreateMAttribute.h"
#include "ActionEditMAttribute.h"
#include "ActionKillMAttribute.h"

#include "Application.h"
#include "docproject.h"
extern CApplication theApp;


CGOClass::CGOClass()
{
    pAttributeCollection = NULL;
    pOperationCollection = NULL;
    pSuperclassCollection = NULL;
    pAssociationCollection = NULL;
}

CGOClass::~CGOClass()
{
    if (pAttributeCollection)
        pAttributeCollection->Release();

    if (pOperationCollection)
        pOperationCollection->Release();

    if (pSuperclassCollection)
        pSuperclassCollection->Release();

    if (pAssociationCollection)
        pAssociationCollection->Release();
}

CGOClass::CreateCOMObjects()
{
    HRESULT hr;

    // ---------------------------------------------------------------------------
    // AttributeCollection
    hr = CComObject<CGOCollection>::CreateInstance(&pAttributeCollection);
    if (SUCCEEDED(hr))
    {
        pAttributeCollection->AddRef();

        MModelElement* pME;

        POSITION pos = ((MClass*)theMModelElement)->structuralFeature.GetHeadPosition();

        // Alle Elemente des Packages durchsuchen
        while (pos != NULL)
        {
            pME = (MModelElement*)((MClass*)theMModelElement)->structuralFeature.GetNext(pos);

            if (pME->GetSubjectType() == Subject::Attribute)
            {
                CComObject<CGOAttribute> *obj = NULL;
```

```
                hr = CComObject<CGOAttribute>::CreateInstance(&obj);
                if (SUCCEEDED(hr))
                {
                    obj->SetMModelElement(pME);
                    obj->SetActionGroup(pActionGroup);
                    obj->CreateCOMObjects();

                    pAttributeCollection->AddElement((CComObject<CGOElement>*) obj);
                    // AddElement Calls AddRef für obj
                }
            }
        }
    }
    else
    {
        AfxMessageBox("Konnte in 'CGOClass' keine AttributeCollection anlegen.");
        pAttributeCollection = NULL;
    }

    // ------------------------------------------------------------------------
    // OperationCollection
    hr = CComObject<CGOCollection>::CreateInstance(&pOperationCollection);
    if (SUCCEEDED(hr))
    {
        pOperationCollection->AddRef();

        MModelElement* pME;

        POSITION pos = ((MClass*)theMModelElement)->behavioralFeature.GetHeadPosition();

        // Alle Elemente des Packages durchsuchen
        while (pos != NULL)
        {
            pME = (MModelElement*)((MClass*)theMModelElement)->behavioralFeature.GetNext(pos);

            if (pME->GetSubjectType() == Subject::Operation)
            {
                CComObject<CGOOperation> *obj = NULL;
                hr = CComObject<CGOOperation>::CreateInstance(&obj);
                if (SUCCEEDED(hr))
                {
                    obj->SetMModelElement(pME);
                    obj->SetActionGroup(pActionGroup);
                    obj->CreateCOMObjects();

                    pOperationCollection->AddElement((CComObject<CGOElement>*) obj);
                    // AddElement Calls AddRef für obj
                }
            }
        }
    }
    else
    {
        AfxMessageBox("Konnte in 'CGOClass' keine OperationCollection anlegen.");
        pOperationCollection = NULL;
    }

    // ------------------------------------------------------------------------
    // AssociationCollection
    hr = CComObject<CGOCollection>::CreateInstance(&pAssociationCollection);
    if (SUCCEEDED(hr))
    {
        pAssociationCollection->AddRef();
        // Nur anlegen, gefüllt wird diese Liste über AddAssociationEnd()
        // aus CGOModel heraus
    }
    else
    {
        AfxMessageBox("Konnte in 'CGOClass' keine AssociationCollection anlegen.");
        pAssociationCollection = NULL;
    }

    // ------------------------------------------------------------------------
    // SuperclassCollection
    hr = CComObject<CGOCollection>::CreateInstance(&pSuperclassCollection);
```

```
    if (SUCCEEDED(hr))
    {
        pSuperclassCollection->AddRef();

        MModelElement* pME;

        POSITION pos = ((MClass*)theMModelElement)->generalization.GetHeadPosition();

        while (pos != NULL)
        {
            pME = (MModelElement*)((MClass*)theMModelElement)->generalization.GetNext(pos)-
>getparent();

            CComObject<CGOClass> *obj = NULL;
            hr = CComObject<CGOClass>::CreateInstance(&obj);
            if (SUCCEEDED(hr))
            {
                obj->SetMModelElement(pME);
                obj->SetActionGroup(pActionGroup);
                obj->CreateCOMObjects();

                pSuperclassCollection->AddElement((CComObject<CGOElement>*) obj);
                // AddElement Calls AddRef für obj
            }
        }
    }
    else
    {
        AfxMessageBox("Konnte in 'CGOClass' keine SuperclassCollection anlegen.");
        pSuperclassCollection = NULL;
    }
}



CComObject<CGOElement>* CGOClass::FindElementWithID(long ID)
{
    long ElementID = 0;
    this->GetID(&ElementID);

    if (ElementID == ID)
        return ((CComObject<CGOElement>*)this);

    // Innerhalb der Klasse alle Unterstrukturen durchsuchen

    if (pAttributeCollection)
    {
        long Anzahl = pAttributeCollection->GetCount();

        for (long index = 0; index < Anzahl; index++)
        {
            CComObject<CGOElement> *theElement = NULL;

            theElement = pAttributeCollection->GetElementAt(index);

            if (theElement != NULL)
            {
                long ElementID = 0;
                theElement->GetID(&ElementID);

                if (ElementID == ID)
                    return theElement;
            }
        }
    }

    if (pOperationCollection)
    {
        long Anzahl = pOperationCollection->GetCount();

        for (long index = 0; index < Anzahl; index++)
        {
            CComObject<CGOElement> *theElement = NULL;

            theElement = pOperationCollection->GetElementAt(index);
```

```
            // Parameter untersuchen in CGOOperation

            if (theElement != NULL)
            {
                CComObject<CGOElement> *searchElement = NULL;

                searchElement = ((CComObject<CGOOperation>*)theElement)->FindElementWithID(ID);

                if (searchElement != NULL)
                    return searchElement;
            }
        }
    }
    return NULL; // nichts gefunden!
}

