D I P L O M A R B E I T

von
Jan Henning Arph

zum Thema

Implementierung eines Objektmodells zur Modellierung verteilter Funktionskomponenten für Netzwerke in Stationsleitsystemen nach IEC 61850

Betreuer:	Prof. Dr.-Ing. Georg Harnischmacher Dipl.-Ing. Kai Luppa
Tag der Einreichung:	13. Oktober 2003

Erklärung gem. § 26 Abs. 1 ADPO

„Hiermit erkläre ich, dass die Diplomarbeit von mir selbstständig verfasst wurde und nur die angegebenen Quellen und Hilfsmittel benutzt wurden."

_____

(Jan Henning Arph)

Dortmund, den 13. Oktober 2003

Inhalt

1 Einleitung

Im Umfeld der Installation, dem Betrieb, sowie der Wartung von Anlagenkomponenten der elektrischen Energieversorgungsnetze, kommt PC-basierten Bedien- und Konfigurationswerkzeugen eine entscheidende Rolle im Bezug auf den wirtschaftlichen und sicheren Betrieb einer Anlage zu.
Schon während der Projektierungsphase einer Anlage liefern graphische und intuitiv zu bedienende Werkzeuge einen entscheidenden Beitrag für die ökonomische Umsetzung der Aufgabe, indem sie eine schnelle, sichere und fehlerfreie Projektierung der Anlage unterstützen.
Zudem wird es durch einfach zu bedienende Werkzeuge möglich, die Investitionen für die Einarbeitung von Anwendern, die ein entsprechendes Tool benutzen sollen, gering zu halten. Der Anwender wird nicht durch komplizierte Bedienungsmethoden des Werkzeugs von seinen eigentlichen Aufgaben abgelenkt. Das Werkzeug soll ihn bei der Umsetzung der Aufgabe, die im Umfeld seiner Kernkompetenz liegen, unterstützen.
Damit Projektierungstools im Anwendungsbereich Akzeptanz erlangen, ist es notwendig, dass die Modelle und Konfigurationen, die mit einem solchen Werkzeug erstellt werden, nicht auf proprietären Strukturen aufsetzten, sondern den Regeln allgemeingültiger Standards zu entsprechen.
Für den Bereich der Stationsleittechnik befindet sich die neue Normenreihe IEC 61850 "Communication networks and systems in substations" in der Einführung. Diese Norm stellt einen Standard bereit, der die Interoperabilität, d.h. die Zusammenarbeit zwischen einzelnen Komponenten der Unterstation gewährleistet.
Der praktische Teil der vorliegenden Arbeit implementiert ein graphisches Werkzeug zur Konfiguration von Unterstationen der elektrischen Energienetze auf Grundlage der Normenreihe IEC 61850.
Die Anwendung mit dem Titel Visual SOM Studio ist als PC-Applikation auf der .NET-Plattform entwickelt.

2 Die Normenreihe IEC 61850

Durch die Deregulierung des Strommarktes in den vergangenen Jahren sind die Wettbewerbsanforderungen für die einzelnen Netzbetreiber stark gestiegen. Um im deregulierten Mark wettbewerbsfähig zu bleiben, ist es erklärtes Ziel der Netzbetreiber den Betrieb der elektrischen Energienetze ökonomisch zu gestalten. Eine sehr kostenintensive Situation mit hohem Einsparungspotential stellt die Stationsleittechnik dar. Häufig existieren in der Stationsleittechnik eine Vielzahl unterschiedlicher Systeme unterschiedlicher Hersteller nebeneinander, die zum größten Teil alle, auf Grund der Ermangelung eines allgemeingültigen Kommunikationsstandards, über proprietäre Protokolle kommunizieren. Dies bedeutet, dass die Austauschbarkeit einzelner Komponenten eines bestehenden Systems, durch eventuell kostengünstigere in Wartung und Betrieb oder aber durch zweckmäßigere Komponenten, ökonomisch betrachtet nicht möglich ist, da die Integrationsinvestitionen für die neuen Komponenten unverhältnismäßig hoch ausfallen.
Durch diese Situation wird die Anpassungsfähigkeit einer Anlage an die zu erfüllende Aufgabe stark eingeschränkt.
Zur Beseitigung dieser Situation wurde die Norm IEC 61850 "Communication networks and systems in substations" eingeführt. Die Norm standardisiert die Kommunikation, d.h. den Informationsaustausch zwischen einzelnen Komponenten innerhalb eines Schaltanlagensystems.
Durch die Sicherstellung der Zusammenarbeit (interoperability) zwischen einzelnen Komponenten durch die Norm, kann der Markt an Geräten und Funktionen für die Stationsleittechnik expandieren. Dadurch wird gleichzeitig eine größere Differenzierung der angebotenen Funktionalitäten stattfinden, womit die Effizienz des Netzbetriebes und dessen Wartung, sowie das Investitionsvolumen optimiert werden.
Das Ziel der Norm IEC 61850 ist es, die Interoperabilität zwischen unterschiedlichen Geräten zu gewähren, ohne dabei jedoch die einzelnen Algorithmen der Funktionsimplementierung, bzw. die zur Kommunikation eingesetzten Systeme zu fixieren.
Zu diesem Zweck benennt die Norm im Teil 7 alle logischen Funktionen (Logical Node LN), die im Schaltanlagensystem Verwendung finden. Dabei wird durch die Norm nicht die Implementierung der Funktion, also deren Verhalten definiert, sondern lediglich deren Datenschnittstellen. Durch die Datenschnittstelle einer Funktion wird festgelegt, welche Informationen die Funktion für ihre Aufgabe benötig bzw. welche Informationen die Funktion für andere am System beteiligte Funktionen bereitstellen kann.
Dadurch, dass die Norm die Datenschnittstellen der einzelnen Funktionseinheiten einer Schaltanlage festlegt, zerlegt die Norm gleichzeitig die einzelnen Funktionseinheiten der Schaltanlage in kleine logische Funktionen. Somit wird eine flexible Konfiguration von stationsleittechnischen System möglich, die es dem Konfigurator erlaubt die einzelnen Funktionen nahezu an beliebiger Stelle im System zu platzieren. Damit trägt die Norm den unterschiedlichen Philosophien der Anlagenkonfiguration der unterschiedlichen Betreiber Rechnung.

Weiterhin hat die Norm die Aufgabe, die Langzeitstabilität des Betriebes eines stationsleittechnischen Systems zu gewährleisten, wobei das System dabei die Möglichkeit besitzen muss, an zukünftigen Technologien partizipieren zu können. Im Bezug auf die sich rasant fortschreitenden Entwicklung der Kommunikationstechnologien ist die Norm so ausgelegt, dass die in einem System eingesetzte Kommunikationstechnik nicht durch die Norm festgelegt wird. Dadurch bleibt die für ein Projekt einzusetzende Kommunikation frei wählbar und lässt sich an die jeweiligen projektspezifischen Bedingungen anpassen. Ebenso lässt sich die komplette Kommunikationsstruktur eines bestehenden Systems austauschen, ohne die Anwendungsschichten des Systems verändern zu müssen.
Um der Langzeitstabilität der System gerecht zu werden, trennt die Norm IEC 61850 Daten, logischen Funktionen und die Kommunikation konsequent von einander. Die

Daten und die logischen Funktionen werden, im Gegensatz zu der Kommunikation durch die Norm beschrieben. Um also in der praktischen Anwendung Informationen über ein Kommunikationssystem austauschen zu können, sieht die Norm eine einheitliche Abbildung (Mapping) auf ein beliebiges, den Anforderungen der Kommunikation zwischen den Funktionen gerecht werdendem System vor. Die Norm IEC 61850 spezifiziert im Teil 8-1 ein Standardmapping auf die Protokolle MMS und ISO/IEC 8802-3.

In den Teilen 7-3 und 7-4 werden die einzelnen logischen Funktionen, die innerhalb einer Schaltanlage eingesetzt werden, aufgelistet, sowie deren Datenschnittstellen in Form von Datenobjekten definiert. Zudem wird mit Hilfe eines hierarchischen Objektbaums die Struktur definiert, über die auf einzelne Funktionen innerhalb der Schaltanlage auf standardisierte Weise zugegriffen werden kann.
Damit jedoch einzelne Funktionen ihre benötigte Daten von anderen Funktionen des System beziehen können, müssen die physikalischen Geräte in denen die jeweiligen Funktionen implementiert sind, darüber informiert sein, wo und über welchen Kommunikationsweg die benötigten Daten bezogen werden können. Die einzelnen Funktionen müssen also über die Konfiguration ihres Umfeldes informiert sein. Um diese Konfiguration einheitlich festlegen zu können, stellt die Norm IEC 61850 den Teil 6 bereit.
Der Teil 6 der Norm spezifiziert eine auf XML basierende Syntax zur Konfiguration stationsleittechnischer Systeme. Diese durch die Norm festgelegte Konfigurationsmethode wird als SCL (Substation Configuration Language) bezeichnet.

Die SCL ist dabei in drei unterschiedliche Bereiche aufgeteilt, dies wird durch die unterschiedliche Farbgebung der einzelnen Objekte des Objektmodells der SCL im Modellkonfigurator der Norm bereits deutlich.

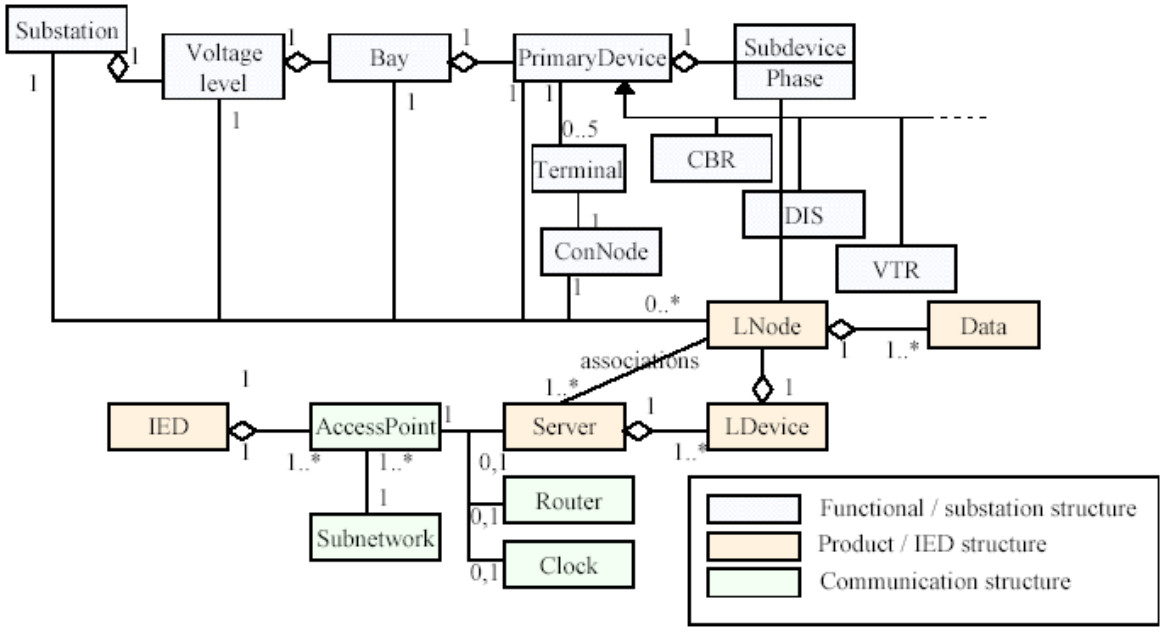

Abbildung 1: SCL Objektmodell

Mit dem Stationsmodell wird auf Grundlage einer hierarchischen Struktur die gesamte Primärtechnik der Anlage abgebildet. Somit wird mit dem Stationsmodell die Verbindung zwischen realer Schaltanlage und deren Abbildung im computertechnischen System der Leittechnik hergestellt. Die gesamte Anlage wird mit dem Wurzelelement "Substation" beschrieben. Dieses Element enthält als weiteres Element "Voltage level", das jeweils eine Spannungsebene repräsentiert. Jede Spannungsebene kann mehrer Felder (Bay) und diese wiederum mehrer Geräte (Primary Device) enthalten. Mit Hilfe der Objekte

ConNode wird die elektrische Verbindung dargestellt, an die die einzelnen Geräte angeschlossen sind. Über die Subdevice-Objekten wird die Verbindung zwischen dem stationsbeschreibenden Stationsmodell und dem gerätebeschreibenden Produkt-(IED) Modell hergestellt.
Mit dem Produktmodell der SCL werden die logischen Knoten (LN) des Teils 7-4 der Norm in einen Gerätekontext eingefügt. Ebenfalls mit Hilfe einer hierarchischen Struktur, angelehnt an die des Teils 7 der Norm, werden mit dem Produktmodell die funktionalen Fähigkeiten eines Gerätes definiert.
An oberster Stelle des Produktmodells steht das IED-Objekt. Jedes IED enthält ein Server-Objekt, das innerhalb des LDevice-Objektes die einzelnen logischen Funktionen, repräsentiert durch die LNode-Objekte des Gerätes, enthält. Die einzelnen Daten einer Funktion, definiert im Teil 7-3, werden schließlich durch das Data-Objekt dargestellt.

Mit dem Kommunikationsmodell, das im Gegensatz zu dem Stations- und dem Produktmodell nicht hierarchisch angeordnet ist, werden die logischen Verbindungen zwischen verschiedenen IED's der zu konfigurierenden Schaltanlage definiert.
Jedem IED wird mit dem Kommunikationsmodell ein AccessPoint-Objekt zugeteilt, das sich in einem bestimmten Netzwerk, dargestellt durch das Subnetwork-Objekt, befindet. Somit können alle IED's, die sich im gleichen Subnetwork befinden, miteinander kommunizieren. Mit Hilfe des Kommunikationsmodells werden somit die verschiedenen logischen Funktionen über ihre jeweiligen Geräte (IED) miteinander verbunden.

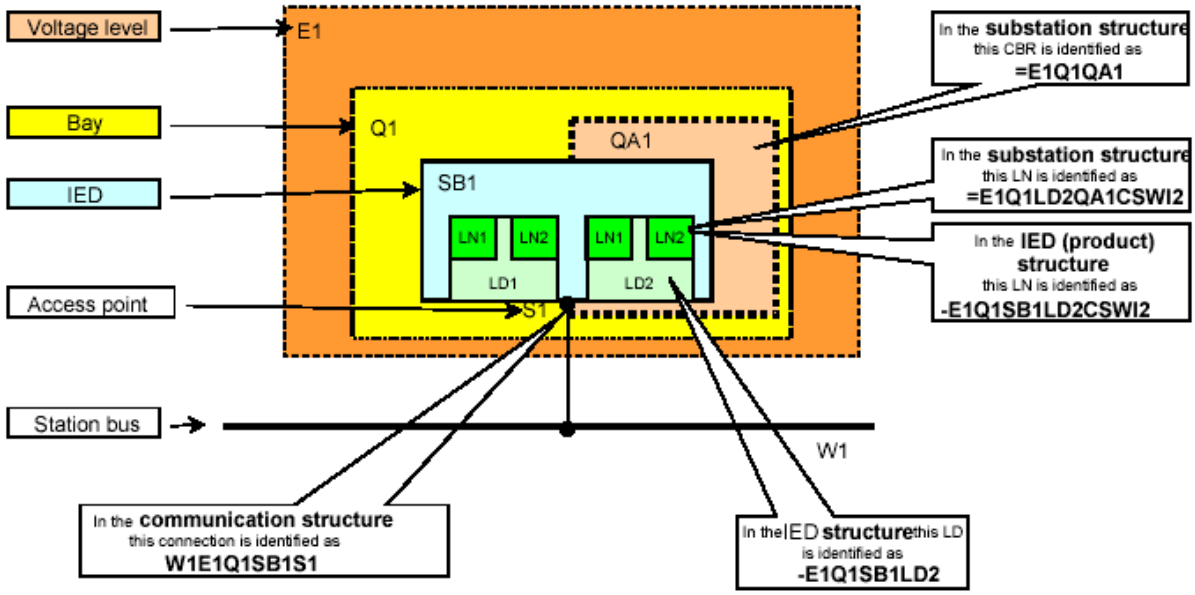

Abbildung 2: Schematische Darstellung einer Station mit den unterschiedlichen Strukturen des SCL-Modells

Unter der Verwendung aller drei Modelle kann somit mit der SCL eine vollständige Anlagenkonfiguration gemäß Abbildung 2 beschrieben werden.

3 Das Substation Object Model SOM

Ein genanntes Ziel der Normenreihe IEC 61850 ist es, die Interoperlabilität zwischen verschiedene Geräten der Stationsleittechnik zu gewährleisten. Zu diesem Zweck sind im Teil 7 der Normenreihe IEC 61850 die Grundlagen für ein Objektmodell definiert, mit dem Komponenten und Funktionen der Stationsleittechnik abgebildet werden können. In der Norm wird davon abgesehen die Implementierung eines solchen Objektmodells, sowie dessen Zielsysteme zu spezifizieren, damit die Norm unabhängig von Plattformen und Kommunikationstechniken bleibt. Durch diese Flexibilität wird die Norm an zukünftige Techniken und Methoden anpassbar gehalten.
Um eine Station basierend auf dem Teil 7 der Norm IEC 61850 in einem DV-System abzubilden, ist es somit notwendig eine Implementierung zu konkretisieren. Das heißt, das auf einem spezielles System mit einer konkreten Kommunikationstechnik das Objektmodell einer Station nach IEC 61850 Teil 7 realisiert wird.
Zu diesem Zweck wurde das Substation Objekt Model (SOM) entwickelt. Das SOM bildet den Teil 7 der Normenreihe IEC 61850 auf der .NET-Plattform der Firma Microsoft ab. Die Kommunikation zwischen einzelnen logischen Funktionen der abgebildeten Station wird dabei über die im .NET-Framework integrierten Kommunikationssysteme abgewickelt. Das .NET-Framework bedient sich bei der Kommunikation in verteilten Anwendungsszenarien, wie sie etwa die Stationsleittechnik fordert, üblicher und in der IT-Welt weitverbreiteter Technologien, wie z.B. SOAP und TCP.
Das SOM definiert Softwareschnittstellen unter deren Verwendung die objekthierarchische Struktur des Teils 7 der Norm IEC 61850 abgebildet werden kann. Unter Verwendung der Schnittstellen (engl. Interface) werden dynamische Container implementiert mit deren Hilfe die Hierarchie des Modells festgelegt wird und gleichzeitig die Flexibilität der Modellierung von Unterstationen nach IEC 61850 Teil 7 gewährleistet bleibt.