CGOClass::AddAssociationEnd(CComObject<CGOElement>* pElement)
{
    pAssociationCollection->AddElement(pElement);
    // AddElement Calls AddRef() für pElement
}



STDMETHODIMP CGOClass::get_Abstract(VARIANT_BOOL* pVal)
{
    if (pVal)
    {
        *pVal = ((MClass*)theMModelElement)->isAbstract;
        return S_OK;
    }
    return E_POINTER;
}

STDMETHODIMP CGOClass::put_Abstract(VARIANT_BOOL newVal)
{
    ActionEditMClass* pEdit = new ActionEditMClass((MClass*)theMModelElement);
    pActionGroup->AddSubAction(pEdit);

    pEdit->SetChangedAbstract(newVal);
    pEdit->Do();

    return S_OK;
}


STDMETHODIMP CGOClass::GetAttributes(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pAttributeCollection (pAttributeCollection);

    if (d_pAttributeCollection)
    {
        return d_pAttributeCollection.CopyTo(ppCollection); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}


STDMETHODIMP CGOClass::AddAttribute(BSTR Name, BSTR Type, BSTR InitValue, IGOAttribute**
ppAttribute)
{
    // Neues Attribut anlegen
    ActionCreateMAttribute* pCreate = new ActionCreateMAttribute((MClass*)theMModelElement);
    pActionGroup->AddSubAction(pCreate);
    pCreate->Do();

    MAttribute* pAttribute = pCreate->GetMAttribute();

    // Eindeutigkeit des Namens prüfen
    if (pAttribute->getowner()->DoesNameExist(Name, Subject::Dummy, pAttribute))
    {
```

```
        AfxMessageBox("Der Name \"" + CString(Name) + "\" wird bereits für ein Member
verwendet.\nBitte wählen Sie einen anderen.");

        pCreate->UnDo();
        pActionGroup->RemoveLastAction();
        delete pCreate;

        return E_FAIL;
    }

    // Attribut bearbeiten
    ActionEditMAttribute* pEdit = new ActionEditMAttribute(pAttribute);
    pActionGroup->AddSubAction(pEdit);

    CDocProject* pDoc = theApp.GetDocument();
    MClassifier* pType = pDoc->GetDataType(Type);

    MModelElement::Visibility vis = MFeature::Private;

    pEdit->SetChangedData(Name, pType, InitValue, vis, false);
    pEdit->Do();

    // ComObject zur Liste hinzufügen
    CComObject<CGOAttribute> *obj = NULL;
    HRESULT hr = CComObject<CGOAttribute>::CreateInstance(&obj);
    if (SUCCEEDED(hr))
    {
        obj->SetMModelElement(pAttribute);
        obj->SetActionGroup(pActionGroup);
        obj->CreateCOMObjects();

        pAttributeCollection->AddElement((CComObject<CGOElement>*) obj);
        // AddElement Calls AddRef für obj
    }

    // Return-Value setzen
    CComQIPtr<IGOAttribute> d_pAttribute(obj);  // Calls QI

    if (d_pAttribute)
    {
        return d_pAttribute.CopyTo(ppAttribute); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}

STDMETHODIMP CGOClass::DeleteAttribute(IGOAttribute* pAttribute, VARIANT_BOOL* ret)
{
    if (ret)
    {
        CComObject<CGOElement> *theAttribute = NULL;

        // ComObject suchen
        if (pAttributeCollection)
        {
            long Anzahl = pAttributeCollection->GetCount();

            for (long index = 0; index < Anzahl; index++)
            {
                CComObject<CGOElement> *theElement = NULL;
                theElement = pAttributeCollection->GetElementAt(index);

                if (theElement != NULL)
                {
                    long ElementID = 0;
                    theElement->GetID(&ElementID);

                    CComQIPtr<IGOElement> pQIAttribute (pAttribute); // Calls QI

                    long AttributeID = 0;
                    pQIAttribute->GetID(&AttributeID);

                    if (ElementID == AttributeID)
```

```
                        {
                            theAttribute = theElement;
                            pAttributeCollection->RemoveElementAt(index);
                        }
                    }
                }
            }

            *ret = VARIANT_FALSE;
            if (theAttribute)
            {
                ActionKillMAttribute* pKill = new ActionKillMAttribute((MAttribute*)theAttribute-
>theMModelElement);
                pActionGroup->AddSubAction(pKill);
                pKill->Do();

                *ret = VARIANT_TRUE;  // Löschen konnte durchgeführt werden
            }

            return S_OK;
        }
        else
        {
            return E_POINTER;
        }
}


STDMETHODIMP CGOClass::GetOperations(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pOperationCollection (pOperationCollection);      // Calls QI

    if (d_pOperationCollection)
    {
        return d_pOperationCollection.CopyTo(ppCollection); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}

STDMETHODIMP CGOClass::GetAssociations(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pAssociationCollection(pAssociationCollection); // Calls QI

    if (d_pAssociationCollection)
    {
        return d_pAssociationCollection.CopyTo(ppCollection); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}

STDMETHODIMP CGOClass::GetSuperclasses(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pSuperclassCollection(pSuperclassCollection); // Calls QI

    if (d_pSuperclassCollection)
    {
        return d_pSuperclassCollection.CopyTo(ppCollection); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}
```

E.4.7 »CGOAttribute«

E.4.7.1 »CGOAttribute.h«

```
/* CGOAttribute.h

     Die Klasse CGOAttribute implementiert die COM-Schnittstelle IGOAttribute.
     Sie repräsentiert das Attribut einer Klasse (Fachkonzept-Objekt MAttribute).

     Autor:            Carsten Schu
     Version           1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOATTRIBUTE_3A2E35DB03AC_INCLUDED
#define _INC_CGOATTRIBUTE_3A2E35DB03AC_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class ATL_NO_VTABLE CGOAttribute
: public CGOElement
, public IDispatchImpl<IGOAttribute,&IID_IGOAttribute,&LIBID_GOLib>
{
public:
     CGOAttribute();
     ~CGOAttribute();

public:
     STDMETHOD (get_Type)(BSTR* pVal);
     STDMETHOD (put_Type)(BSTR newVal);

     STDMETHOD (get_Static)(VARIANT_BOOL* pVal);
     STDMETHOD (put_Static)(VARIANT_BOOL newVal);

     STDMETHOD (get_Derived)(VARIANT_BOOL* pVal);
     STDMETHOD (put_Derived)(VARIANT_BOOL newVal);

     STDMETHOD (get_InitValue)(BSTR* pVal);
     STDMETHOD (put_InitValue)(BSTR newVal);

DECLARE_REGISTRY_RESOURCEID(IDR_GOATTRIBUTE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOAttribute)
     COM_INTERFACE_ENTRY(IGOAttribute)
     COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
     COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};

#endif /* _INC_CGOATTRIBUTE_3A2E35DB03AC_INCLUDED */
```

E.4.7.2 »CGOAttribute.cpp«

```
// CGOAttribute.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOAttribute.h"

#include "ActionEditMAttribute.h"

#include "MAttribute.h"

#include "Application.h"
#include "docproject.h"
extern CApplication theApp;


CGOAttribute::CGOAttribute()
{
}

CGOAttribute::~CGOAttribute()
{
}


STDMETHODIMP CGOAttribute::get_Type(BSTR* pVal)
{
    if (pVal)
    {
        CString text = ((MAttribute*)theMModelElement)->gettype()->getname();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGOAttribute::put_Type(BSTR newVal)
{
    CString text = CString(newVal);
    CDocProject* pDoc = theApp.GetDocument();
    MClassifier* pType = pDoc->GetDataType(text);

    if (pType)
    {
        ActionEditMAttribute* pEdit = new ActionEditMAttribute((MAttribute*)theMModelElement);
        pActionGroup->AddSubAction(pEdit);

        pEdit->SetChangedType(pType);
        pEdit->Do();

        return S_OK;
    }
    else
    {
        return E_FAIL;
    }
}


STDMETHODIMP CGOAttribute::get_InitValue(BSTR* pVal)
{
    if (pVal)
    {
        CString text = ((MAttribute*)theMModelElement)->getinitialValue();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
```

```
        return E_POINTER;
    }
}

STDMETHODIMP CGOAttribute::put_InitValue(BSTR newVal)
{
    CString text = CString(newVal);

    ActionEditMAttribute* pEdit = new ActionEditMAttribute((MAttribute*)theMModelElement);
    pActionGroup->AddSubAction(pEdit);

    pEdit->SetChangedDefault(text);
    pEdit->Do();

    return S_OK;
}



// Klassen-Attribute werden in GO momentan noch nicht unterstützt
STDMETHODIMP CGOAttribute::get_Static(VARIANT_BOOL* pVal)
{
    return E_NOTIMPL;
}

STDMETHODIMP CGOAttribute::put_Static(VARIANT_BOOL newVal)
{
    return E_NOTIMPL;
}


// Abgeleitete Attribute werden in GO momentan noch nicht unterstützt
STDMETHODIMP CGOAttribute::get_Derived(VARIANT_BOOL* pVal)
{
    return E_NOTIMPL;
}

STDMETHODIMP CGOAttribute::put_Derived(VARIANT_BOOL newVal)
{
    return E_NOTIMPL;
}
```

E.4.8 »CGOOperation«

E.4.8.1 »CGOOperation.h«

```
/* CGOOperation.h

    Die Klasse CGOOperation implementiert die COM-Schnittstelle IGOOperation.

    CGOOperation repräsentiert die Operation einer Klasse
    (Fachkonzept-Objekt MOperation).

    Sie enthält eine Liste aller zugehörigen Parameter.

    Autor:       Carsten Schu
    Version      1.0
    Erstellt am: 02.01.2001

    Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOOPERATION_3A2E361F03AC_INCLUDED
#define _INC_CGOOPERATION_3A2E361F03AC_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class CGOCollection;


class ATL_NO_VTABLE CGOOperation
: public CGOElement
, public IDispatchImpl<IGOOperation,&IID_IGOOperation,&LIBID_GOLib>
{
private:
    CComObject<CGOCollection> *pParameterCollection;

public:
    CGOOperation();
    ~CGOOperation();

    CreateCOMObjects();
    CComObject<CGOElement>* FindElementWithID(long ID);

public:
    STDMETHOD (get_ReturnType)(BSTR* pVal);
    STDMETHOD (put_ReturnType)(BSTR newVal);

    STDMETHOD (GetParameters)(IGOCollection** ppCollection);

    STDMETHOD (RemoveAllParameters)();

    STDMETHOD (AddParameter)(BSTR Name, BSTR Type, BSTR InitValue, long Position, IGOParameter**
ppParameter);

DECLARE_REGISTRY_RESOURCEID(IDR_GOOPERATION)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOOperation)
    COM_INTERFACE_ENTRY(IGOOperation)
    COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
    COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};
#endif /* _INC_CGOOPERATION_3A2E361F03AC_INCLUDED */
```

E.4.8.2 »CGOOperation.cpp«

```
// CGOOperation.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"

#include "CGOOperation.h"
#include "CGOParameter.h"
#include "CGOCollection.h"

#include "MOperation.h"

#include "ActionEditMOperation.h"
#include "ActionCreateMParameter.h"
#include "ActionEditMParameter.h"
#include "ActionKillMParameter.h"

#include "Application.h"
#include "docproject.h"
extern CApplication theApp;


CGOOperation::CGOOperation()
{
    pParameterCollection = NULL;
}

CGOOperation::~CGOOperation()
{
    if (pParameterCollection)
        pParameterCollection->Release();
}


CGOOperation::CreateCOMObjects()
{
    HRESULT hr;

    // ---------------------------------------------------------------------------
    // ParameterCollection
    hr = CComObject<CGOCollection>::CreateInstance(&pParameterCollection);
    if (SUCCEEDED(hr))
    {
        pParameterCollection->AddRef();

        MModelElement* pME;

        POSITION pos = ((MOperation*)theMModelElement)->parameter.GetHeadPosition();

        // Alle Elemente des Packages durchsuchen
        while (pos != NULL)
        {
            pME = (MModelElement*)((MOperation*)theMModelElement)->parameter.GetNext(pos);

            if (pME->GetSubjectType() == Subject::Parameter)
            {
                CComObject<CGOParameter> *obj = NULL;
                hr = CComObject<CGOParameter>::CreateInstance(&obj);
                if (SUCCEEDED(hr))
                {
                    obj->SetMModelElement(pME);
                    obj->SetActionGroup(pActionGroup);
                    obj->CreateCOMObjects();

                    pParameterCollection->AddElement((CComObject<CGOElement>*) obj);
                    // AddElement Calls AddRef für obj
                }
            }
        }
    }
    else
    {
```