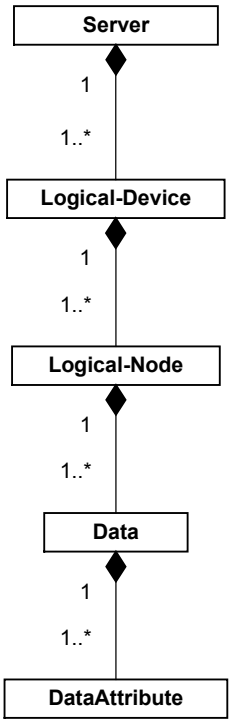

Abbildung 3: Die durch den Teil 7 der Norm IEC 61850 definierte Objekthierarchie

Die hierarchische Struktur, die im Teil 7 der Norm IEC 61850 festgelegt ist, wird im SOM durch die Schnittstellen IServer, ILogicalDevice, ILogicalNode, IDataObject und IDataAttribut erzeugt, wobei das SOM weitere Schnittstellen ILogicalDevices, ILogicalNodes und IDataObjects für die Implementierung der dynamischen Container bereitstellt.

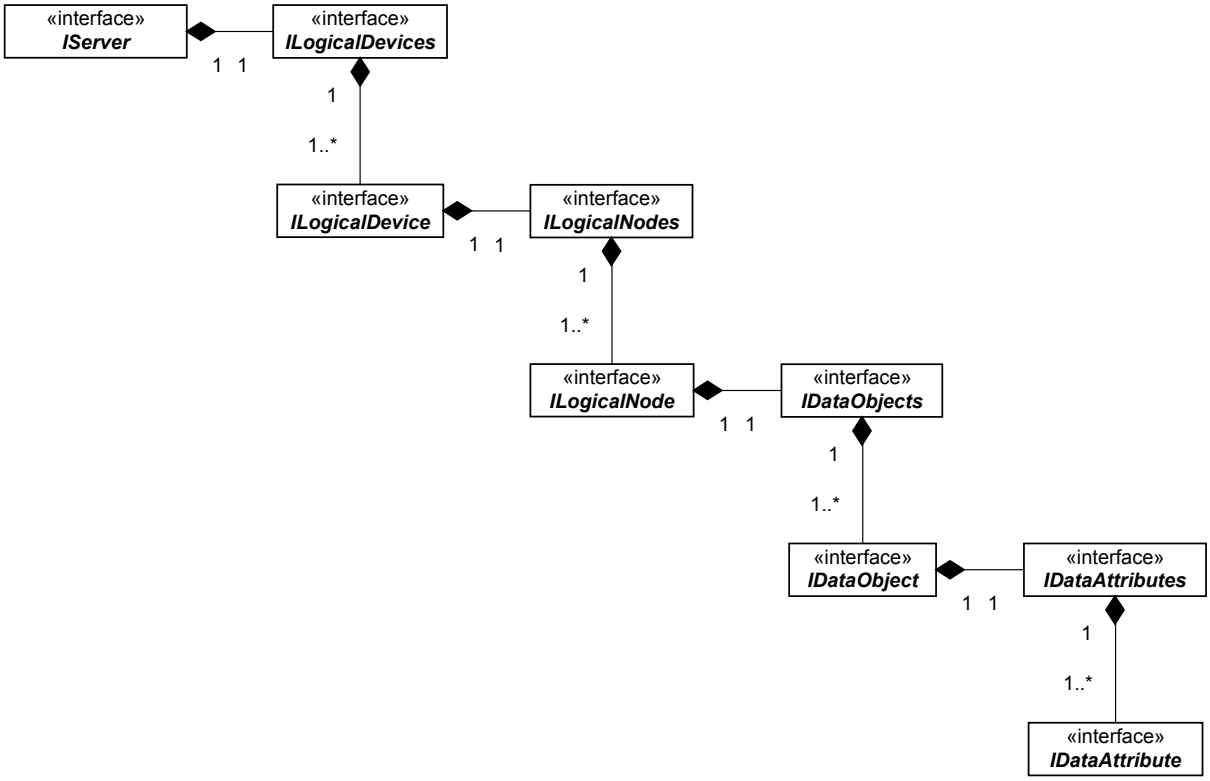

Abbildung 4: Das SOM-Schnittstellenmodell mit dynamischen Containern

Der Begriff der Schnittstelle, dessen Methodik im SOM eingesetzt wird, darf in diesem Kontext nicht verwechselt werden mit konkreten physikalischen Geräteschnittstellen, wie etwa Klemmleisten oder Stecker.
Eine Softwareschnittelle ist eine Beschreibung von Attributen, Methoden und Ereignissen. Der Unterschied zu einer Klasse besteht darin, dass die Methoden jedoch keine Implementierungen enthalten. Eine Schnittstelle entspricht somit einer abstrakten Klasse. Somit kann eine Schnittstelle nicht instanziert werden, sondern nur im Rahmen einer Klasse verwendet werden. Dort muss die Schnittstelle implementiert werden. Genauer gesagt, dort müssen alle Attribute und Ereignisse deklariert, sowie alle Methoden deklariert und gegebenenfalls implementiert werden **[6]**.

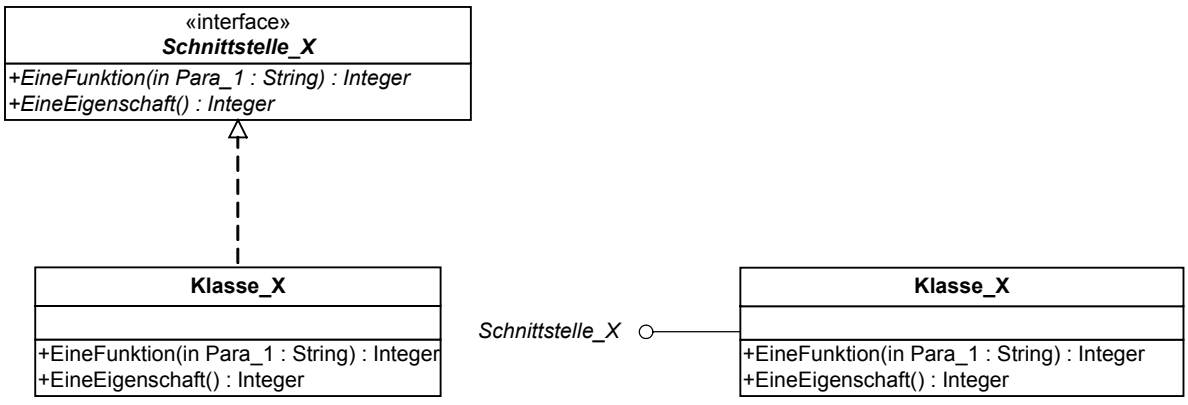

Abbildung 5: Zwei Varianten zur Darstellung von Schnittstellen

Im nachfolgenden Codebeispiel wird zunächst eine Schnittstelle implementiert, die eine Funktion und eine Eigenschaft enthält.

```
Public Interface Schnittstelle_X

    Function EineFunktion(ByVal Para_1 As String) As Integer
    ReadOnly Property EineEigenschaft() As Integer

End Interface
```

Dadurch, dass die Klasse `Klasse_X` die Schnittstelle `Schnittstelle_X` implementiert, muss die Klasse `Klasse_X` alle Methode der Schnittstelle `Schnittstelle_X` auch implementieren.

```
Public Class Klasse_X
    Implements Schnittstelle_X

    Public Function EineFunktion(ByVal Para_1 As String) As Integer
                  Implements Schnittstelle_X.EineFunktion

    End Function

    Public ReadOnly Property EineEigenschaft() As Integer Implements
                        Schnittstelle_X.EineEigenschaft
        Get

        End Get
    End Property
End Class
```

Dadurch, dass eine Klasse eine Schnittstelle implementiert, versichert diese Klasse alle Methoden, Attribute und Ereignisse, die durch die Schnittstelle festgelegt sind, zu implementieren.

Das auf Schnittstellen basierende SOM bietet somit ein abstraktes Objektmodell. Das SOM stellt keine konkrete Klassen, die etwa logischen Knoten oder Datenobjekten aus dem Teil 7 der Norm IEC 61850 implementieren, bereit. Somit wird durch das SOM eine Implementierungsvorgabe für konkrete Klassen gegeben, die zusammen ein Objektmodell bilden, das eine Unterstation normgerecht nach IEC 61850 Teil 7 in einem Computersystem abbilden.

4 Kommunikation

4.1 Kommunikation in der Stationsleittechnik

Der Kommunikation innerhalb eines Datenverarbeitungssystems fällt eine bedeutende Aufgabe zu. Die Kommunikation ermöglicht es Daten innerhalb eines Systems für alle beteiligten Komponenten verfügbar zu machen. Besonders in verteilten Systemen, wie etwa der Netzleittechnik, ist eine einwandfrei funktionierende Kommunikation Grundvoraussetzung für den sicheren und stabilen Betrieb des Systems. Fällt die Kommunikation zwischen einzelnen Systemkomponenten aus, stehen eventuell wichtige Daten nicht zur Verfügung. Neben der Kommunikationssicherheit und der Sicherheit von Übertragungswegen sollte ein Kommunikationssystem innerhalb von verteilten Anwendungen standardisiert sein. Damit wird die Wartbarkeit des Systems erhöht. Einzelne Komponenten des Systems können dadurch ohne größere Aufwand ersetzt werden. Gleichzeitig kann so das System an zukünftige Aufgaben leichter angepasst werden.
In der konventionellen Stationsleittechnik erfolgt die Kommunikation auf elektrischem Wege über physikalische Größen. Der Informationsaustausch erfolgt über standardisierte analoge Strom- und Spannungsschnittstellen, so dass der Kommunikationsweg zwischen Geräte über einfaches Verdrahten erfolgt. Durch den Einsatz der digitalen Stationsleittechnik werden die analogen Schnittstellen durch serielle Kommunikationsschnittstellen ersetzt. Um die Flexibilität des Systems zu erhalten, ist es notwendig den seriellen Datenstrom zwischen den Gräten zu standardisieren. Zu diesem Zweck entstanden Normenreihen, wie etwa IEC 60870-5. Diese Normen spezifizieren eine eindeutige Protokollsyntax für den Datenaustausch zwischen Komponenten. Bei diesen Protokollen wird die Semantik der Daten durch ihre Position innerhalb eines Protokolls definiert. Dies macht es notwendig, dass die Spezifizierung der Kommunikation bis auf die Ebene eines einzelnen Bits herunterreicht. Damit Systeme an projekt- oder betriebsspezifische Anforderungen angepasst werden können, sind ergänzend zu den Grundnormenreihen anwendungsspezifische Protokollfestlegungen notwendig. Beispiele dafür sind das Fernwirkprofil IEC 60870-5-101 oder die Informationsschnittstelle des Schutzes IEC 60870-5-103. Innerhalb dieser Profile können hersteller- oder projektspezifische Anpassungen innerhalb vorgegebener Freiräume vorgenommen werden. Allgemeingültige Regelwerke für die Semantik der in den projektspezifischen Bereichen abgebildeten Daten sind jedoch durch die Norm nicht festgelegt. Damit bleibt, selbst durch die Möglichkeit von Profilfestlegungen, die inhaltliche Auszeichnung und damit die Grundlage der Erstellungen eines Datenmodels, wie beispielsweise die Adressierung einer Leistungsschalter-Störmeldung, projekt- oder herstellerspezifisch.

Um ein genormtes Rahmenwerk für die inhaltliche Bewertung einer Nachricht zwischen verschiedenen Geräten der Stationsleittechnik zu definieren, wurde die Normenreihe IEC 61850 entwickelt. Grundlage der Normenreihe IEC 61850 ist es, ein grundlegendes Daten- und Dienstmodell für ein Kommunikationssystem, das eine Interoperabilität zwischen Funktionen und Geräten verschiedener Hersteller gewährleistet, bereitzustellen. Um dies zu erreichen beschreibt die Normenreihe IEC 61850 in ihrem Teil 7 eine protokollunabhängige, abstrakte Darstellungsform für die Datenmodellierung. Dies bedeutet jedoch, dass ein Datenmodell, das mit den Regeln von IEC 61850-7 konform geht, zur realen Anwendung gelangen soll, auf einen bestimmten Protokollstack gemappt werden muss.
Die zur Kommunikation eingesetzte Technologie bleibt damit unabhängig von der Normung. Die Norm beschreibt lediglich Funktionen und Dienste oberhalb der Anwendungsschicht des ISO/OSI-Modells. Alle anderen unterlagerten, für die Kommunikation benötigten Schichten können ausgetauscht werden, bei gleichzeitiger Wahrung der Norm. Durch die Austauschbarkeit der Kommunikationsstacks wird das System flexibler in Bezug auf den Einsatz zukünftiger, vielleicht effizienteren Kommunikationstechnologien.

Durch die Möglichkeit des Einsatzes unterschiedlicher Kommunikationsstacks können durchaus zukünftige Anwendungen, die auf der Normenreihe IEC 61850 basieren, also ein einheitliches Datenmodell darstellen, sich in der Abbildung auf einen Protokollstack voneinander unterscheiden.
Mit der Schnittstellenspezifikation SOM (Substation Object Model) kann ein auf IEC 61850-7 basierendes Datenmodel mit dynamischen Kardinalitäten mittels der .NET-Remotingtechnologie auf bestehende Protokollstacks der allgemeinen Informationstechnologie, wie etwa TCP, HTTP abgebildet werden.

4.2 Remoting per MBV und MBR

Mit Hilfe des .NET-Remoting ist es möglich, ein Objekt in einem Prozess A für Objekte in einem anderen Prozess B verfügbar zu machen, ohne den Entwickler mit der Komplexität der konkreten Verteilung und Verwaltung der einzelnen Objekte zu belasten. Ein Objekt über seine Anwendungsdomäne hinaus verfügbar zu machen, wird Marshalling genannt. Dabei muss zwischen zwei grundlegend verschiedenen Methoden des Marshallings unterschieden werden. Bei der Methode marshal by value (MBV) richtet der Client seine Anfrage an den Server, daraufhin wird eine exakte Kopie des Serverobjekts erstellt und an den Client zurück gesendet. Das Objekt wird jetzt im Prozessraum des Client ausgeführt. So kann der Client die Daten und Funktionalität des Objektes direkt in seinem Prozess verwenden, ohne weitere Anfragen an den Server richten zu müssen. Das Serverobjekt kann dabei jedoch nicht als gemeinsamer Datenpool für andere beteiligte Clients fungieren, da für jeden Client eine neue Kopie des Serverobjekts angelegt wird.

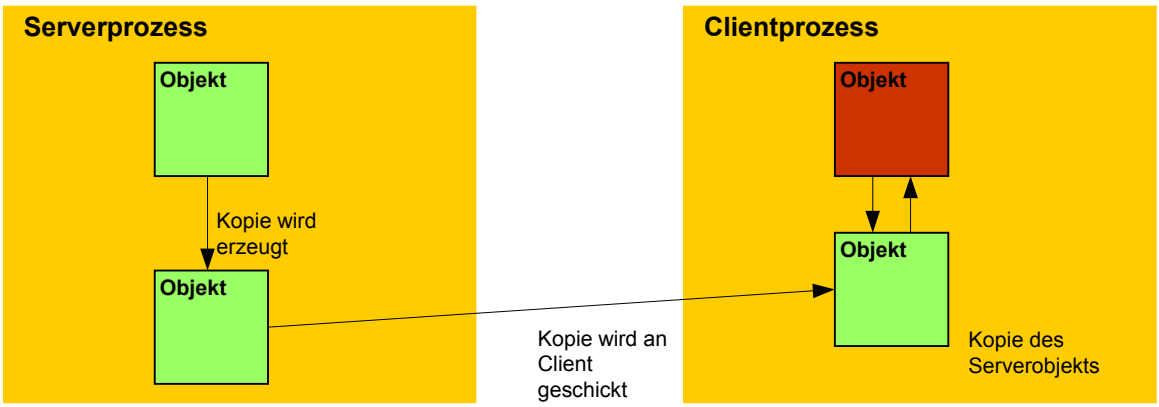

Abbildung 6: „marshal by value“ MBV

Um die Methode MBV in Verbindung mit dem .NET-Remoting einzusetzen, muss die Klasse eines zu marshallendem Objekts die Schnittstelle ISerializable implementieren, oder aber mit dem Attribute <Serializable> ausgestattet sein.

Im Gegensatz zu der oben dargestellten Methode MBV, wir bei der Methode marshal by reference (MBR) ein sogenanntes Proxyobjekt verwendet. Sobald ein Client auf ein MBR-Objekt zugreift, wird im Prozessraum des Client das Proxyobjekt erzeugt. Diese Stellvertreterobjekt verfügt über die exakt gleichen Schnittstellen, wie das zu vertretende Serverobjekt. Die vom Client ausgeführten Methodenaufrufe am Proxy werden direkt von diesem an das eigentliche Serverobjekt weitergereicht. Dabei kümmert sich das .NET-Remoting eigenständig um den Datenaustausch zwischen Objekt und dessen Proxy, so das der Zugriff auf das Serverobjekt auf Seiten des Clients über den Proxy transparent für den Entwickler geschieht. Der Client kann somit das entfernte Objekt wie ein lokales Objekt benutzen.