```
        AfxMessageBox("Konnte in 'CGOOperation' keine ParameterCollection anlegen.");
        pParameterCollection = NULL;
    }
}


CComObject<CGOElement>* CGOOperation::FindElementWithID(long ID)
{
    long ElementID = 0;
    this->GetID(&ElementID);

    if (ElementID == ID)
        return ((CComObject<CGOElement>*)this);

    if (pParameterCollection)
    {
        long Anzahl = pParameterCollection->GetCount();

        for (long index = 0; index < Anzahl; index++)
        {
            CComObject<CGOElement> *theElement = NULL;

            theElement = pParameterCollection->GetElementAt(index);

            if (theElement != NULL)
            {
                long ElementID = 0;
                theElement->GetID(&ElementID);

                if (ElementID == ID)
                    return theElement;
            }
        }
    }
    return NULL; // nichts gefunden
}


STDMETHODIMP CGOOperation::get_ReturnType(BSTR* pVal)
{
    if (pVal)
    {
        CString text = ((MOperation*)theMModelElement)->getreturntype()->getname();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGOOperation::put_ReturnType(BSTR newVal)
{
    CString text = CString(newVal);
    CDocProject* pDoc = theApp.GetDocument();
    MClassifier* pType = pDoc->GetDataType(text);

    if (pType)
    {
        ActionEditMOperation* pEdit = new ActionEditMOperation((MOperation*)theMModelElement);
        pActionGroup->AddSubAction(pEdit);

        pEdit->SetChangedReturnType(pType);
        pEdit->Do();

        return S_OK;
    }
    else
    {
        return E_FAIL;
    }
}
```

```
STDMETHODIMP CGOOperation::GetParameters(IGOCollection** ppCollection)
{
    CComQIPtr<IGOCollection> d_pParameterCollection(pParameterCollection); // Calls QI

    if (d_pParameterCollection)
    {
        return d_pParameterCollection.CopyTo(ppCollection); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}


STDMETHODIMP CGOOperation::RemoveAllParameters()
{
    if (pParameterCollection)
    {
        long Anzahl = pParameterCollection->GetCount();

        for (long index = 0; index < Anzahl; index++)
        {
            CComObject<CGOElement> *theParameter = NULL;
            theParameter = pParameterCollection->GetElementAt(index);

            ActionKillMParameter* pKill = new ActionKillMParameter((MParameter*)theParameter->
theMModelElement);
            pActionGroup->AddSubAction(pKill);
            pKill->Do();
        }

        pParameterCollection->RemoveAllElements();

        return S_OK;
    }
    return E_FAIL;
}


// Position wird in GO noch nicht unterstützt!!!
STDMETHODIMP CGOOperation::AddParameter(BSTR Name, BSTR Type, BSTR InitValue, long Position,
IGOParameter** ppParameter)
{
    // Neuen Parameter anlegen
    ActionCreateMParameter* pCreate = new ActionCreateMParameter((MOperation*)theMModelElement);
    pActionGroup->AddSubAction(pCreate);
    pCreate->Do();

    MParameter* pParameter = pCreate->GetMParameter();

    // evtl. noch Eindeutigkeit des Namens prüfen

    // Parameter bearbeiten
    ActionEditMParameter* pEdit = new ActionEditMParameter(pParameter);
    pActionGroup->AddSubAction(pEdit);

    CDocProject* pDoc = theApp.GetDocument();
    MClassifier* pType = pDoc->GetDataType(Type);

    pEdit->SetChangedData(Name, pType, InitValue);
    pEdit->Do();

    // ComObject zur Liste hinzufügen
    CComObject<CGOParameter> *obj = NULL;
    HRESULT hr = CComObject<CGOParameter>::CreateInstance(&obj);
    if (SUCCEEDED(hr))
    {
        obj->SetMModelElement(pParameter);
        obj->SetActionGroup(pActionGroup);
        obj->CreateCOMObjects();

        pParameterCollection->AddElement((CComObject<CGOElement>*) obj);
        // AddElement Calls AddRef für obj
    }
```

```
    // Return-Value setzen
    CComQIPtr<IGOParameter> d_pParameter(obj);  // Calls QI

    if (d_pParameter)
    {
        return d_pParameter.CopyTo(ppParameter); // Calls AddRef !
    }
    // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück
    // andernfalls E_POINTER

    return E_FAIL;
}
```

E.4.9 »CGOParameter«

E.4.9.1 »CGOParameter.h«

```
/* CGOParameter.h

     Die Klasse CGOParameter implementiert die COM-Schnittstelle IGOParameter.

     CGOParameter repräsentiert einen Parameter einer Operation
     (Fachkonzept-Objekt MParameter).

     Autor:       Carsten Schu
     Version      1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOPARAMETER_3A2E363201F4_INCLUDED
#define _INC_CGOPARAMETER_3A2E363201F4_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class ATL_NO_VTABLE CGOParameter
: public CGOElement
, public IDispatchImpl<IGOParameter,&IID_IGOParameter,&LIBID_GOLib>
{
public:
     CGOParameter();
     ~CGOParameter();

public:
     STDMETHOD (get_Type)(BSTR* pVal);
     STDMETHOD (put_Type)(BSTR newVal);


DECLARE_REGISTRY_RESOURCEID(IDR_GOPARAMETER)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOParameter)
     COM_INTERFACE_ENTRY(IGOParameter)
     COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
     COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};

#endif /* _INC_CGOPARAMETER_3A2E363201F4_INCLUDED */
```

E.4.9.2 »CGOParameter.cpp«

```
// CGOParameter.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOParameter.h"

#include "MParameter.h"

#include "ActionEditMParameter.h"

#include "Application.h"
#include "docproject.h"
extern CApplication theApp;


CGOParameter::CGOParameter()
{
}

CGOParameter::~CGOParameter()
{
}


STDMETHODIMP CGOParameter::get_Type(BSTR* pVal)
{
    if (pVal)
    {
        CString text = ((MParameter*)theMModelElement)->gettype()->getname();
        *pVal = text.AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGOParameter::put_Type(BSTR newVal)
{
    CString text = CString(newVal);
    CDocProject* pDoc = theApp.GetDocument();
    MClassifier* pType = pDoc->GetDataType(text);

    if (pType)
    {
        ActionEditMParameter* pEdit = new ActionEditMParameter((MParameter*)theMModelElement);
        pActionGroup->AddSubAction(pEdit);

        pEdit->SetChangedType(pType);
        pEdit->Do();

        return S_OK;
    }
    else
    {
        return E_FAIL;
    }
}
```

E.4.10 »CGOAssociation«

E.4.10.1 »CGOAssociation.h«

```
/* CGOAssociation.h

     Die Klasse CGOAssociation implementiert die COM-Schnittstelle IGOAssociation.
     Sie repräsentiert eine Assoziation (Fachkonzept-Objekt MAssociation).
     Sie enthält die beiden Assoziationsenden (Rollen) der Assoziation.

     Autor:       Carsten Schu
     Version      1.0
     Erstellt am: 02.01.2001

     Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOASSOCIATION_3A2E35ED019A_INCLUDED
#define _INC_CGOASSOCIATION_3A2E35ED019A_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"
#include "CGORole.h"

class ATL_NO_VTABLE CGOAssociation
: public CGOElement
, public IDispatchImpl<IGOAssociation,&IID_IGOAssociation,&LIBID_GOLib>
{
private:
     CComObject<CGORole> *pGORole1;
     CComObject<CGORole> *pGORole2;

public:
     CGOAssociation();
     ~CGOAssociation();

     CreateCOMObjects();
     CComObject<CGOElement>* FindElementWithID(long ID);

     TransmitClasses(CComObject<CGOClass>* pClass1, CComObject<CGOClass>* pClass2);

public:
     STDMETHOD (GetRole1)(IGORole** ppRole1);
     STDMETHOD (GetRole2)(IGORole** ppRole2);


DECLARE_REGISTRY_RESOURCEID(IDR_GOASSOCIATION)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGOAssociation)
     COM_INTERFACE_ENTRY(IGOAssociation)
     COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
     COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};

#endif /* _INC_CGOASSOCIATION_3A2E35ED019A_INCLUDED */
```

E.4.10.2 »CGOAssociation.cpp«

```
// CGOAssociation.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGOAssociation.h"

#include "MAssociation.h"

CGOAssociation::CGOAssociation()
{
     pGORole1 = NULL;
     pGORole2 = NULL;
}

CGOAssociation::~CGOAssociation()
{
     if (pGORole1)
     {
          pGORole1->Release();
     }

     if (pGORole2)
     {
          pGORole2->Release();
     }
}


CGOAssociation::CreateCOMObjects()
{
     HRESULT hr;

     // ------------------------------------------------------------------------
     // Role1
     hr = CComObject<CGORole>::CreateInstance(&pGORole1);
     if (SUCCEEDED(hr))
     {
          pGORole1->AddRef();

          MModelElement* pME = ((MAssociation*)theMModelElement)->connection.GetHead();

          pGORole1->SetMModelElement(pME);
          pGORole1->SetActionGroup(pActionGroup);
          pGORole1->CreateCOMObjects();
     }
     else
     {
          AfxMessageBox("Konnte in 'CGOAssoiation' keine Role1 anlegen.");
          pGORole1 = NULL;
     }

     // ------------------------------------------------------------------------
     // Role2
     hr = CComObject<CGORole>::CreateInstance(&pGORole2);
     if (SUCCEEDED(hr))
     {
          pGORole2->AddRef();

          MModelElement* pME = ((MAssociation*)theMModelElement)->connection.GetTail();

          pGORole2->SetMModelElement(pME);
          pGORole2->SetActionGroup(pActionGroup);
          pGORole2->CreateCOMObjects();
     }
     else
     {
          AfxMessageBox("Konnte in 'CGOAssociation' keine Role2 anlegen.");
          pGORole2 = NULL;
     }

     // Die jeweiligen Other-Roles setzen
```

```
    if (pGORole1 && pGORole2)
    {
        pGORole1->SetOtherRole(pGORole2);
        pGORole2->SetOtherRole(pGORole1);
    }
}


CComObject<CGOElement>* CGOAssociation::FindElementWithID(long ID)
{
    long ElementID = 0;

    this->GetID(&ElementID);

    if (ElementID == ID)
        return ((CComObject<CGOElement>*) this);

    if (pGORole1)
    {
        //long ElementID = 0;
        pGORole1->GetID(&ElementID);

        if (ElementID == ID)
            return ((CComObject<CGOElement>*) pGORole1);
    }

    if (pGORole2)
    {
        pGORole2->GetID(&ElementID);

        if (ElementID == ID)
            return ((CComObject<CGOElement>*) pGORole2);
    }

    return NULL; // nichts gefunden
}


CGOAssociation::TransmitClasses(CComObject<CGOClass>* pClass1, CComObject<CGOClass>* pClass2)
{
    if (pGORole1 && pGORole2)
    {
        pGORole1->SetClass(pClass1);
        pGORole2->SetClass(pClass2);
    }
}



STDMETHODIMP CGOAssociation::GetRole1(IGORole** ppRole1)
{
    CComQIPtr<IGORole> d_pGORole1 (pGORole1);   // Calls QI

    if (d_pGORole1)
    {
        return d_pGORole1.CopyTo(ppRole1); // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück

    return E_FAIL;
}

STDMETHODIMP CGOAssociation::GetRole2(IGORole** ppRole2)
{
    CComQIPtr<IGORole> d_pGORole2 (pGORole2);   // Calls QI

    if (d_pGORole2)
    {
        return d_pGORole2.CopyTo(ppRole2); // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück

    return E_FAIL;
}
```

E.4.11 »CGORole«

E.4.11.1 »CGORole.h«

```
/* CGORole.h

    Die Klasse CGORole implementiert die COM-Schnittstelle IGORole.

    CGORole repräsentiert die Rolle einer Assoziation.
    Im Fachkonzept entspricht eine Rolle einem MAssociationEnd.