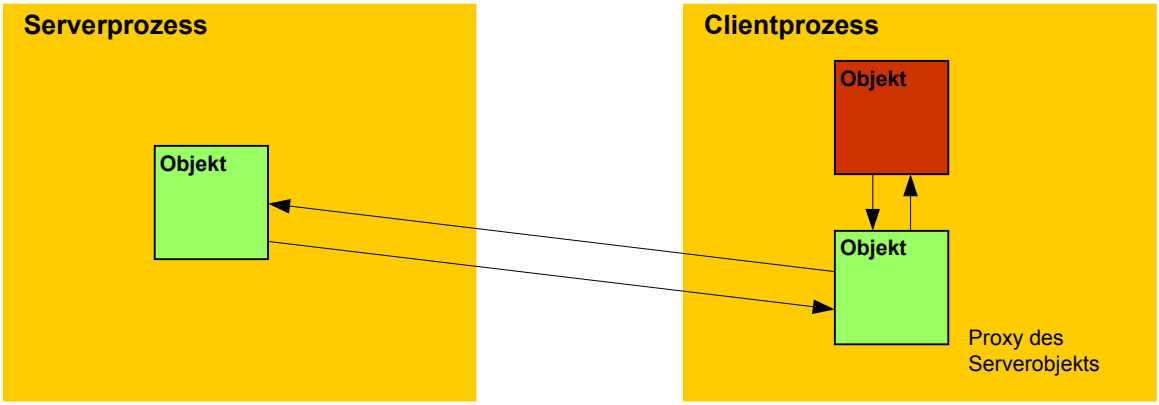

Abbildung 7: „marshal by reference" MBR

Um die Methode MBR mit dem .NET-Remotingframework umzusetzen, muss die Klasse eines zu marshallendem Objekts von der Klasse `MarshalByRefObject` abgeleitet werden.

Die beiden im folgenden dargestellten Anwendungen sollen die Methode MBV und MBR an Hand einer Parameterübergabe an eine Methode eines entferntes Objekt veranschaulichen.
Serverkomponente und Clientkomponenten sind in beiden Fällen, sowohl bei der Implementierung für MBV, als auch bei der für MBR identisch. Die ausführbare Serveranwendung hostet ein Objekt der Klasse `HostingClass` und registriert dieses im .NET-Remoting.

```
Public Class Server

Shared Sub Main()

    Dim chan As New HttpChannel(8085)
    ChannelServices.RegisterChannel(chan)
    RemotingConfiguration.RegisterWellKnownServiceType(
            GetType(Share.RemotingSamples.HostingClass), "SayHello",
            WellKnownObjectMode.SingleCall)

    System.Console.WriteLine("Hit <enter> to exit...")

    System.Console.ReadLine()

End Sub

End Class
```

Der Client instanziert ein lokales Proxyobjekt des vom Server gehosteten Objekts. Anschließend instanziert der Client eine neues Objekt der Klasse `ForwardMe` und ruft die Methode `CallMe` mit dem Parameter `"Regards from the client"` auf. Daraufhin ruft der Client die Methode `InvokeCallMe_OnParam` des Proxyobjekts auf und übergibt an diese das Objekt der Klasse `ForwardMe`.

```
Public Class Client

Shared Sub Main
    Dim chan As New Http.HttpChannel(8086)
    ChannelServices.RegisterChannel(chan)
    Dim obj As Share.RemotingSamples.HostingClass

    obj =  Type(Activator.GetObject(
                GetType(Share.RemotingSamples.HostingClass),
                "http://MyServer:8085/SayHello"), HostingClass)

    Dim param As New ForwardMe()
    param.CallMe("Regards from the client")
    obj.InvokeCallMe_OnParam(param)

    System.Console.ReadLine()
End Sub

End Class
```

4.2.1 Darstellung der Methode MBV

Die Komponente Share.dll enthält zwei Klassen. Mit Hilfe der Klasse `ForwardMe` soll zunächst das Verhalten eines MBV-Objekts deutlich gemacht werde. Dazu ist die Klasse als <serializable> gekennzeichnet und verfügt über die Methode `CallMe`. Diese Methode gibt den Wert des an sie übergebenen Parameters auf der Konsole aus. Somit lässt sich zur Laufzeit verdeutlichen, in welchem Prozessraum sich das Objekt gerade aufhält. Befindet sich das Objekt im Prozessraum des Clients wird die Ausgabe auf der Konsole des Clients erfolgen, wohingegen die Ausgabe auf der Konsole des Servers erfolgt, wenn sich das Objekt im Prozessraum des Servers befindet. Die Klasse `HostingClass` ist von der Klasse `MarshalByRefObject` abgeleitet. Die Instanz dieser Klasse wird in der Komponente server.exe gehostet. Die Klasse `HostingClass` verfügt über einen Konstruktor, der auf der Konsole anzeigt, das eine neue Instanz der Klasse erzeugt wurde. Die Klasse verfügt zudem über eine Methode `InvokeCallMe_OnParam`. Dieser wird als Parameter ein Objekt der Klasse `ForwardMe` übergeben. Die Methode ruft an dem übergebenen Objekt der Klasse `ForwardMe` die Methode `CallMe` auf um anzuzeigen, in welchem Prozessraum das Objekt der Klasse `ForwardMe` sich gerade befindet.

```
<Serializable()> Public Class ForwardMe
    Public Function CallMe(ByVal value As String) As Object
        Console.WriteLine(value)
    End Function
End Class

Public Class HostingClass
    Inherits MarshalByRefObject

    Public Sub New()
        Console.WriteLine(Me.GetType.Name & " activated")
```

```
    End Sub

    Public Sub InvokeCallMe_OnParam(ByVal obj As ForwardMe)
        obj.CallMe("Regards from the server")
    End Sub
End Class
```

Zunächst wird der Server gestartet. Der Server registriert die Klasse HostingClass im Remotingframework. Anschließend wird der Client gestartet. Der Client versucht ebenfalls eine Instanz der Klasse HostingClass anzulegen. Über die Methode GetObject des Activator-Objekts wird im Serverprozessraum ein neues Objekt der Klasse HostingClass erzeugt. Daraufhin erhält der Client einen Verweis auf das im Server gehostete Objekt, indem eine neue Proxyinstanz des Serverobjekts im Prozessraum des Clients erzeugt wird.
Der Client erhält einen Verweis auf das entfernte Objekt über einen Proxy, da die Klasse HostingClass des entfernten Objekts abgeleitet ist von `MarshalByRefObject`. Somit liegt zu diesem Zeitpunkt der Anwendungsausführung folgende Situation vor:

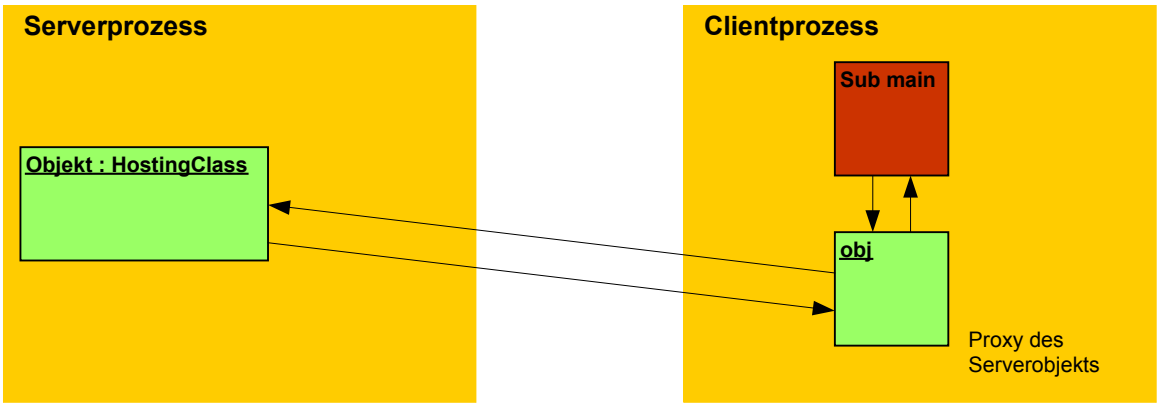

Abbildung 8: Objektinstanz und zugehörige Proxyinstanz

Jetzt erzeugt der Client eine neue Instanz der Klasse `ForwardMe` und ruft die Methode `CallMe` mit dem Parameter `"Regards from the client"` auf. Die Ausgabe des Textes erscheint im Ausgabefenster des Client, da sich das Objekt im Prozessraum des Clients befindet. Anschließend wird die Methode `InvokeCallMe_OnParam` des Proxy-Objektes aufgerufen und dieser als Parameter die Instanz der Klasse `ForwardMe` übergeben. Da es sich bei der Klasse `ForwardMe` um eine Klasse handelt, die mit dem Attribut <serializable> gekennzeichnet ist, wird bei diesem Methodenaufruf vom .NET-Remoting eine Kopie der Instanz angelegt und diese als Parameter an das Serverobjekt übergeben. Die Methode `InvokeCallMe_OnParam` ruft an dem an sie übergeben Objekt der Klasse `ForwardMe` die Methode `CallMe` mit dem Parameter `"Regards from the server"` auf. Da sich zu diesem Zeitpunkt die Kopie des Objekt der Klasse `ForwardMe` im Prozessraum des Server befindet, erscheint die Ausgabe des Textes im Ausgabefenster des Servers.

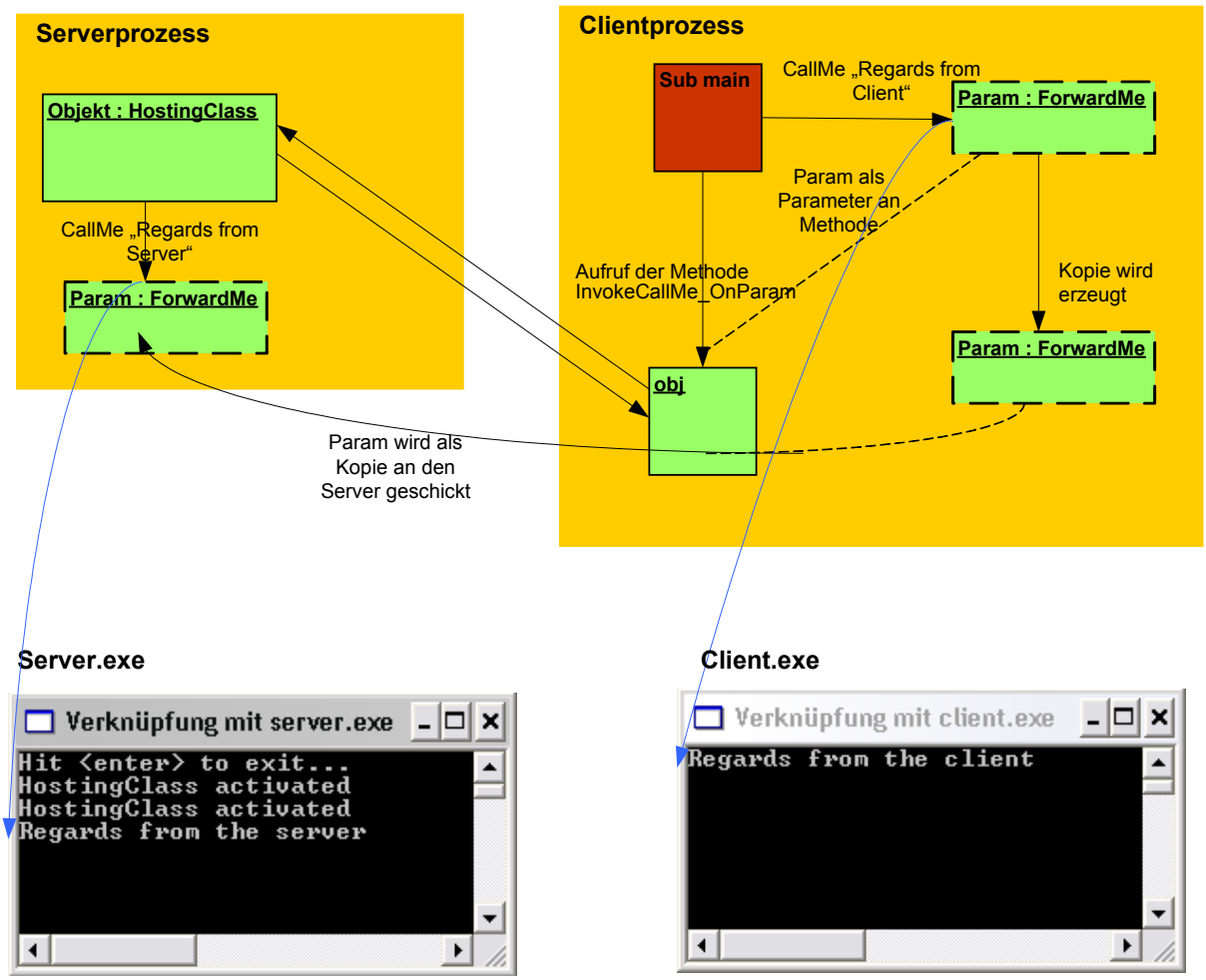

Abbildung 9: Das Objekt `Para` der Klasse `ForwardMe` als serialiserbarer Parameter

Da die Kommunikation zwischen Client und Server über http abgewickelt wird und das .NET-Remoting bei einer http-Kommunikation standardmäßig als Datenformat SOAP verwendet lässt sich der Aufruf der Methode `InvokeCallMe_OnParam` mit Hilfe eines Netzwrkmonitors **[4]** direkt nachvollziehen. Bei den dargestellten Protokollmitschnitten wurden übertragenen Daten von anderen Protokollen, wie etwa TCP und IP zum Zweck der besseren Übersicht entfernt. Ebenso wurde der SOAP-ENV:Envelope-Tag in verkürzter Schreibweise dargestellt und bestimmte Formatierungen vorgenommen.

Das Proxy-Objekt auf Clientseite reicht den Aufruf der Methode `InvokeCallMe_OnParam` an das Objekt im Server weiter. Dafür schickt es eine http-POST-Methode an den Server.

```
1 : POST /SayHello HTTP/1.1
2 : User-Agent: Mozilla/4.0+(compatible; MSIE 6.0; Windows 5.1.2600.0; MS .NET Remoting; MS .NET CLR
                                                                                    1.0.3705.288 )
3 : Content-Type: text/xml; charset="utf-8"
4 : SOAPAction:
"http://schemas.microsoft.com/clr/nsassem/Share.RemotingSamples.HostingClass/Share
#
                                                                    InvokeCallMe_OnP
                                                                aram"
5 : Content-Length: 810
6 : Expect: 100-continue
7 : Connection: Keep-Alive
8 : Host: myserver

9 : <SOAP-ENV:Envelope>
10:  <SOAP-ENV:Body>
11:    <i2:InvokeCallMe_OnParam id="ref-1" xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Share.
                                                                    RemotingSamples.HostingClass/Share">
12:      <obj href="#ref-4"/>
13:    </i2:InvokeCallMe_OnParam>
14:    <a1:ForwardMe id="ref-4" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Share
                        .RemotingSamples/Share%2C%20Version%3D1.0.1345.26976%2C%20Cultur
                                                    e%3Dneutral%2C%20PublicKeyToken%3Dnull">
15:    </a1:ForwardMe>
16:  </SOAP-ENV:Body>
17: </SOAP-ENV:Envelope>
```

In den Zeilen 1 bis 8 ist der http-Header dargestellt. Neben der Angabe, dass es sich um eine http-POST-Methode handelt, wird die aufzurufenden Ressource „SayHello" angegeben. SayHello ist der ObjektURI-Name, der beim Anmelden des Objekts am Remoting vergeben wird (s. server-Code). Der http-Header ist zudem um ein spezielles Feld für SOAP erweitert. Das Feld SOAP-Action hat die Aufgabe Informationen über die sich im Anhang an den http-Header befindende Nutzlast bereitzustellen, so dass der Header zur Sicherheit vorverarbeitet und somit die SOAP-Nachricht autorisiert werden kann.
Im Anhang an die http-Methode wird die eigentliche SOAP-Nachricht übermittelt. Die Nachricht wird eingefasst von dem Tag <SOAP-ENV:Envelope > und enthält ein Tag <SAOP-ENV:Body>. Das Body-Tag enthält den Methodenaufruf <i2:InvokeCallMe_OnParam …>. Das Tag für den Methodenaufruf enthält das Tag <obj href="#ref-4"/>, dieses verweist auf den Parameter, der an die Methode `InvokeCallMe_OnParam` übergeben wird, also ein Objekt der Klasse ForwardMe. Da es sich bei dem Parameter des dargestellten Methodenaufruf um ein Objekt handelt, dass mit der MBV-Methode übergeben wird, stellt das Tag <a1:ForwardMe id="ref-4" … > mit der id ref-4 die komplette Kopie des Objektes der Klasse `ForwardMe` aus dem Clientprozess in serialisierter Form dar.

Auf diesen Methodenaufruf antwortet der Server mit einer dem SOAP-Protokoll entsprechenden Nachricht.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Server: MS .NET Remoting, MS .NET CLR 1.0.3705.0
Content-Length: 600

<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <i2:InvokeCallMe_OnParamResponse id="ref-1" xmlns:i2="http://schemas.microsoft.com/clr/nsassem
                                                                    /Share.RemotingSamples.HostingClass/Share">
    </i2:InvokeCallMe_OnParamResponse>
```

```
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Mit dem Tag <i2:InvokeCallMe_OnParamResponse ...> wird dem aufrufendem Objekt, in diesem Fall der Client mitgeteilt, dass der Methodenaufruf bearbeitet wurde. Somit ist der Methodenaufruf abgeschlossen und der Client fährt mit der Programmabarbeitung fort. Die Kommunikation zwischen den beiden Prozessen beläuft sich lediglich auf den Methodenaufruf und die darauf folgende Antwort.

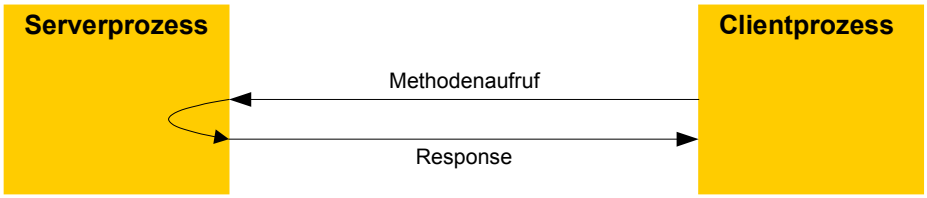

Abbildung 10: schematische Darstellung des Informationsflusses während des Methodenaufrufes mit serialisierbarem Parameter

4.2.2 Darstellung der Methode MBR

Um die Methode MarshalByReference darzustellen, werden die gleichen Komponenten der Anwendung, wie im vorhergehenden Abschnitt beschrieben eingesetzt. Einziger Unterschied ist, dass die Klasse `ForwardMe` nicht als serialisierbar gekennzeichnet ist, sondern statt dessen von der Klasse `MarshalByRefObject` abgeleitete ist.

```
Public Class ForwardMe
    Inherits MarshalByRefObject

    Public Function CallMe(ByVal value As String) As Object
        Console.WriteLine(value)
    End Function
End Class
```

Nachdem Server und Client gestartet sind, ergibt sich ebenso das zuvor beschriebene, in Abbildung 8 dargestellte Szenario. Auch der Aufruf des Clients der Methode `CallMe` des neu erzeugten Objekts der Klasse `ForwardMe` führt zu der beschriebenen Textausgabe auf der Konsole des Clients.
Der Aufruf der Methode `InvokeCallMe_OnParam` an dem Proxyobjekt des entfernten Objekts führt jedoch zu einem anderen Verhalten. Da die Klasse ForwardMe von der Klasse MarshalByRefObject abgeleitet ist, wird als Parameter der Methode `InvokeCallMe_OnParam` an das entfernte Objekt nicht eine Kopie des Objektes übergeben, sondern eine Verweis auf dieses. Das als Parameter übergebene Objekt verbleibt im Prozessraum des Clients, der Server erhält Zugriff auf das Objekt über ein entsprechendes Proxyobjekt, auf das er in seinem Prozessraum, sprich dem Serverprozessraum Zugriff hat.

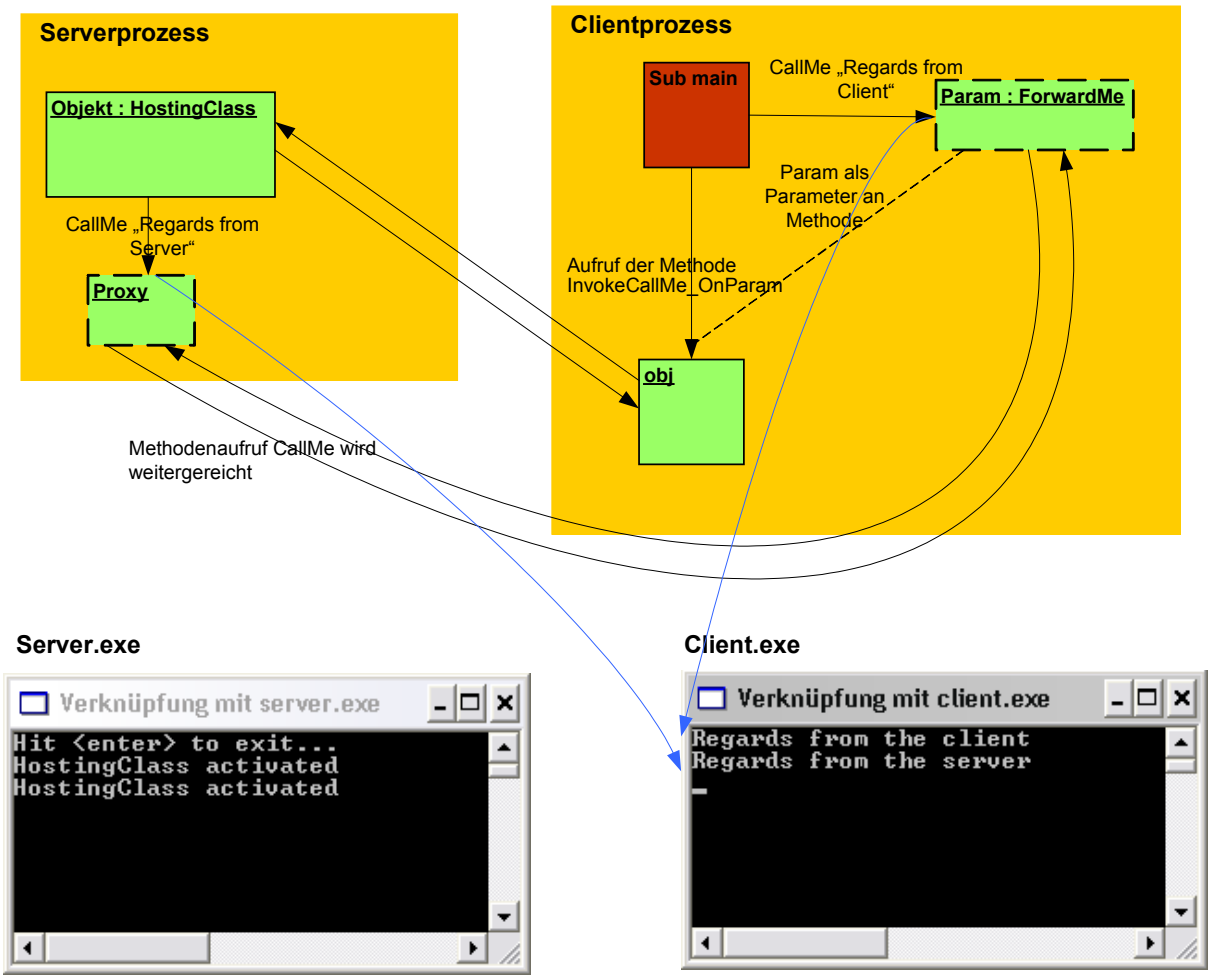

Abbildung 11: Das Objekt `Para` der Klasse `ForwardMe` als Parameter, abgeleitet von der Klasse MarshalByRefObject

Ebenso wie im zuvor beschriebenen Szenario gibt das Proxyobjekt den Aufruf der Methode `InvokeCallMe_OnParam` an den Server weiter und schickt diesem dazu eine http-POST-Nachricht. Anders als zuvor wird hier jedoch im SOAP-Anhang nicht die Kopie des Objekts, welches sich im Prozessraum des Clients befindet und als Parameter an die Methode `InvokeCallMe_OnParam` übergeben wurde, übermittelt, sondern ein Verweis auf diese. Der Verweis ist vom Typ ObjRef.

```
POST /SayHello HTTP/1.1
User-Agent: Mozilla/4.0+(compatible; MSIE 6.0; Windows 5.1.2600.0; MS .NET Remoting; MS .NET CLR
                                                                                    1.0.3705.288 )
Content-Type: text/xml; charset="utf-8"
SOAPAction: "http://schemas.microsoft.com/clr/nsassem/Share.RemotingSamples.HostingClass
                                                                    /Share#InvokeCallMe_OnParam"
Content-Length: 2123
Expect: 100-continue
Connection: Keep-Alive
Host: myserver

<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <i2:InvokeCallMe_OnParam id="ref-1" xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Share.
                                                                    RemotingSamples.HostingClass/Share">
      <obj href="#ref-3"/>
    </i2:InvokeCallMe_OnParam>
    <a1:ObjRef id="ref-3" xmlns:a1="http://schemas.microsoft.com/clr/ns/System.Runtime.Remoting">
      <uri id="ref-4">/3dc9602d_c9c9_44bc_b0ec_a3d832644d4f/3177769_1.rem</uri>
      <objrefFlags>0</objrefFlags>
```

```
    <typeInfo href="#ref-5"/>
    <envoyInfo xsi:null="1"/>
    <channelInfo href="#ref-6"/>
   </a1:ObjRef>
   <a1:TypeInfo id="ref-5" xmlns:a1="http://schemas.microsoft.com/clr/ns/System.Runtime.Remoting">
    <serverType id="ref-7">Share.RemotingSamples.ForwardMe, Share, Version=1.0.1345.20745,
                                                    Culture=neutral, PublicKeyToken=null</serverType>
    <serverHierarchy xsi:null="1"/>
    <interfacesImplemented xsi:null="1"/>
   </a1:TypeInfo>
   <a1:ChannelInfo id="ref-6" xmlns:a1="http://schemas.microsoft.com/clr/ns/System.Runtime.Remoting">
    <channelData href="#ref-8"/>
   </a1:ChannelInfo>
   <SOAP-ENC:Array id="ref-8" SOAP-ENC:arrayType="xsd:anyType[2]">
    <item href="#ref-9"/>
    <item href="#ref-10"/>
   </SOAP-ENC:Array>
   <a3:CrossAppDomainData id="ref-9" xmlns:a3="http://schemas.microsoft.com/clr/ns/System.
                                                                    Runtime.Remoting.Channels">
    <_ContextID>1350392</_ContextID>
    <_DomainID>1</_DomainID>
    <_processGuid id="ref-11">71518cc0_53c6_43fb_ae5e_ae3aa40df4a7</_processGuid>
   </a3:CrossAppDomainData>
   <a3:ChannelDataStore id="ref-10" xmlns:a3="http://schemas.microsoft.com/clr/ns/System.
                                                                    Runtime.Remoting.Channels">
    <_channelURIs href="#ref-12"/>
    <_extraData xsi:null="1"/>
   </a3:ChannelDataStore>
   <SOAP-ENC:Array id="ref-12" SOAP-ENC:arrayType="xsd:string[1]">
    <item id="ref-13">http://192.10.200.71:8086</item>
   </SOAP-ENC:Array>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Als Parameter für den Aufruf der Methode `<i2:InvokeCallMe_OnParam ...>` wird das Objekt `<obj href="#ref-3"/>` übergeben. Dieses ist weiter unten in der SOAP-Nachricht als `<a1:ObjRef id="ref-3" ...>` angegeben. Wichtigstes Element, dass dieses Objekt enthält, ist das Element `<uri id="ref-4">/3dc9602d_c9c9_44bc_b0ec_a3d832644d4f/3177769_1.rem</uri>` . Über das uri-Element wird dem aufgerufenem Objekt der Verweis auf das, der Methode übergebene Objekt geliefert. Des weiteren enthält das ObjRef-Objekt ein CrossAppDomainData-Objekt, das Informationen über den Ausführungsort, wie etwa ProcessGuid, des eigentlichen Objekts enthält. Der Server erhält somit ein Proxyobjekt auf das entfernte Objekt der Klasse `ForwardMe`. Nachdem die Methode `InvokeCallMe_OnParam` aufgerufen wurde und dieser als Parameter der Verweis auf ein Objekt der Klasse `ForwardMe` übergeben wurde, ruft die Methode `InvokeCallMe_OnParam` die Methode `CallMe` des Objekts der Klasse `ForwardMe`, über das erhaltene Proxyobjekt auf. Als Parameter wird der Text `"Regards from the server"` übergeben (s. Share-Code). Dafür sendet der Server die folgende SOAP-Nachricht an den Client. Dabei wird über den Referenzverweis, angegeben in der ersten Zeile des http-Headers, Bezug auf das, den Methodenaufruf zu empfangende Objekt genommen.

```
POST /3dc9602d_c9c9_44bc_b0ec_a3d832644d4f/3177769_1.rem HTTP/1.1
User-Agent: Mozilla/4.0+(compatible; MSIE 6.0; Windows 5.1.2600.0; MS .NET Remoting; MS .NET CLR
                                                                                    1.0.3705.0 )
Content-Type: text/xml; charset="utf-8"
SOAPAction: "http://schemas.microsoft.com/clr/nsassem/Share.RemotingSamples.ForwardMe/Share#CallMe"
Content-Length: 604
Expect: 100-continue
Connection: Keep-Alive
Host: 192.10.200.71

<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <i2:CallMe id="ref-1" xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Share.
                                                                    RemotingSamples.ForwardMe/Share">
      <value id="ref-3">Regards from the server</value>
```

```
   </i2:CallMe>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Das Objekt der Klasse `ForwardMe`, das sich im Prozessraum des Clients befindet, empfängt den Methodenaufruf und führt die Methode CallMe aus. Anschließend sendet es die Responseantwort auf den Methodenaufruf.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Server: MS .NET Remoting, MS .NET CLR 1.0.3705.288
Content-Length: 616

<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <i2:CallMeResponse id="ref-1" xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Share.
                                                                    RemotingSamples.ForwardMe/Share">
      <return xsi:type="xsd:anyType" xsi:null="1"/>
    </i2:CallMeResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Da die Methode `CallMe` als `Function` deklariert ist liefert der Aufruf dieser Methode ein Rückgabewert, in diesem Fall `<return xsi:type="xsd:anyType" xsi:null="1"/>` wird kein Wert zurückgegeben.

Durch die Responsenachricht wird die Programmausführung im Serverprozess weiter abgearbeitet und die Methode `InvokeCallMe_OnParam` beendet, woraufhin der Server an den Client die Beendigung des Aufrufs der Methode `InvokeCallMe_OnParam` mit der folgenden Nachricht mitteilt.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Server: MS .NET Remoting, MS .NET CLR 1.0.3705.0
Content-Length: 600

<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <i2:InvokeCallMe_OnParamResponse id="ref-1" mlns:i2="http://schemas.microsoft.com/clr/nsassem/Share
                                                                    .RemotingSamples.HostingClass/Share">
    </i2:InvokeCallMe_OnParamResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Damit ist der Aufruf der Methode `InvokeCallMe_OnParam` ausgehend vom Clientprozess beendet.
In diesem Szenario ist die Kommunikation etwas umfangreicher als im vorhergehenden Beispiel. Zunächst wird dem Serverprozess ein Verweis auf ein Objekt übergeben. Dieser ruft daraufhin eine Methode an dem für ihn entfernten Objekt auf.

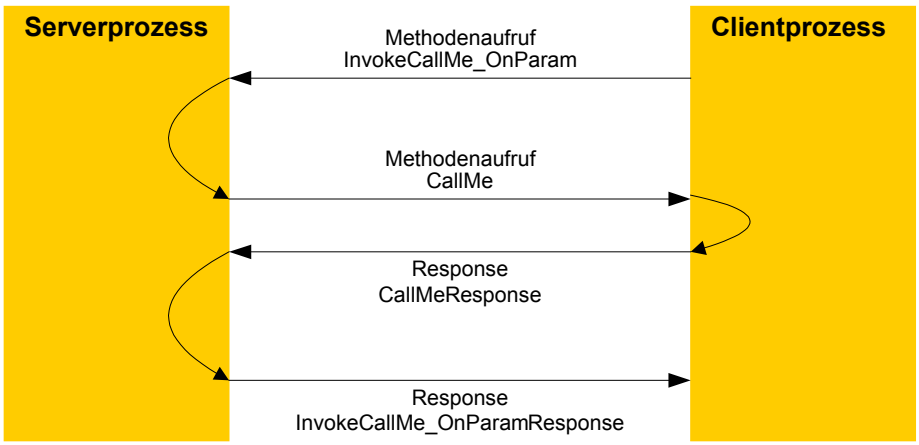

Abbildung 12: schematische Darstellung des Informationsflusses während des Methodenaufrufes mit von `MarshalByRefObject` abgeleitetem Parameter

4.3 Das Leased based lifetime managment system und der Carbage Collector

Der Lebenszyklus eines jeden .NET-Objektes wird durch den Carbage Collector beendet. Der GC entfernt dabei alle Objekte, die nicht mehr „erreichbar" sind, d.h. auf die kein anderes Objekt eine Referenz hält. Um die Erreichbarkeit eines Objektes zu ermitteln, durchläuft der Garbage Collector alle Objektreferenzgraphen angefangen bei deren Wurzelelementen und markiert alle Objekte, die er erreicht. Im weiteren Schritt werden alle nicht markierten Objekte vom Garbage Collector aus dem Speicher entfernt und deren Ressourcen wieder frei gegeben, zudem defragmentiert der Garbage Collector in diesem Schritt den Speicher.