    Autor:       Carsten Schu
    Version      1.0
    Erstellt am: 02.01.2001

    Copyright (c) 2000 / 2001 by Carsten Schu
*/

#if defined (_MSC_VER) && (_MSC_VER >= 1000)
#pragma once
#endif
#ifndef _INC_CGOROLE_3A2E36120280_INCLUDED
#define _INC_CGOROLE_3A2E36120280_INCLUDED

#include "resource.h"
#include "GO_i.h"

#include "CGOElement.h"

class CGOClass;

class ATL_NO_VTABLE CGORole
: public CGOElement
, public IDispatchImpl<IGORole,&IID_IGORole,&LIBID_GOLib>
{
private:
    CComObject<CGOClass> *pClass;
    CComObject<CGORole> *pOtherRole;

public:
    CGORole();
    ~CGORole();

    SetOtherRole(CComObject<CGORole> *pRole);
    SetClass(CComObject<CGOClass> *pC);

public:
    STDMETHOD (get_Cardinality)(BSTR* pVal);
    STDMETHOD (put_Cardinality)(BSTR newVal);

    STDMETHOD (get_Navigable)(VARIANT_BOOL* pVal);
    STDMETHOD (put_Navigable)(VARIANT_BOOL newVal);

    STDMETHOD (GetClass)(IGOClass** ppClass);

    STDMETHOD (GetOtherRole)(IGORole** ppRole);

public:
DECLARE_REGISTRY_RESOURCEID(IDR_GOROLE)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(CGORole)
    COM_INTERFACE_ENTRY(IGORole)
    COM_INTERFACE_ENTRY2(IDispatch, IGOElement)
    COM_INTERFACE_ENTRY_CHAIN(CGOElement)   // Interface-Map aus CGOElement übernehmen
END_COM_MAP()

};
#endif /* _INC_CGOROLE_3A2E36120280_INCLUDED */
```

E.4.11.2 »CGORole.cpp«

```
// CGORole.cpp
// Copyright (c) 2000 / 2001 by Carsten Schu

#include "stdafx.h"
#include "CGORole.h"

#include "CGOClass.h"

#include "MAssociation.h"

#include "ActionEditMAssociation.h"


CGORole::CGORole()
{
    pClass = NULL;
    pOtherRole = NULL;
}

CGORole::~CGORole()
{
    // Achtung: Gegenseitige Referenz der beiden Rollen einer
    // Assoziation aufeinander!!!
    // Deshalb auch kein Release() für pOtherRole

    // Ebenfalls kein Release() für pClass (s.u. SetClass())
}


CGORole::SetOtherRole(CComObject<CGORole> *pRole)
{
    // Achtung gegenseitige Referenz der Rollen aufeinander!!!
    // deshalb kein AddRef(), da sonst Deadlock entsteht
    // (Zerstörung der Objekte würde unmöglich)

    pOtherRole = pRole;
}

CGORole::SetClass(CComObject<CGOClass> *pC)
{
    // analog zu SetOtherRole, allerdings hier keine direkte gegenseitige
    // Referenz, sondern Ringreferenz (Role-Association-Class-Association-Role)

    pClass = pC;
}


STDMETHODIMP CGORole::get_Cardinality(BSTR* pVal)
{
    if (pVal)
    {
        *pVal = ((MAssociationEnd*)theMModelElement)-
>getmultiplicity().GetString().AllocSysString();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGORole::put_Cardinality(BSTR newVal)
{
    MAssociationEnd* pAE = (MAssociationEnd*)theMModelElement;
    MAssociation* pA = pAE->gettheAssociation();

    ActionEditMAssociation* pEdit = new ActionEditMAssociation(pA);
    pActionGroup->AddSubAction(pEdit);

    if (pA->connection.GetHead() == pAE)
    {   // Rolle1 ändern
```

```
        pEdit->SetChangedRolle1Cardinality(newVal);
    }
    else
    {   // Rolle2 ändern
        pEdit->SetChangedRolle2Cardinality(newVal);
    }

    pEdit->Do();

    return S_OK;
}

STDMETHODIMP CGORole::get_Navigable(VARIANT_BOOL* pVal)
{
    if (pVal)
    {
        *pVal = ((MAssociationEnd*)theMModelElement)->getisNavigable();
        return S_OK;
    }
    else
    {
        return E_POINTER;
    }
}

STDMETHODIMP CGORole::put_Navigable(VARIANT_BOOL newVal)
{
    MAssociationEnd* pAE = (MAssociationEnd*)theMModelElement;
    MAssociation* pA = pAE->gettheAssociation();

    ActionEditMAssociation* pEdit = new ActionEditMAssociation(pA);
    pActionGroup->AddSubAction(pEdit);

    if (pA->connection.GetHead() == pAE)
    {   // Rolle1 ändern
        pEdit->SetChangedRolle1Navigable(newVal);
    }
    else
    {   // Rolle2 ändern
        pEdit->SetChangedRolle2Navigable(newVal);
    }

    pEdit->Do();

    return S_OK;
}

STDMETHODIMP CGORole::GetClass(IGOClass** ppClass)
{
    CComQIPtr<IGOClass> d_pClass (pClass);  // Calls QI

    if (d_pClass)
    {
        return d_pClass.CopyTo(ppClass); // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück

    return E_FAIL;
}

STDMETHODIMP CGORole::GetOtherRole(IGORole** ppRole)
{
    CComQIPtr<IGORole> d_pOtherRole (pOtherRole);    // Calls QI

    if (d_pOtherRole)
    {
        return d_pOtherRole.CopyTo(ppRole); // Calls AddRef !
    }   // CopyTo liefert bei erfolgreicher Ausführung S_OK zurück

    return E_FAIL;
}
```

E.5 Test-Client in Visual Basic

```
Private theApp As GOApplication 'Declared only in this module!!!
Private theModel As GOModel
Private logicalView As GOPackage


Private Sub GOStarten_Click()

    On Error GoTo ErrorHandler   ' Enable error-handling routine.

    Set theApp = New GOApplication
    Set theModel = theApp.GetCurrentModel()
    Set logicalView = theModel.GetLogicalView()

    FindElementWithID.Visible = True
    ModelInfo.Visible = True
    Traversiere.Visible = True
    GOBeenden.Visible = True
    GOStarten.Visible = False

Exit Sub

ErrorHandler:   ' Error-handling routine.

    Dim text As String
    Dim FehlerNummer As String

    FehlerNummer = Err.Number

    text = "Fehler: " + Chr(13)
    text = text + Err.Description + Chr(13)

    MsgBox (text)
    MsgBox ("GO muss gestartet sein und ein Modell muss in GO existieren!")