Die recht simple Methodik des Garbage Collector, alle Objekte zu entfernen auf die kein anderes aktives Objekt einen Verweis hält, ist besonders für den Einsatz in lokalen Anwendungen geeignet. Das Konzept lässt sich jedoch nicht ohne weiteres auf verteilte Systeme anwenden. Besonders in verteilten Anwendungsstrukturen, in denen eine Vielzahl an verbindungslosen Clients Referenzen auf entfernte Objekte halten, kann diese Methode nicht eingesetzt werden.
Die Implementierung des performanten und skalierbaren Carbage Collectors, der in lokalen .NET-Anwendungen eingesetzt wird, wird deshalb in .NET-Remoting-Applikationen nicht verwendet. Um das .NET-Konzept auch im .NET-Remoting durchgängig zu halten und das Speichermanagement für den Entwickler von verteilten Anwendungen auch dort transparent zu gestalten, verwendet das .NET-Remoting ein sogenanntes "lease based lifetime management system" in Kombination mit dem „normalen" Carbage Collector.
Dieses System sorgt dafür, das für jedes aktivierte MBR-Objekt auf Serverseite ein sogenanntes lease-Objekt angelegt wird. Somit existiert zu jedem aktivierten Objekt, das abgeleitet ist von `MarshalByRefObjekt` ein lease-Objekt. Die Aufgabe des lease-Objektes ist, die Lebenszeit des MBR-Objektes zu verfolgen. Dafür hält das lease-Objekt über die Eigenschaft `CurrentLeaseTime` vom Typ `TimeSpan` die aktuell verbleibende Lebensspanne des MBR-Objektes bereit. Neben der Eigenschaft `CurrentLeaseTime`, die die Lebenszeit des MBR-Objektes verfolgt, stellt das lease-Objekt die Eigenschaft `RenewOnCallTime` zur Verfügung. Der Wert der Eigenschaft wird vom .NET-Remotingframework verwendet, um die Lebenszeit des MBR-Objekt bei einem Methodenaufruf zurückzusetzen. Dadurch, das das lease-Objekt lediglich die Aufgabe hat, die Lebenszeit eines MBR-Objekts zu verfolgen, wird es nicht verwendet um die Zeitspanne nach einem Methodenaufruf zurückzusetzen, oder etwa die aktuelle Lebensspanne zu dekrementieren. Diese Aufgaben werden durch zwei weitere Klasse des .NET-Remoting-Frameworks erledigt. Für das Dekrementiern der Zeitspanne eines MBR-Objekts ist der LeaseManager zuständig. Diese Klasse benutzt einen eigenen Thread, in dem die lease-Objekte aller gemarshallten MBR-Objekte, d.h. aller MBR-Objekte die von Clients aktiviert wurden, überprüft und deren Lebensspanne dekrementiert werden. Der LeaseManager dekrementiert somit nicht direkt die Lebenszeit eines MBR-Objektes, sondern die des zu diesem gehörenden lease-Objekts.
Die zweite Klasse ist die Klasse LeaseSink. Diese Klasse setzt bei einem Methodenaufruf an das MBR-Objekt die Lebenszeit des zu diesem MBR-Objekt korrespondierenden lease-Objekts zurück.
Damit ein gemarshalltes MBR-Objekt nach Ablauf der Lebenszeit seines lease-Objektes entfernt wird, werden die beiden Systeme, das des Carbage Collectors und das lease based lifetime managment system verknüpft.
Wird ein MBR-Objekt über das .NET-Remotingsystem gemarshallt wird ein sogenanntes Identity-Objekt erzeugt. Diese Identity-Objekt verweist auf das serverseitige MBR-Objekt, zudem enthält das Objekt die objektURI, welche beim Marshallen angegeben wird. Über diese objektURI wird das MBR-Objekt für andere Objekte im verteilten System erreichbar gemacht. Nach dem das Identity-Objekt erzeugt wurde, wird es in einer HashTable abgelegt. Dabei wird die objectURI als Schlüssel innerhalb der HashTable verwendet. Diese HashTable wird benutzt, um eingehende Anfragen an ein MBR-Objekt

gekennzeichtet durch dessen objectURI aufzuschlüsseln und diese an das MBR-Objekt weiter zu reichen.

Key	Value
/URI/to/Object1	Identity -> Object#1
/URI/to/Object2	Identity -> Object#2
/URI/to/Object3	Identity -> Object#3

Abbildung 13: Der Remotingstack

Empfängt der Remotingstack eine Anfrage mit dem Ziel objectURI wird über die HashTable das korrespondierende Identity-Objekt ermittelt und die Anfrage an das MBR-Objekt weitergegeben.

Der Carbage Collector ist alleine nicht in der Lage, Objekte aus der HashTable zu entfernen, dies geschieht mit Hilfe des lease based lifetime managemant systems des .NET-Remotings. Sobald die Lebensspanne eines lease-Objektes null erreicht hat entfernt das .NET-Remoting das Identity-Objekt des korrespondierenden MBR-Objekts aus der HashTable. Dadurch wird das MBR-Objekt nicht mehr erreichbar für Clients des MBR-Objekts. Da das Identity-Objekt gelöscht wurde existiert zudem keine Referenz mehr auf das MBR-Objekt und somit wird dieses vom Garbage Collector aus dem Speicher entfernt.

4.3.1 Manipulation der Aufenthaltsdauer eines MBR-Objekts im Speicher

Bei dem Einsatz von Objekten, die abgeleitet sind von `MarshalByRefObject`, wird das in 4.3 dargestellte leased based lifetime managment system eingesetzt. Mit diesem wird die Aufenthaltsdauer eines MBR-Objekt im Speicher festgelegt und gesteuert, dabei kann die Lebensspanne eines MBR-Objekt von außen gesteuert werden.
Die Lebensdauer eines MBR-Objekt kann im Wesentlichen auf zwei verschiedene Weisen geändert werden. Zum einen kann über das lease-Objekt eines jeden MBR.Objekts über die Eigenschaft InitialLeaseTime gesetzt werden. Der Wert der Eigenschaft legt fest, wie lange das Objekt im Speicher verbleibt.
Zum anderen kann jeder Client über den Methodenaufruf `Renew` des lease-Objekts des entfernten MBR-Objekts dessen Lebensdauer im Speicher des Servers verlängern.

Standardmäßig ist die Lebensdauer eines MBR-Objekts mit fünf Minuten vorgegeben. Das folgende Code-Beispiel zeigt, wie diese Zeitspanne auf eine Minute geändert werden kann.

```
Public Class LSClass
    Inherits MarshalByRefObject

    Public Overrides Function InitializeLifetimeService() As Object
        Dim lease As ILease = MyBase.InitializeLifetimeService()

        If lease.CurrentState = LeaseState.Initial Then
            lease.InitialLeaseTime = TimeSpan.FromMinutes(1)
        End If
        Return lease
    End Function
End Class
```

Dazu wird die Methode `InitializeLifetimeService` überschrieben. Das geänderte lease-Objket wird von der Methode zurückgegeben.

Das Ändern der Eigenschaft `InitialLeaseTime` geschieht zu einem Zeitpunkt geschehen, in dem sich das lease-Objekt im Status initial befindet

5 SOM-Objektmodellierung mit Visual SOM Studio

5.1 Die Anwendung Visual SOM Studio

In heute aktuellen Computersystemen ist die Darstellung und Manipulation von Daten über graphische Benutzerschnittstellen Stand der Technik. Mussten früher komplexe Daten oder Konfiguration per Hand mit einfachen Texteditoren bearbeitet werden, so ist der Anwender heute gewohnt solche Tätigkeiten über komfortable graphische Oberflächen durchzuführen. Neben der Eigenschaft Anwendungen über graphische Schnittstellen leicht und intuitiv bedienen zu können, bieten graphische Benutzeroberflächen die Möglichkeit, dass der Personenkreis der Anwender nicht auf wenige Anwendungsspezialisten beschränkt bleiben muss. Der Einarbeitungsaufwand für die Verwendung einer Applikation wird minimiert und Anwendungen können als arbeitsentlastendes Hilfsmittel eingesetzt werden.
Besonders im Umfeld von Projektierungsaufgaben ist der Einsatz von Anwendungen mit graphischen Schnittstellen, die leicht und intuitiv zu bedienen sind, von großer Bedeutung. Hier ist es wichtig, den Stand eines Projekts jederzeit überschaubar darzustellen, was mit Hilfe von graphischen Methoden meist am effektivsten zu bewerkstelligen ist. Ebenso sind Anwender mit der Projektierung betraut, deren Kernkompetenz im Themenbereich des Projektes liegen und nicht unbedingt im Bereich der Computertechnik. Dem Anwender wird durch graphische Oberflächen die Möglichkeit gegeben sich auf seine Kernkompetenz zu konzentrieren ohne von einer komplexen Anwendungssteuerung abgelenkt zu werden.

Die im Rahmen dieser Arbeit entwickelte Anwendung Visual SOM Studio hat die Aufgabe ein Werkzeug bereitzustellen, mit dem die Sekundärtechnik von Unterstationen nach der Norm IEC 61850 Teil 7, basierend auf dem SOM-Schnittstellenmodell, modelliert werden kann. Mit Visual SOM Studio lassen sich mittels graphischer Methoden hierarchische Objektmodelle generieren, verwalten und manipulieren, die das SOM-Schnittstellenmodell unterstützen.
Um die einfache und intuitive Bedienung der Anwendung zu erreichen, ist das Design von Visual SOM Studio stark an existierende Anwendungen aus dem Windowsbereich angelehnt. Zudem ist Visual SOM Studio in MDI (Multiple Document Interface)-Technik implementiert, um dem Anwender verschiedene Ansichten auf das zu bearbeitende Objektmodell bereitzustellen. Ebenso verwendet Visual SOM Studio überwiegend Standardsteuerelemente des .NET-Frameworks, wie etwa das TreeView-Steuerelement, um hierarchische Strukturen der Objektkonfiguration darzustellen, oder das PropertyGrid-Steuerelement, um Eigenschaften von einzelnen Objekten zu manipulieren. Einzelne Komponenten, die nach IEC 61850 Teil 7 für die Modellierung der Sekundärtechnik von Unterstationen benötigt werden, werden im Modelldesigner durch graphische Objekte dargestellt. Über die Maus können diese Objekte eingefügt, platziert und deren Position innerhalb der Objekthierarchie des SOM-Schnittstellenmodell bestimmt werden. Die Hierarchie einer Objektkonfiguration wird durch Verschachtelung von einzelnen graphischen Objekten erreicht.

5.2 Dokumentelemente von Visual SOM Studio

Die Anwendung Visual SOM Studio wird durch die Ausführung der Datei VSS.exe gestartet. Beim erstmaligen Starten der Anwendung erscheint das leere Hauptfenster, wie in Abbildung 14 dargestellt.

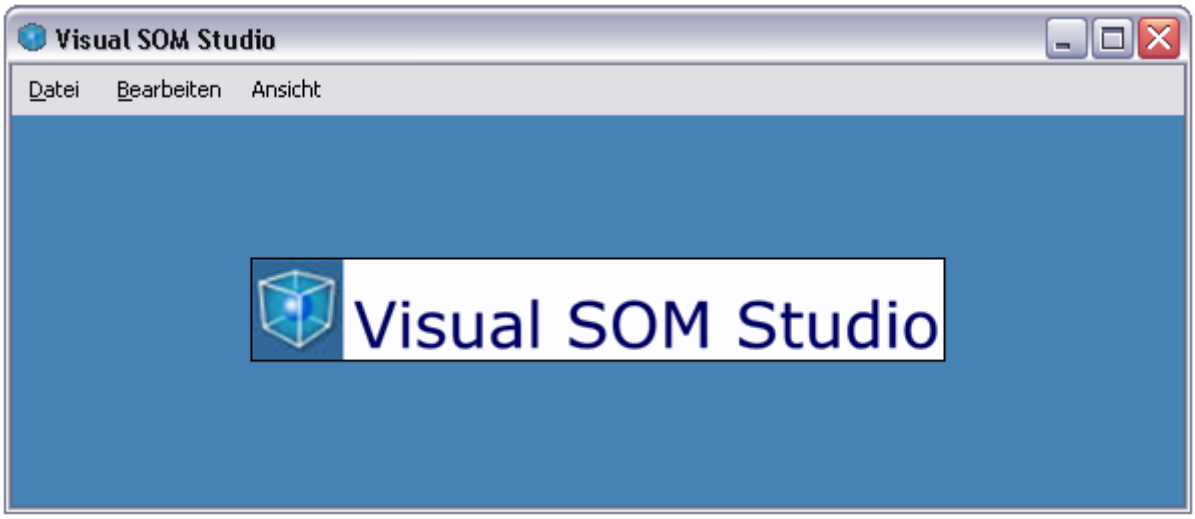

Abbildung 14: Leeres Hauptfenster von Visual SOM Studio

Über die Menüleiste wird die Anwendung gesteuert, sowie deren Elemente verwaltet.

5.2.1 Die Projektmappe

Die Anwendung Visual SOM Studio verwaltet die anfallenden Daten in einem hierarchischen System. An erster Stelle der Hierarchie steht die Projektmappe. Visual SOM Studio kann nicht mehrere Projektmappen gleichzeitig anzeigen, so dass lediglich eine Mappe zur Bearbeitung bereitsteht. Die Projektmappe ist für die Verwaltung von beliebig vielen Projekten zuständig. Somit ist es möglich eine beliebige Anzahl von Projekten mit Visual SOM Studio gleichzeitig zu bearbeiten.
Über den Menüpunkt

<Datei> / <Neu> / <Projektmappe>

wird eine neue Projektmappe angelegt. Diese erhält beim Anlegen den Standardnamen *Neue Projektmappe*. Ebenso erhält die Projektmappe eine Standardangabe über deren Speicherort in Form einer vollständigen Pfadangabe. Die so standardmäßig gesetzten Eigenschaften der Projektmappe können im Eigenschaftsfenster (siehe Abschnitt 5.3.4) über die Eigenschaften Name und Datei geändert werden.
Über den Menüpunkt

<Datei> / <Alles Speichern>

wird die gesamte Projektmappe gespeichert. Dabei werden alle enthaltenen Projekte und deren Daten in der Projektmappendatei gespeichert. Die Projektmappendatei enthält somit die gesamten Daten aller in der Projektmappe enthaltenen Projekte und kann so als Einheit verteilt werden. Die Projektmappendatei wird mit der Dateiendung .ssf (SOM Studio File) gespeichert.
Dargestellt wird die Projektmappe innerhalb der Anwendung Visual SOM Studio in dem Ansichtfenster Projekt Explorer (siehe Abschnitt 5.3.2)

5.2.2 Das Projekt

Das zweite Element innerhalb des hierarchischen Verwaltungssystems von Visual SOM Studio ist das Projektelement. Projekte werden mit der Projektmappe verwaltet. Jedes Projekt enthält standardmäßig ein graphisches SOMObjekt, welches ein konkretes Objekt, dass die Schnittstelle IServer des SOM-Schnittstellenmodells implementiert, enthält. Dieses SOMObjekt kann aus dem Projekt nicht entfernt werden. Somit wird festgelegt, dass ein Projekt mindestens einen Server des SOM-Schnittstellenmodells enthalten kann.
Über den Menüpunkt

<Datei> / <Neu> / <Projekt>

wird einer vorhandenen Projektmappe ein neues Projekt hinzugefügt. Dabei wird dem Projekt ein Standardname zugewiesen.
Über den Menüpunkt

<Bearbeiten> / <Löschen>

kann ein bestehendes Projekt aus einer Projektmappe entfernt werden. Dabei muss das zu löschende Projekt ausgewählt sein. Das Auswählen eines Projektes geschieht über den Projekt Explorer oder aber in der Objekt Ansicht (siehe Abschnitt 5.3.1).
Ebenso wie die Projektmappe verfügt das Projekt über eine Eigenschaft, die den Speicherort des Projektes angibt. Diese Eigenschaft des Projekts ist dafür vorgesehen, dass Projekte unabhängig von der Projektmappe gespeichert werden können. Visual SOM Studio verwendet diese Möglichkeit nicht. Alle Projekte und deren enthaltenen Daten werden zusammen in der Projektmappendatei gespeichert.
Das Projekt wird innerhalb des hierarchischen Systems von Visual SOM Studio als erstes Element graphisch in der Objekt Ansicht angezeigt. In der Objekt Ansicht wird ein Projekt als leere Zeichenfläche dargestellt. Auf dieser Zeichenfläche können beliebige SOMObjekte angelegt und bearbeitet werden. Da jedes Projekt standardmäßig ein SOMObjekt, das auf ein Objekt vom Typ IServer verweist, enthält, wird dieses SOMObjekt immer auf der Zeichenfläche des Projektes dargestellt.

5.2.3 Das SOMObjekt

Das SOMObjekt ist das Verbindungsglied zwischen existierenden Objekten, die Schnittstellen des SOM's implementieren, und Visual SOM Studio. Mit Hilfe der SOMObjekte können innerhalb von Visual SOM Studio die Daten für eine Modellkonfiguration basierend auf dem SOM-Schnittstellenmodell erstellt werden. Die SOMObjekte stellen dabei die graphische Repräsentation eines konkreten Objektes, das eine Schnittstelle des SOM's unterstützt, innerhalb eines Projektes dar. Um eine hierarchische Objektstruktur, wie sie die Norm IEC 61850 Teil 7 und damit auch das SOM-Schnittstellenmodell fordern, modellieren zu können, sind die SOMObjekte in der Lage weitere SOMObjekte zu enthalten.
Das oberste SOMObjekt in der hierarchischen Dokumentstruktur von Visual SOM Studio ist das, zu jedem Projekt gehörende SOMObjekt, welches auf ein Objekt, das die SOM-Schnittstelle IServer implementieren, verweist. Dieses kann, der Hierarchie ILogicalDevice, ILogicalNode, IDataObject und IDataAttribute folgend, weitere SOMObjekte enthalten.
Über die Eigenschaften eines SOMObjekts kann im Eigenschaftsfenster der Verweis auf eine konkrete Klasse des SOM's hergestellt werden. Um eine vollständige Objektkonfiguration zu erstellen, ist es notwendig, dass jedes SOMObjekt eines Projektes einen Verweis auf eine existierende Klasse des SOM'S hält.
Die SOMObjekte stellen, wie bereits oben erwähnt, eine graphische Schnittstelle bereit. Die Anzeige der SOMObjekte erfolgt in der Objekt Ansicht auf der Zeichenebene eines Projektes. Die SOMObjekte werden in Form von Rechtecken, die in Größe, Form und Hintergrundfarbe veränderbar sind, dargestellt. Zudem wird der Name eines SOMObjekts

im linken oberen Bereich des Rechteckes angezeigt. Alle nicht schreibgeschützten Eigenschaften des SOMObjekts werden mit Hilfe des Eigenschaftsfernsters manipuliert.

5.3 Ansichten und Werkzeuge von Visual SOM Studio

Visual SOM Studio stellt zwei unterschiedliche Ansichten auf das Anwendungsdokument, sprich die Projektmappe und alle in dieser enthaltenen Dokumentelemente, bereit.
Der Projekt Explorer stellt die hierarchische Struktur des Anwendungsdokuments dar. Die Objekt Ansicht hingegen stellt das Anwendungsdokument in graphischer Form dar, wobei auch in dieser Ansicht die Hierarchie, hier jedoch durch Verschachtelung der einzelnen graphischen Objekte, dargestellt wird. Neben der Darstellung des Anwendungsdokuments wird mit Hilfe der Objekt Ansicht die Konfiguration des Dokuments erstellt und bearbeitet.
Neben den beiden Dokumentansichten stellt Visual SOM Studio zwei Werkzeuge für die Bearbeitung des Anwendungsdokumentes bereit. Mit dem Werkzeug Toolbox können in das Dokumentelement Projekt neue SOMObjekte, die auf ganz bestimmte Objekte des SOM's verweisen können, eingefügt werden.
Mit dem Werkzeug Eigenschaftsfenster werden die individuellen Eigenschaftswerte der einzelnen Dokumentelemente angezeigt und bearbeitet.
Alle Ansichten und Werkzeuge werden über den Menüpunkt <Ansicht> und den entsprechenden Unterpunkt angezeigt.

5.3.1 Die Objekt Ansicht

Die Objekt Ansicht wird über den Menüpunkt
<Ansicht> / <Objekt Ansicht>
aufgeschaltet. Sie dient dem graphischen Anzeigen von Projekten und deren enthaltenen SOMObjekten. Zudem ist die Objekt Ansicht der zentrale Anwendungsbereich, in dem die Konfiguration eines, auf dem SOM basierenden Modells, erstellt wird.
In der Objekt Ansicht können einzelne SOMObjekte selektiert werden. Dabei wird die Selektion durch Objektgriff rings um ein SOMObjekt herum angezeigt.

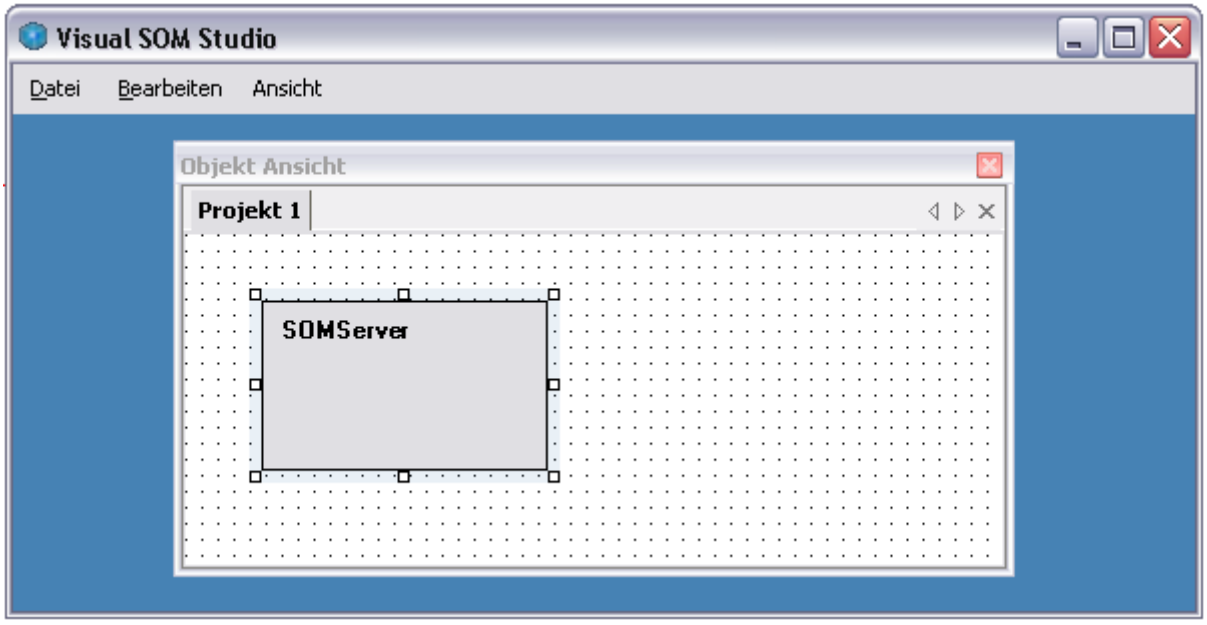

Abbildung 15: Objekt Ansicht von Visual SOM Studio

Mit der Maus wird das Objekt über die Griffe in seiner Form verändert. Ein Objektgriff wird „angefasst" indem die Maus über einen entsprechenden Griff geführt wird und die linke Maustaste gedrückt wird. Solange die linke Maustaste gedrückt bleibt, kann der Griff bewegt und damit die Form des Objektes geändert werden.
Die Griffe des Objekts in der Mitte einer jeden Seite verändern dessen Form in horizontaler bzw. vertikaler Richtung, wohingegen die Griffe in den Ecken des Objekts dieses in horizontaler und vertikaler Ausdehnung gleichzeitig verändern. Die Position des Objekts wird dadurch verändert, indem die linke Maustaste über dem Objekt selber gedrückt und gehalten wird, bis mit der Bewegung der Maus die neue Position erreicht ist.
Mit Hilfe der Toolbox (siehe 5.3.3) werden in der Objekt Ansicht neue SOMObjekte in bestehende SOMObjekte eingefügt und somit die Objektkonfiguration erweitert.

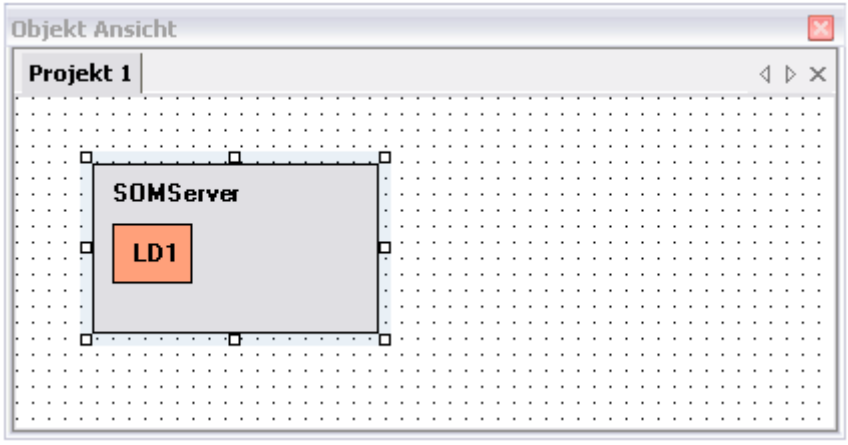

Abbildung 16: neu hinzugefügtes SOMObjekt

Neue SOMObjekte werden immer in das zuvor selektierte SOMObjekt eingefügt.
Durch Selektieren eines SOMObjekts in der Objekt Ansicht werden die Eigenschaften dieses Objektes im Eigenschaftsfenster (siehe 5.3.4) angezeigt.

Die Objekt Ansicht stellt über Registerkarten alle Projekte, die in einer Projektmappe enthalten sind, zur direkten Auswahl bereit. So kann die Ansicht von Projekt zu Projekt einfach über die Auswahl der jeweiligen Registerkarte erfolgen.

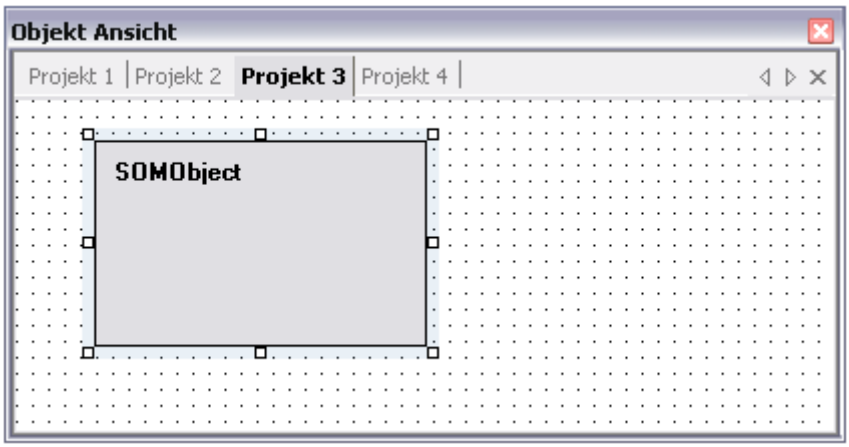

Abbildung 17: Darstellung der Projekte in Registerkarten

Durch Selektion einer Registerkarte mit der Maus wird das Projekt in der Objekt Ansicht angezeigt.

5.3.2 Der Projekt Explorer

Der Projekt Explorer hat die Aufgabe die Projektmappe, deren enthaltenen Projekte, sowie die Objektkonfigurationen der einzelnen Projekte hierarchisch darzustellen. Der Projekt Explorer dient lediglich dem Anzeigen nicht aber dem Manipulieren des Anwendungsdokuments. So können z.B. im Projekt Explorer die Namen der einzelnen Dokumentelemente nicht bearbeitet werden. Alle Manipulationen erfolgen über das Eigenschaftsfenster (siehe 5.3.4).

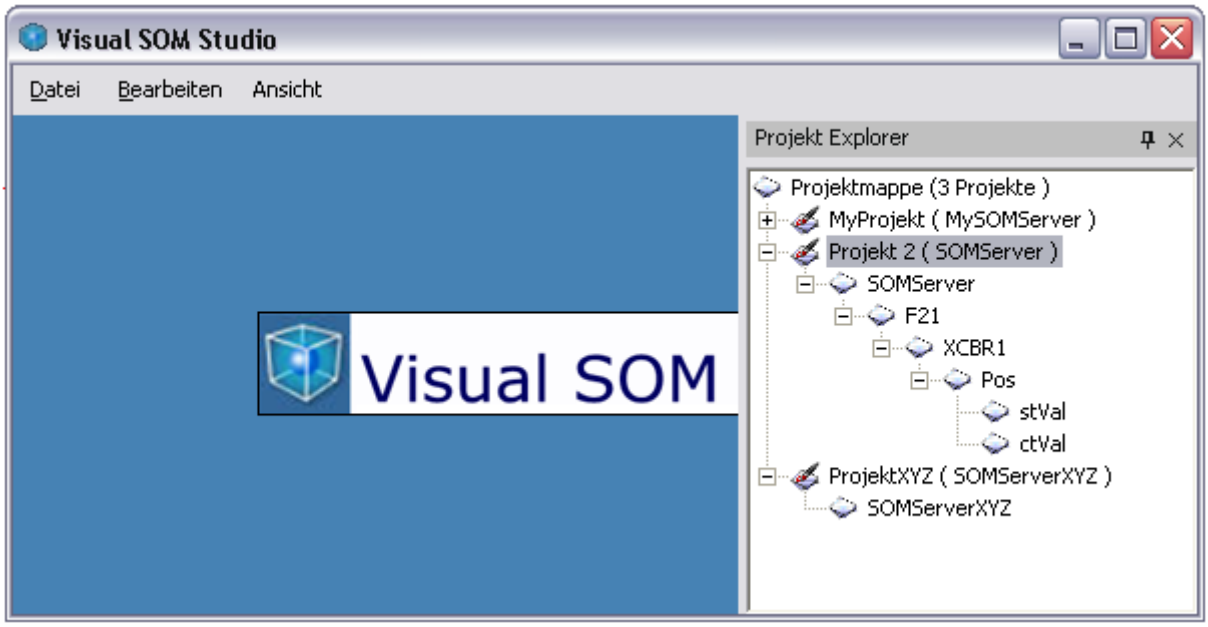

Abbildung 18: Projekt Explorer

In der obersten Hierarchieebene wird der Name der Projektmappe angezeigt. Hinter dem Projektmappenname wird zusätzliche die Anzahl der in der Projektmappe enthaltenen Projekten angezeigt. In der zweiten Hierarchieebene, unterhalb der Projektmappe werden die einzelnen Projekte namentlich aufgelistet, wobei hinter jedem Projektname der Name des in dem Projekt enthaltenem SOMServers angezeigt wird. Unterhalb eines jeden Projektes werden der Hierarchie folgend alle im Projekt enthaltenen SOMObjekte aufgelistet.

5.3.3 Die Toolbox

Die Toolbox hat die Aufgabe über einfache Methoden neue SOMObjekte, die auf einen bestimmten Objekttyp des SOM's verweisen können, in ein Projekt einzuzufügen. Dafür stellt die Toolbox vier Schaltflächen bereit, über die die jeweiligen SOMObjekte neu angelegt werden und in ein Projekt eingefügt werden. Neue SOMObjekte werden immer in das aktuell, in der Objekt Ansicht selektierte SOMObjekt eingefügt.

Abbildung 19: Toolbox

So kann einem SOMObjekt, dass auf ein SOMServerobjekt verweist, über den Schalter LogicalDevice neue SOMObjekte, die Verweise auf LogicalDeviceobjekte des SOM's halten können, hinzugefügt werden (siehe z.B. Abbildung 16).

5.3.4 Das Eigenschaftsfenster

Aufgabe des Eigenschaftsfensters ist es, Eigenschaften eines Dokumentelements anzuzeigen und für diese, soweit sie nicht schreibgeschützt sind, Editierfunktionen bereit zustellen. Über das Eigenschaftsfenster werden Name und Datei, sprich Speicherort der Dokumentelemente Projektmappe und Projekt gesetzt. Ebenso werden im Eigenschaftsfenster die graphischen Eigenschaften eines SOMObjektes eingestellt. Eigenschaften eines SOMObjekts werden im Eigenschaftsfenster angezeigt, sobald das entsprechende SOMObjekt in der Objekt Ansicht selektiert wird.

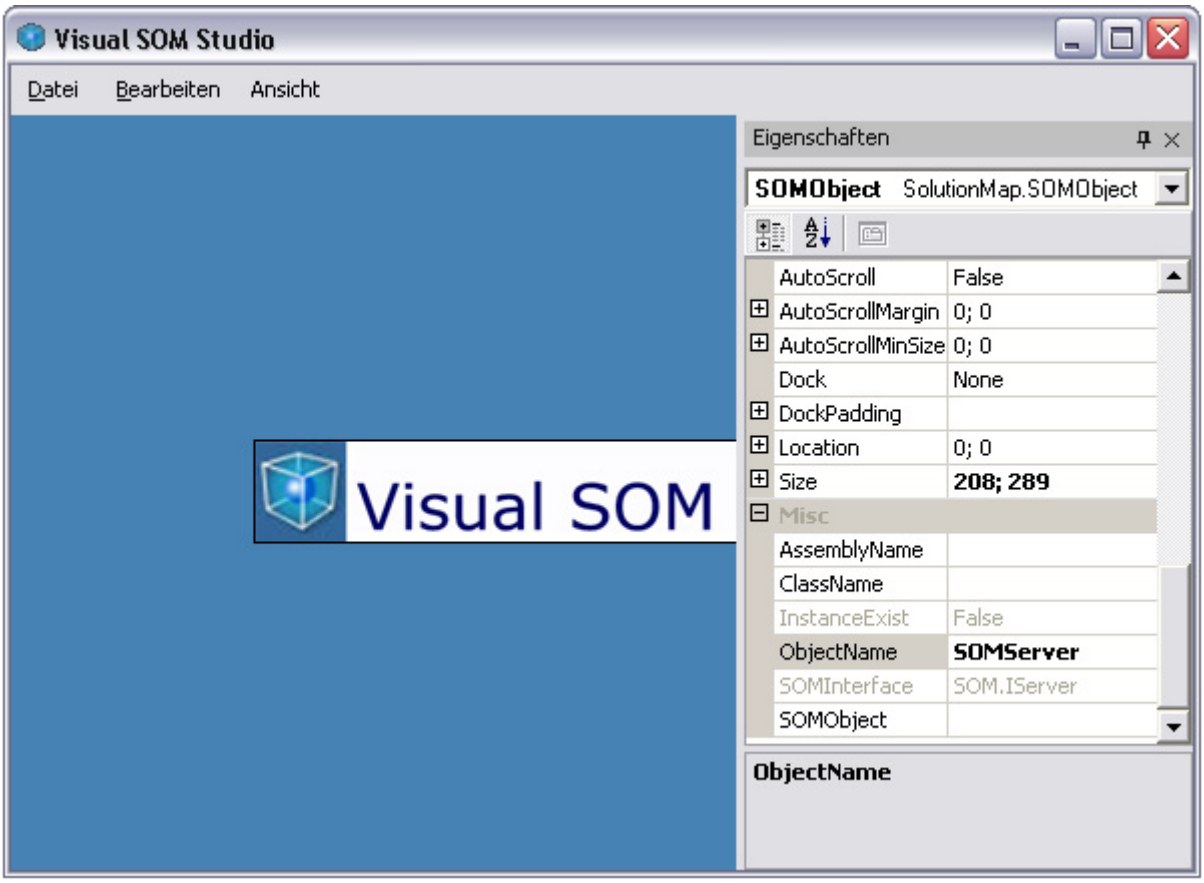

Abbildung 20: Eigenschaftsfenster

Im Eigenschaftsfenster werden dann alle Eigenschaften des SOMObjekts, wie etwa Position, Größe und Hintergrundfarbe dargestellt. Ebenso wird über das Eigenschaftsfenster der Verweis auf eine konkrete Klasse, die eine Schnittstelle des SOM's implementiert, hergestellt. Dieser Verweis kann auf zwei verschieden Arten über das Eigenschaftsfenster gesetzt werden.
Im ersten Falle wird zunächst über die Eigenschaft AssemblyName das Assembly ausgewählt, das die entsprechende Klasse enthält. Daraufhin wird in dem Editierfeld der Eigenschaft ClassName im Eigenschaftsfenster ein Liste gefüllt, die alle Klassen des ausgewählten Assemblies enthält, die die durch die Eigenschaft SOMInterface spezifizierte Schnittstelle implementieren.