    Set theApp = Nothing
    Set theModel = Nothing
    Set logicalView = Nothing
End Sub



Private Sub GOBeenden_Click()
    FindElementWithID.Visible = False
    ModelInfo.Visible = False
    Traversiere.Visible = False
    GOBeenden.Visible = False
    GOStarten.Visible = True

    Set theModel = Nothing
    Set theApp = Nothing
    Set logicalView = Nothing
End Sub



Private Sub FindElementWithID_Click()
    Dim Message, Title, Default As String
    Dim IDString As String
    Dim ID As Long

    Message = "Bitte Geben Sie eine ID ein."
    Title = "FindElementWithID"
    Default = "1"
    IDString = InputBox(Message, Title, Default)

    If IDString <> "" Then
        ID = IDString

        Dim FindElement As GOElement
        Set FindElement = theModel.FindElementWithID(ID)
```

```
        If Not FindElement Is Nothing Then
            MsgBox "Element gefunden: " + FindElement.Name
        Else
            MsgBox "Leider kein Element mit ID " + IDString + " gefunden: "
        End If
    End If
End Sub

Private Sub ModelInfo_Click()
    Dim TheElement As GOElement
    Set TheElement = theModel

    MsgBox ("Model-Name: " + TheElement.Name)

    ' Wert verändern
    TheElement.Name = "Neuer Test-Name"
    MsgBox ("Geänderter Name: " + TheElement.Name)

End Sub


Private Sub Traversiere_Click()
    Call TraversierePackages(logicalView)
End Sub


Private Sub TraversierePackages(thePackage As GOPackage)
    Dim thePackageElement As GOElement

    Dim ClassCollection As GOCollection
    Dim theClass As GOClass
    Dim theClassElement As GOElement

    Dim SuperclassCollection As GOCollection
    Dim SuperclassElement As GOElement

    Dim AssociationCollection As GOCollection
    Dim theAssociation As GOAssociation
    Dim Role1 As GORole
    Dim Role2 As GORole
    Dim Role1Element As GOElement
    Dim Role2Element As GOElement

    Dim AttributeCollection As GOCollection
    Dim theAttribute As GOAttribute
    Dim theAttributeElement As GOElement

    Dim OperationCollection As GOCollection
    Dim theOperation As GOOperation
    Dim theOperationElement As GOElement

    Dim ParameterCollection As GOCollection
    Dim theParameter As GOParameter
    Dim theParameterElement As GOElement

    Dim SubPackageCollection As GOCollection
    Dim SubPackage As GOPackage
    Dim SubPackageElement As GOElement

    Set thePackageElement = thePackage

    '-----------------------------------------------------
    ' Alle Klassen des Packages durchgehen
    Set ClassCollection = thePackage.GetClasses()

    Dim Title, text As String
    Dim Response

    Title = "Klassen im Package >" + thePackageElement.Name + "<"
    text = ""
    For l = 0 To ClassCollection.Count - 1
        Set theClassElement = ClassCollection.GetAt(l)
        text = text + ">" + theClassElement.Name + "<" + Chr(13)

        Set theClass = theClassElement
```

```
        Set SuperclassCollection = theClass.GetSuperclasses()
        For Superclass_nr = 0 To SuperclassCollection.Count - 1
            Set theSuperclassElement = SuperclassCollection.GetAt(Superclass_nr)
            text = text + "     :" + theSuperclassElement.Name + Chr(13)
        Next Superclass_nr

        Set AttributeCollection = theClass.GetAttributes()
        For Attribute_nr = 0 To AttributeCollection.Count - 1
            Set theAttributeElement = AttributeCollection.GetAt(Attribute_nr)
            Set theAttribute = theAttributeElement
            text = text + "      ." + theAttributeElement.Name + ":" + theAttribute.Type + "
initvalue='" + theAttribute.InitValue + "'" + Chr(13)

            ' Werte verändern
            theAttribute.Type = "Integer"
            theAttribute.InitValue = "555"

        Next Attribute_nr

        ' ----
        'Attribute hinzufügen und löschen
         Set neuesAttribut = theClass.AddAttribute("TestAttribut", "Boolean", "False")
         MsgBox ("Attribut hinzugefügt")

         ret = theClass.DeleteAttribute(neuesAttribut)
         If ret Then
             MsgBox ("Delete-OK")
         Else
             MsgBox ("Delete fehlgeschlagen")
         End If
        ' ----

        Set OperationCollection = theClass.GetOperations()
        For Operation_nr = 0 To OperationCollection.Count - 1
            Set theOperationElement = OperationCollection.GetAt(Operation_nr)
            Set theOperation = theOperationElement
            text = text + "      ." + theOperationElement.Name + "("

            Set ParameterCollection = theOperation.GetParameters()
            For Parameter_nr = 0 To ParameterCollection.Count - 1
                Set theParameterElement = ParameterCollection.GetAt(Parameter_nr)
                Set theParameter = theParameterElement

                If Parameter_nr > 0 Then
                    text = text + ", "
                End If

                text = text + theParameterElement.Name + ":" + theParameter.Type

                ' Type ändern
                ' theParameter.Type = "Boolean"
            Next Parameter_nr

            ' Parameter hinzufügen
            Set dummy = theOperation.AddParameter("NeuerParam", "String", "Hallo", 1)

            ' Alle Parameter löschen
            theOperation.RemoveAllParameters

            text = text + ") : " + theOperation.ReturnType + Chr(13)

            ' Return-Type ändern
            theOperation.ReturnType = "Void"
        Next Operation_nr

        Set AssociationCollection = theClass.GetAssociations()
        For Association_nr = 0 To AssociationCollection.Count - 1
            Set theAssociation = AssociationCollection.GetAt(Association_nr)
            Set Role1 = theAssociation.GetRole1()

            ' Die beiden folgenden Zeilen liefern alternativ jeweils die andere Rolle
            'Set Role2 = theAssociation.GetRole2()
            Set Role2 = Role1.GetOtherRole()
```