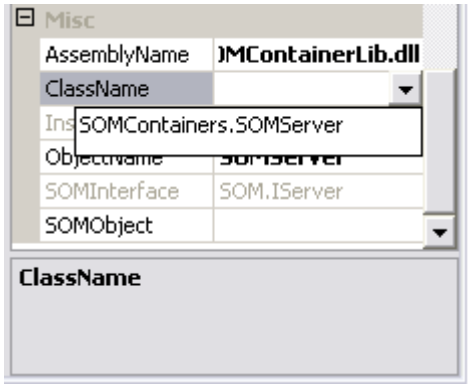

Abbildung 21: Anzeige aller Klassen, die die Schnittstelle SOM.IServer implementieren

Nach dem beide Eigenschaften gesetzt wurden, wird zur Kontrolle der Instanzierbarkeit eine Instanz dieser Klasse erzeugt. Dies wird über den booleschen Wert der Eigenschaft InstanceExist angezeigt.
Im zweiten Fall kann über die Eigenschaft SOMObject die Referenz auf eine Klasse, die eine Schnittstelle des SOM's implementiert, hergestellt werden. Durch selektieren des Editierfeldes der Eigenschaft SOMObjekt im Eigenschaftsfenster wird eine Schaltfläche angezeigt, nach deren Betätigung der folgende Dialog aufgeschaltet wird.

Abbildung 22: der SOMObjekt Selektor

Hier wird zunächst das Assembly ausgewählt , das die entsprechenden Klassen enthält. Nach der Auswahl eines Assemblies werden alle Klassen, die die jeweilige Schnittstelle implementieren, in Abbildung 22 die Schnittstelle IServer, aufgelistet. Durch die Auswahl einer Klasse und Bestätigung mit „übernehmen", wird der Verweis auf diese Klasse erzeugt.

5.4 Beispielhafte Konfiguration eines Leistungsschalters basierend auf IEC 61850 Teil 7-4 mit Visual SOM Studio

Das im folgenden dargestellte Projekt veranschaulicht die Konfiguration eines Leistungsschalters XCBR basierend auf dem SOM-Schnittstellenmodell nach IEC 61850 Teil 7-4. Das Projekt erhebt dabei nicht den Anspruch auf Vollständigkeit. Es sollen lediglich die Möglichkeiten von Visual SOM Studio dargestellt werden.

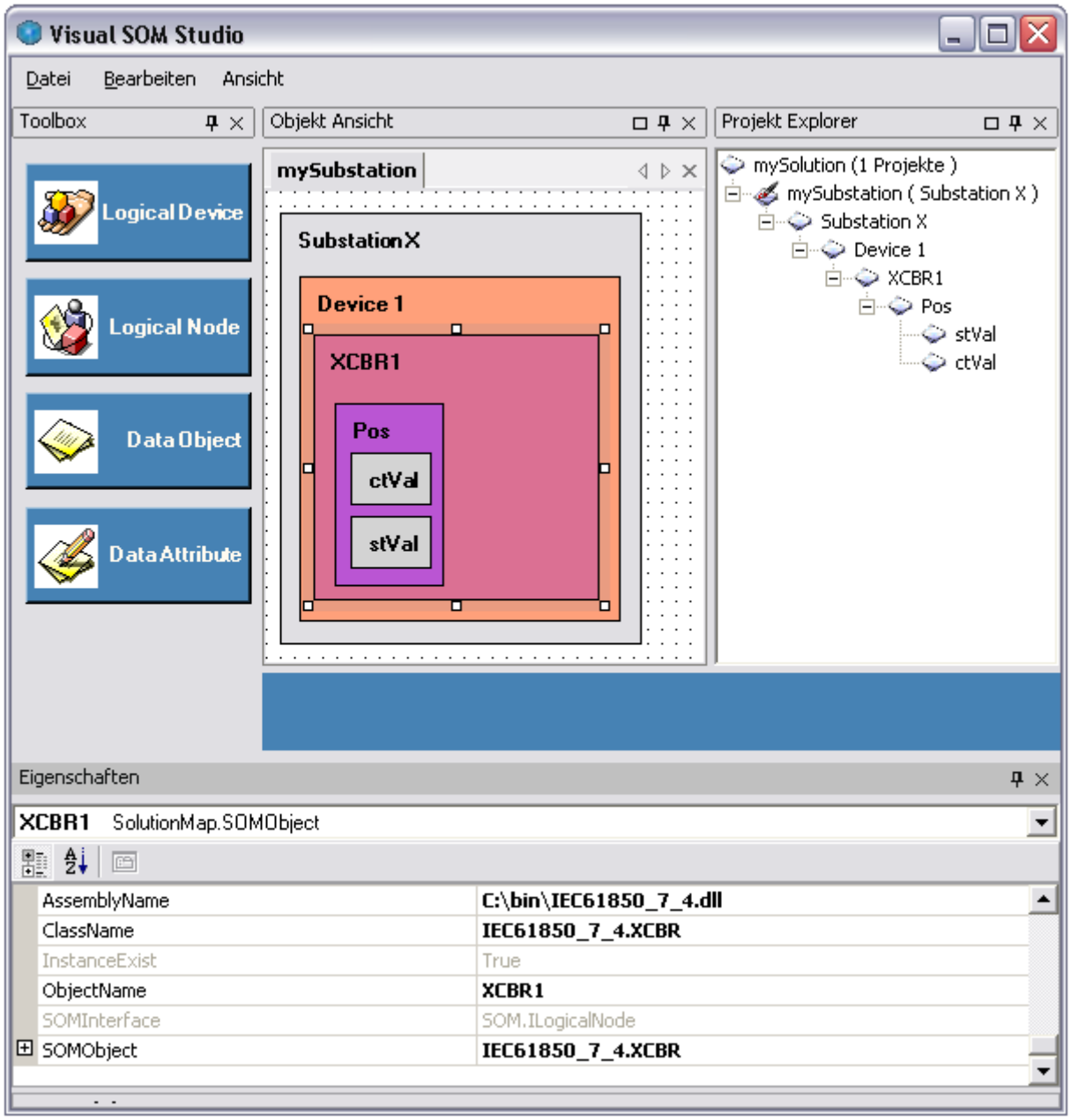

Abbildung 23: Beispielmappe mySolution

Auf oberster Ebene des SOM-Schnittstellenmodells befindet sich, hier grau dargestellt, der Server mit dem Namen SubstationX. Dieser enthält ein logisches Gerät (LogicalDevice), hier orange dargestellt, mit dem Namen Device 1. Das logische Gerät Device 1 enthält den logischen Knoten (LogicalNode) XCBR1. Der logische Knoten XCBR1 steht als Repräsentant des realen Leistungsschalters. Über das Datenobjekt (DataObject) Pos des XCBR's und die in diesem enthaltenen Datenattribute (DataAttribute) wird der

Leistungsschalter verwaltet. Über das Datenattribut ctVal (controlValue) wird der Leistungsschalter gesteuert und über das Datenattribut stVal (statusValue) kann die Leistungsschalterstellung ermittelt werden.

Das violett dargestellte graphische SOMObjekt des XCBR1 weißt, dargestellt im Eigenschaftsfenster, auf die Klasse XCBR des Assemblies IEC61850_7_4.dll. Diese Klasse implementiert die SOM-Schnittstelle IlogicalNode. Ebenso wie der XCBR1 weisen alle andern Objekte auf entsprechende Klassen, die das SOM-Schnittstellenmodell unterstützen. Auf diese Weise kann mit Visual SOM Studio unter Verwendung des SOM's eine Konfiguration für ein Objektmodell basierend auf der Norm IEC 61850 Teil 7 erstellt werden.

Die erzeugte Konfiguration wird in der Datei mySolution.ssf in einem Binärformat gespeichert. Somit kann die mit Visual SOM Studio erstellt Konfiguration verteilt werden und in anderen Anwendungen verwendet werden.

6 Softwaredokumentation der Anwendung Visual SOM Studio

6.1 Die Komponente VSS

6.1.1 Aufgaben der Komponente VSS

Die Komponente VSS stellt Klassen bereit, die es dem Anwender ermöglichen auf graphische Weise Objekte der Geschäftslogik zu bearbeiten.
Damit stellt die Komponente VSS die Präsentationsebene der Anwendung Visual SOM Studio dar.
Die graphische Benutzerschnittstelle der Anwendung Visual SOM Studio ist als MDI-Anwendung (multiple document interface) implementiert. Zentraler MDI-Container der Anwendung ist das Formular FVSS. Die Klasse FVSS stellt zur Laufzeit den Bearbeitungsrahmen der Anwendung Visual SOM Studio, in dem alle anderen Formulare angezeigt werden, dar. Die Klasse FVSS verfügt über eine Menübar, über die die Anwendung Visual SOM Studio gesteuert wird. So lassen sich über die Toolbar neue Projekte anlegen oder aber bereits vorhandenen Projekte laden. Ebenso werden über die Toolbar die verschiedenen Werkzeuge und Ansichten, die die Anwendung Visual SOM Studio bereit stellt, aufgeblendet.
Die Komponente VSS beinhaltet neben der oben erwähnten Klasse FVSS, die als MDI-Container dient, weitere fünf Klassen, die innerhalb der Anwendung als MDI-Subfenster verwendet werden. Das bedeutet, dass während der Laufzeit der Anwendung Instanzen dieser Formularklassen innerhalb des MDI-Containers dargestellt werden.
Als Besonderheit werden die MDI-Subfenster nicht als „normale" Windows-Fenster im MDI-Container dargestellt, sondern werden mit Hilfe der Magic-Library **[2]** als sogenannte Docking-Fenster **[3]** angezeigt. Dies ermöglicht eine bessere Anpassungsfähigkeit der Anwendung Visual SOM Studio an die Gewohnheiten des Anwenders, dadurch, dass beliebige Fenster an beliebige Stellen innerhalb des Anwendungsfensters platziert werden können.

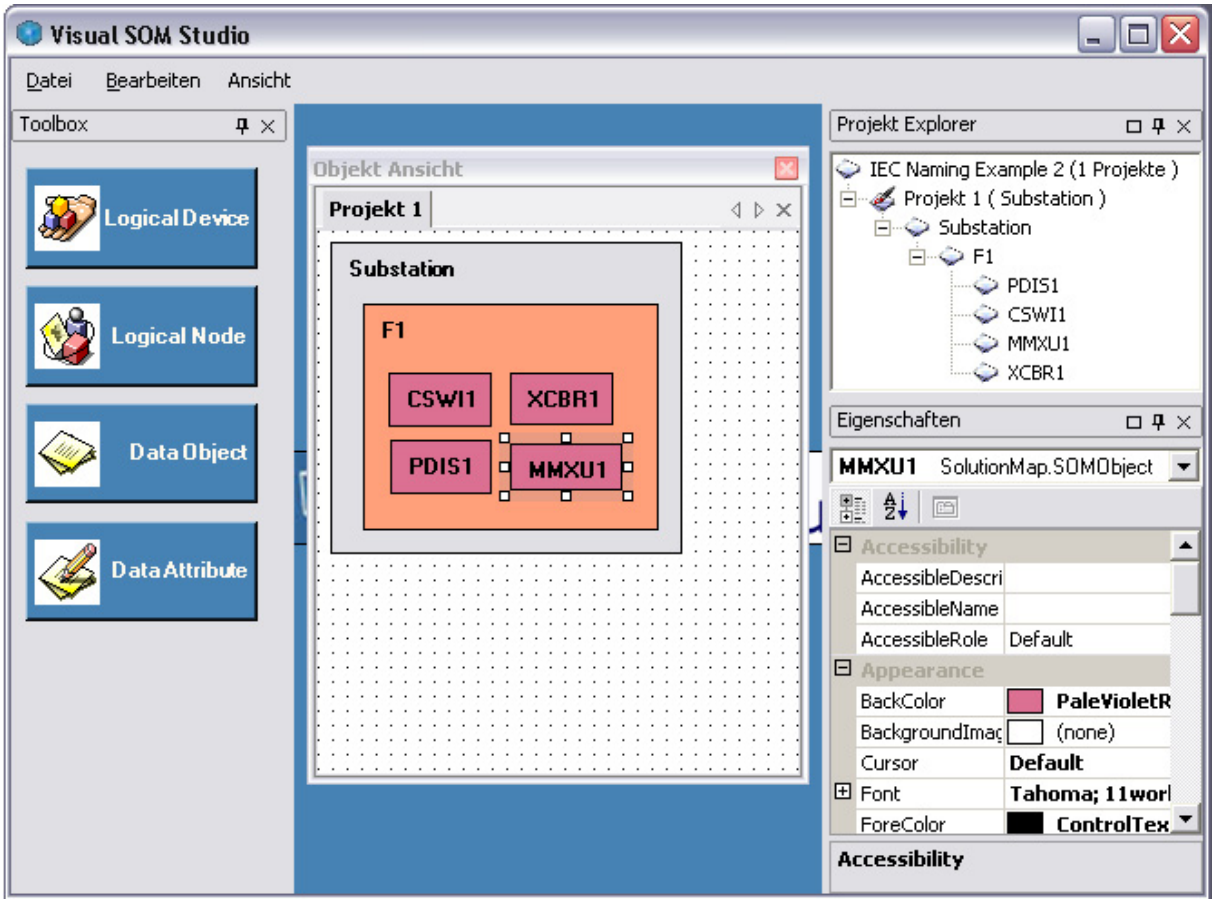

Abbildung 24

Die gewählte Einstellung, sprich die Positionen der einzelnen Fenster werden beim Beenden der Anwendung in der Datei vsscnfg.xml gespeichert, so das bei einer erneuten Programmausführung die eingestellte Konfiguration wieder hergestellt wird.

6.1.2 Das Objektmodell

In dem folgendem UML Diagramm ist das Objektmodell der Komponente VSS graphisch dargestellt.

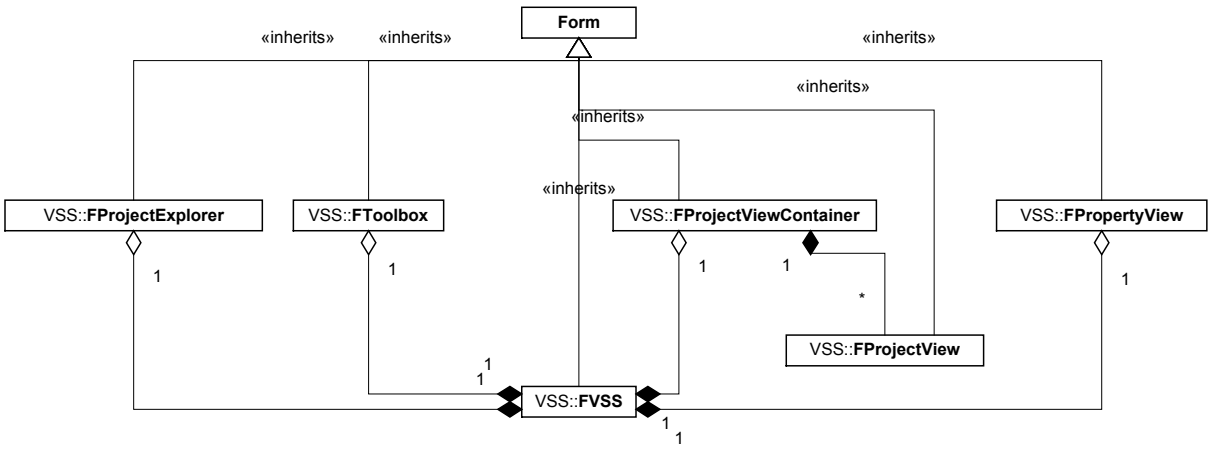

Abbildung 25: Das Objektmodell der Komponente VSS

Das dargestellt Diagramm verdeutlicht, dass alle Klassen der Komponente VSS abgeleitete Klassen der Klasse `System.Windows.Forms.Form` sind. Die Kompositionen zwischen der zentralen Klasse FVSS und den beteiligten Kompositionspartnern `FProjectExplorer`, `FToolbox`, `FProjectViewConatiner` und `FPropertyView` verdeutlichen, dass die beteiligten Kompositionsklassen nur während des Lebenszyklusses einer Instanz der Klasse FVSS existieren. Das bedeutet das die einzelnen Formularinstanzen nur zur Verfügung stehen, wenn ein Instanz der Klasse FVSS existiert. Diese Komposition seitens der Klasse FVSS wird durch jeweilige Referenzverweise auf die entsprechenden Formularobjekte der jeweiligen Klasse gehalten.
Die umgekehrten Referenzverweise der Klassen `FProjectExplorer`, `FToolbox`, `FProjectViewConatiner` und `FPropertyView` auf die Instanz der zentralen Klasse FVSS wird durch die jeweils Aggregation dargestellt.
Das Verhalten der Formularobjekte der Klassen `FProjectExplorer`, `FToolbox`, `FProjectViewConatiner` und `FPropertyView` wird innerhalb der Klassen FVSS mittels eines DockingManagers aus der Magic-Bibliothek erreicht. Der DockingManager verwaltet alle als Dockingfenster anzuzeigenden Formulare mittels Objekten der Klasse Content (Magic-Bibliothek). Jedes Content-Objekt hält dafür eine Referenz auf ein Objekt des als gedockt anzuzeigenden Formularklasse. Die nachfolgend dargestellten Codezeilen verdeutlichen die Referenzbeziehungen zwischen einzelnen Objekten.

```
m_ProjectExplorer = New FProjectExplorer(Me)

dockContentProjectExplorer =
    dockManag.Contents.Add(m_ProjectExplorer,
    "Project Explorer")
```

Zunächst wird das Objekt m_ProjectExporer der Klasse `FProjectExplorer` instanziert. Dabei wird diesem über `Me` ein Verweis auf das Objekt der Klasse `FVSS` übergeben.Durch diese zirkuläre Referenz kann über die Objektvariable m_ProjectExplorer das Objekt der Klasse `FVSS` auf eine Instanz der Klasse `FProjectExplorer` zugreifen. Umgekehrt kann ein Objekt der Klasse `FProjectExplorer` über den an seinen Konstruktor mit übergebenen Parameter auf die Instanz der Klasse `FVSS` zugreifen. Anschließend wird über die Methode `Add()` des Contents-Containers des DockingManagers ein neues Content-Objekt erzeugt. Diesem wird beim Instanzieren ein Verweis auf das entsprechend zu verwaltende Formularobjekt, in diesem Fall das Objekt der Klasse FProjectExplorer, mit übergeben.