```
            Set Role1Element = Role1
            Set Role2Element = Role2

            Dim nav1 As String
            Dim nav2 As String

            If Role1.Navigable Then
                nav1 = " ;isNav"
            End If

            If Role2.Navigable Then
                nav2 = " ;isNav"
            End If

            Dim Class1 As GOElement
            Set Class1 = Role1.GetClass()
            If (Class1.Name = theClassElement.Name) Then
                text = text + "      " + Role1Element.Name + " (" + Role1.Cardinality + nav1 + ")
<-->" + Role2Element.Name + " (" + Role2.Cardinality + nav2 + ")" + Chr(13)
            Else
                text = text + "      " + Role2Element.Name + " (" + Role2.Cardinality + nav2 + ")
<-->" + Role1Element.Name + " (" + Role1.Cardinality + nav1 + ")" + Chr(13)
            End If

            ' Werte setzen
            Role1.Cardinality = "5"
            Role1.Navigable = True

        Next Association_nr

        text = text + Chr(13)

    Next l

    If (text <> "") Then
        Response = MsgBox(text, , Title)
    End If

    '-----------------------------------------------------
    ' Alle Unter-Packages durchgehen
    Set SubPackageCollection = thePackage.GetPackages()

    Title = "Unter-Packages im Package >" + thePackageElement.Name + "<"
    text = ""
    For i = 0 To SubPackageCollection.Count - 1
        Set SubPackage = SubPackageCollection.GetAt(i)
        Set SubPackageElement = SubPackage
        text = text + ">" + SubPackageElement.Name + "<" + Chr(13)
    Next i

    If (text <> "") Then
        Response = MsgBox(text, , Title)
    End If

    '-----------------------------------------------------
    ' Rekursiver Abstieg für alle Unter-Packages
    For i = 0 To SubPackageCollection.Count - 1
        Call TraversierePackages(SubPackageCollection.GetAt(i))
    Next i
End Sub
```

Anhang F – Abbildungsverzeichnis

Abb. 1-1: Anwendungsoberfläche von GO (***Generating Objects***) 1
Abb. 1-2: Kommunikation zwischen GO und JANUS-Specifier 2

Abb. 3.1-1: Beispiel einer COM-Komponente mit zwei Schnittstellen 6
Abb. 3.1-2: Die Schnittstelle IUnknown 8
Abb. 3.1-3: Registrierungseditor – Eintrag der Komponente GOApplication 12
Abb. 3.1-4: Prozessinterner Server /Westhoff 00/ 14
Abb. 3.1-5: Lokaler Server /Westhoff 00/ 15
Abb. 3.1-6: Entfernter Server /Westhoff 00/ 15
Abb. 3.2-1: Beispiel für COM_INTERFACE_ENTRY2 24
Abb. 3.2-2: Vererbungsdiagramm COM-Komponente 27

Abb. 4.2-1: Lineare Schichtentrennung 32
Abb. 4.2-2: UML-Modell der relevanten M-Klassen 33
Abb. 4.2-3: COM-Komponente CGOClass mit Schnittstelle und Fachkonzeptklasse 33
Abb. 4.3-1: OOA-Diagramm einer COM-Komponente 49
Abb. 4.3-2: Vereinfachtes OOA-Diagramm einer COM-Komponente 50
Abb. 4.5-1: Hauptfenster des ***Dummy-Specifiers*** 57
Abb. 4.5-2: Traversier-Dialog des ***Dummy-Specifiers*** 58

Abb. C-1: UML-Klassendiagramm – Gesamtübersicht (Vererbungshierarchie) 70
Abb. C-2: UML-Klassendiagramm – GOApplication 71
Abb. C-3: UML-Klassendiagramm – GOCollection 72
Abb. C-4: UML-Klassendiagramm – GOElement 73
Abb. C-5: UML-Klassendiagramm – GOPackage 74
Abb. C-6: UML-Klassendiagramm – GOModel 75
Abb. C-7: UML-Klassendiagramm – GOClass 76
Abb. C-8: UML-Klassendiagramm – GOAttribute 77
Abb. C-9: UML-Klassendiagramm – GOOperation 78
Abb. C-10: UML-Klassendiagramm – GOParameter 79
Abb. C-11: UML-Klassendiagramm – GOAssociation 80
Abb. C-12: UML-Klassendiagramm – GORole 81
Abb. C-13: UML-Sequenzdiagramm – Initialisierung des Schnittstellenmodells (1. Teil) 82
Abb. C-14: UML-Sequenzdiagramm – Initialisierung des Schnittstellenmodells (2. Teil) 83

Anhang G – Tabellenverzeichnis

Tabelle 3.1-1: HRESULT-Werte und ihre Bedeutung 10
Tabelle 3.1-2: Automatisierungskompatible Datentypen.....17
Tabelle 4.2-1: COM-Komponenten und ihre zugehörigen Schnittstellen in GO.....33
Tabelle 4.5-1: LOC nach Dateien.....59
Tabelle 4.5-2: Arbeitsaufwand nach Tätigkeiten.....60

Anhang H – Literaturverzeichnis

/Balzert 96/
Balzert H., ***Lehrbuch der Software-Technik, Band 1,*** Heidelberg: Spektrum Akademischer Verlag 1996

/Balzert 99/
Balzert H., ***Komponentenbasierte Software-Entwicklung,*** Vorlesungsunterlagen zur Vorlesung Ingenieur-Informatik, Lehrstuhl für Software-Technik, Ruhr-Universität-Bochum 1999

/Balzert 00/
Balzert H., ***Lehrbuch der Software-Technik, Band 1, 2. Auflage,*** Heidelberg: Spektrum Akademischer Verlag 2000

/Gamma et al. 95/
Gamma E., Helm R., Johnson R., Vlissides J., ***Design Patterns – Elements of Reusable Object-Oriented Software***, Reading: Addison-Wesley Publishing Company 1995

/Kruglinski, Wingo, Sheperd 98/
Kruglinski D., Wingo S., Sheperd G., ***Inside Visual C++ 6.0***, Microsoft Press Deutschland 1998

/MSDN 00/
Microsoft, ***Dr. GUI über Komponenten, COM und ATL,*** in der MSDN Library July 2000

/OMG 99/
Object Management Group – UML Revision Task Force, ***UML 1.3***, www.omg.org 1999

/Rector, Sells 00/
Rector B., Sells C., ***ATL Internals, 4. Auflage***, Massachusetts: Addison Wesley Longman Inc. 2000

/Westhoff 00/
André Westhoff, ***Eine DTA-Komponente als COM-Server,*** Studienarbeit 03/00 am Lehrstuhl für Software-Technik, Ruhr-Universität Bochum 2000