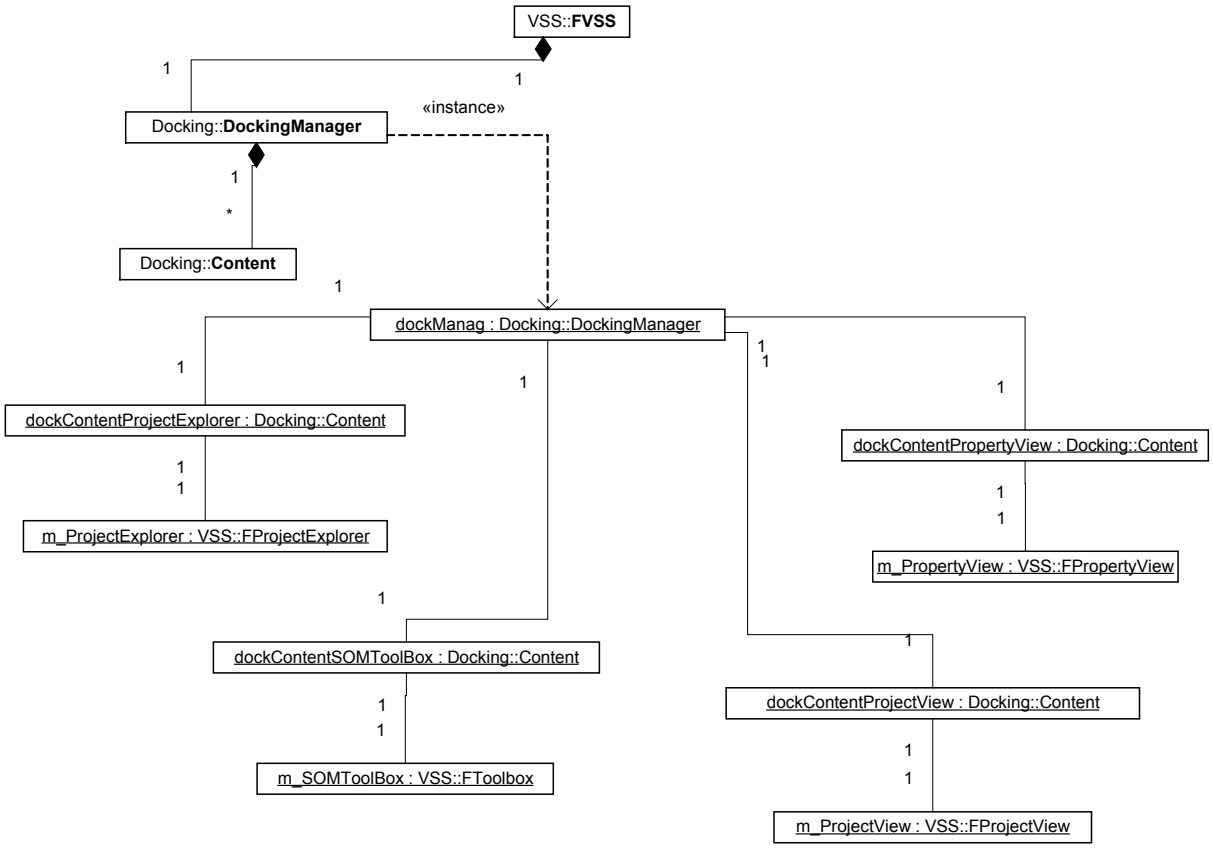

Abbildung 26: Verwaltung der Formularobjekte mit dem Dockingmanager

Die beschriebene Vorgehensweise wird für jedes Objekt der Formularklassen `FProjectExplorer`, `FToolbox`, `FProjectViewConatiner` und `FpropertyView` durchgeführt. Dadurch ergeben sich die in Abbildung 26 gezeigten Objektbeziehungen. Dargestellt sind nicht die zirkulären Referenzen zwischen einem Objekt der Klasse VSS und den daran beteiligten Klassen.

6.1.2.1 Die Klasse FVSS

Eine Instanz der Klasse `FVSS` stellt den grundlegenden Rahmen der Anwendung Visual SOM Studio dar.

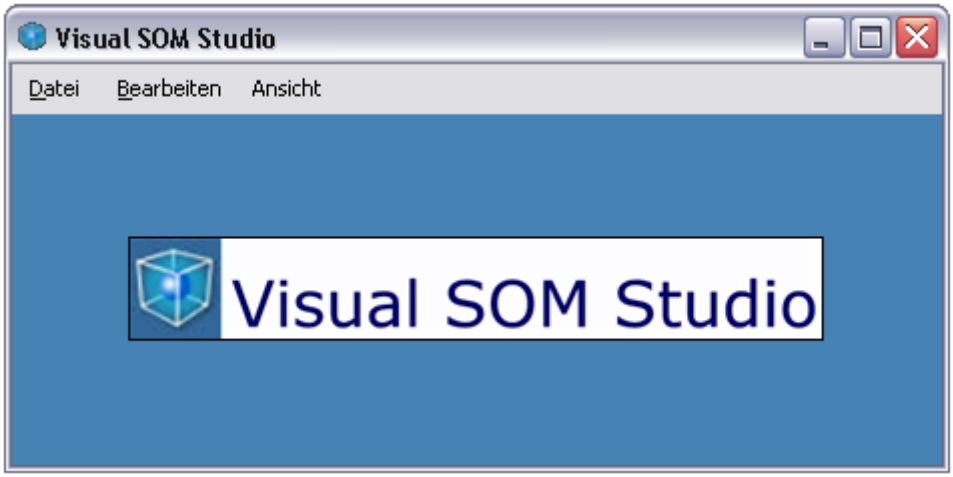

Abbildung 27: Instanz der Klasse `FVSS`

Das Formular der Klasse `FVSS` erscheint beim Start der Anwendung VSS.exe. Die Klasse `FVSS` ist unter den Projekteigenschaften der Komponente VSS als Startup-Objekt angegeben. Um das Verhalten einer Standardformularklasse zu erwerben ist die Klasse `FVSS` von der Klasse `System.Windows.Forms.Form` abgeleitet. Die Instanz der Klasse FVSS hat zur Laufzeit die Aufgabe einen MDI-Container bereit zu stellen, in dem alle anderen Formulare der Anwendung angezeigt werden. Dafür ist die Eigenschaft `IsMidContainer` der Klasse `FVSS` gesetzt (`= True`).
Zur Laufzeit der Anwendung halten alle anderen Formulare Verweise auf die Instanz der Klasse `FVSS`. Aus diesem Grund übernimmt die Instanz der Klasse `FVSS` auch die Funktion der globalen Datenhaltung der Anwendung. Zu diesem Zweck stellt die Klasse `FVSS` über die Eigenschaft `Solution` die aktuell geladenen Projektmappe bereit. Ebenso kann über die Eigenschaft `SelectedProject` auf das aktuelle Projekt und über die Eigenschaft `SelectedSOMObject` auf das aktuelle `SOMObject` zugegriffen werden.
Diese Eigenschaften sind als `Friend` deklariert.

Die Klasse `FVSS` verfügt über eine Menübar, über die der Anwender die Anwendung Visual SOM Studio steuert. Die Menübar enthält die in Abbildung 28 dargestellten Menüeinträge: Datei, Bearbeiten und Ansicht.

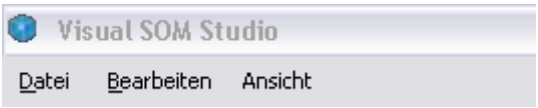

Abbildung 28: Die Menüleiste einer Instanz der Klasse `FVSS`

6.1.2.1.1 Der Menüeintrag Datei

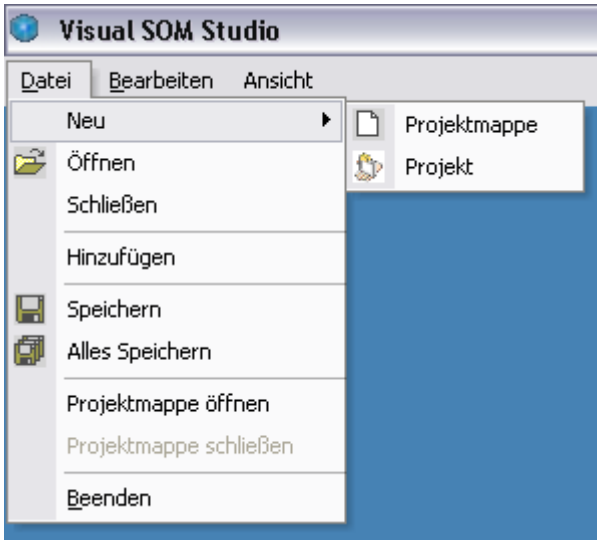

Abbildung 29: Der Menüeintrag Datei

Nicht alle Menüeinträge sind mit Funktionalitäten unterlegt. Lediglich die Funktionalitäten hinterlegt unter den Menüeinträgen <Neu → Projektmappe>, <Neu → Projekt>, <Alles Speichern>, <Projektmappe öffnen>, <Projektmappe schließen> und <Beenden> sind implementiert.

Über den Menüpunkt <Neu → Projektmappe> bzw. <Neu → Projekt> lassen sich entsprechend eine neue Projektmappe anlegen oder zu einer bereits bestehenden Projektmappe ein neues Projekt hinzufügen.

Das Diagramm in Abbildung 30 veranschaulicht den Prozess des Anlegens eines neuen Projektes.

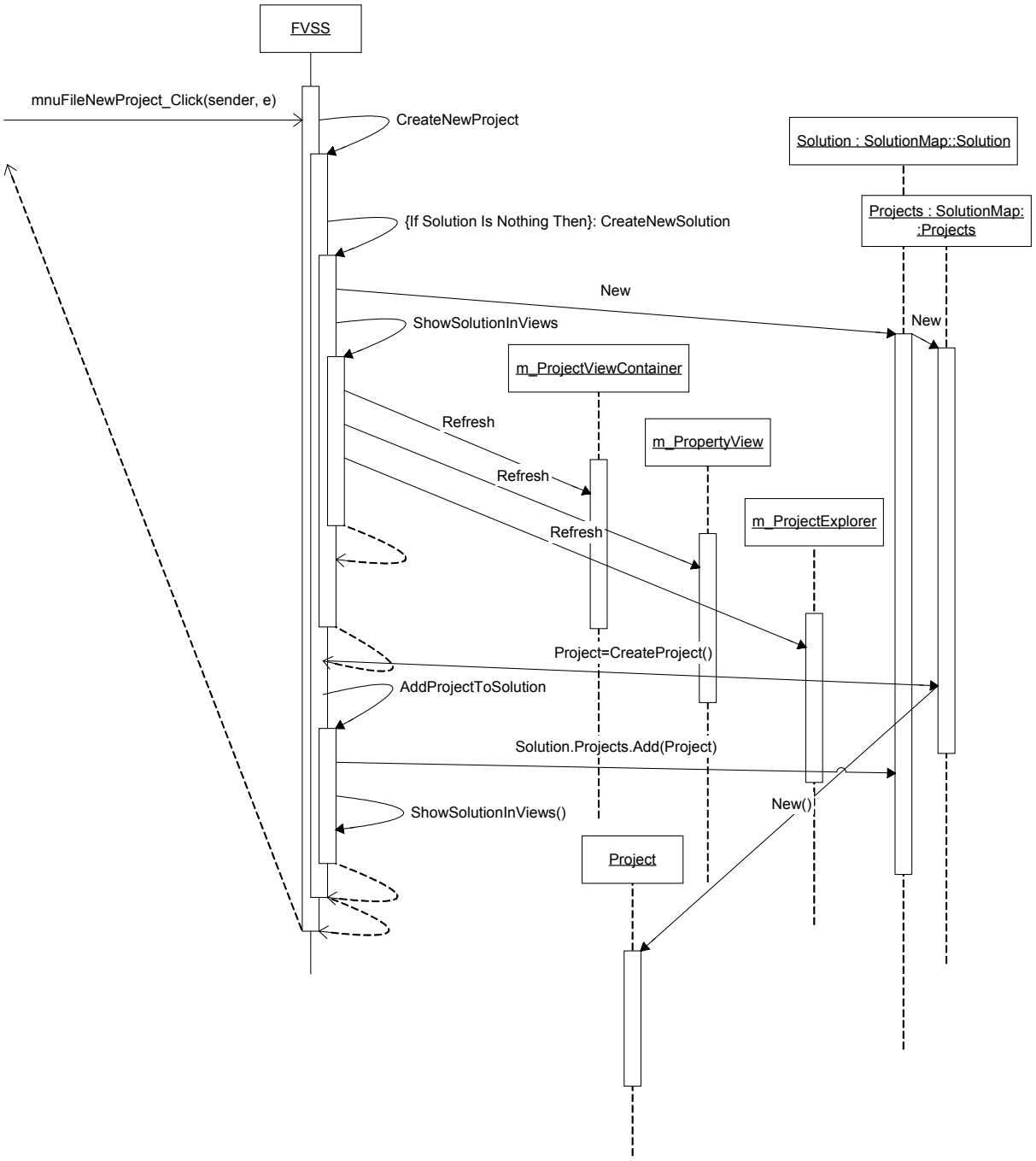

Abbildung 30

Durch die Auswahl des Menüpunktes <Neu → Projekt> wird die Ereignisbehandlungsroutine `mnuFileNewProject_Click` aufgerufen. Innerhalb der Methode `mnuFileNewProject_Click` wird die private Methode `CreateNewProject` aufgerufen. Diese überprüft zunächst, ob eine Projektmappe vorhanden ist. Ist dies nicht der Fall wird die private Methode `CreateNewSolution` aufgerufen. Diese instanziert ein neues Objekt der Klasse `SolutionMap.Solution` und weist diesem ein Default-Name und ein Default-Speicherort zu. Daraufhin wird die private Methode `ShowSolutionInViews` aufgerufen. Die Methode `ShowSolutionInViews` ruft nacheinander die Refresh-Methoden der Formulare `m_PropertyView`, `m_ProjectExplorer` und `m_ProjectViewContainer` auf. Über den Aufruf der Refresh-Methode eines Formulars stellt dieses, seinem Verhalten entsprechend die neue Projektmappe dar. Ist eine Projektmappe bereits vorhanden ruft die Methode `CreateNewProject` an dem Projects-Container des Objektes `Solution` die

Methode CreateProject auf. Der Methode wird der Default-Name für das neu zu instanzierende Objekt der Klasse `Project` übergeben. Als Rückgabewert liefert die Methode `CreateNewProject` eine Verweis auf das neue Objekt der Klasse `Project`. Innerhalb der Ereignisbehandlungsroutine `mnuFileNewProject_Click` wird im nächsten Schritt der Rückgabewert der Methode `CreateNewProject`, also der Objektverweis auf das neue Objekt der Klasse `Project` an die Methode `AddProjectToSolution` übergeben. Hier wird dem Projects-Container des Solution Objekts über die Methode `Add` das neue Objekt der Klasse `Project` hinzugefügt. Anschließend wird die Methode `ShowSolutionInViews` aufgerufen, um die Darstellungen der Projektmappe mit dem neuen Projekt in allen Formularen zu aktualisieren.

Die Menüpunkte <Öffnen>, <Schließen>, <Hinzufügen> und <Speichern> sind nicht mit Funktionalität hinterlegt.

6.1.2.1.2 Der Menüeintrag Bearbeiten

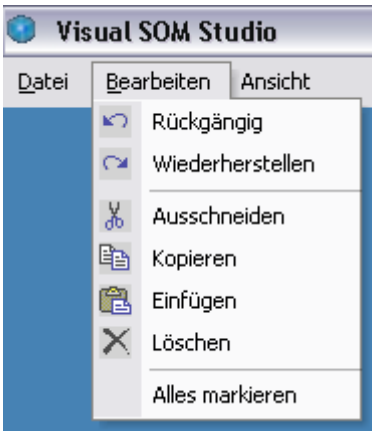

Abbildung 31: Der Menüeintrag Bearbeiten

Die Funktionalitäten, die über das Menü Bearbeiten zugänglich gemacht werden, sind nicht implementiert.
Damit die Anwendung Visual SOM Studio in einem realen Umfeld eingesetzt werden kann müssen diese Funktionen implementiert werden, um die Bedienbarkeit und den Komfort der Anwendung zu steigern und um dem Anwender ein vertrautes Anwendungsverhalten zu bieten.

6.1.2.1.3 Der Menüeintrag Ansicht

Abbildung 32: Der Menüeintrag Ansicht

Über die Menüpunkte des Menueintrages Ansicht, lassen sich die Ansichtsfenster und Werkzeugfenster der Anwendung Visual SOM Studio aufschalten. Da es sich bei allen,

innerhalb von Visual SOM Studio verwendeten Formulare um sogenannte Dockingfenster handelt, wird die Darstellung des Formulars über den Dockingmanager der Klasse `FVSS realisiert`. Der folgende Code verdeutlicht exemplarisch für das Formular der Klasse `FPropertyView`, die notwendigen Schritte zum Anzeigen eines Formulars über den jeweiligen Menüpunkt.

```
Private Sub mnuViewProperty_Click(ByVal sender As Object, ByVal e As
        System.EventArgs) Handles mnuViewProperty.Click

    Me.dockManag.ShowContent(Me.dockContentPropertyView)

End Sub
```

Durch Anwahl des Menüpunktes <Ansicht → Eigenschaften> wird die Ereignisbehandlungsroutine `mnuViewProperty_Click` ausgelöst.
Innerhalb der Ereignisbehandlungsroutine wird die Methode `ShowContent` des Dockingmanagers aufgerufen und dieser ein Objekt der Klasse `Content` übergeben. Das Objekt `dockContentPropertyView` wurde bereits beim Instanzieren des Formulars der Klasse `FVSS` erzeugt. Dabei wurde dem Objekt `dockContentPropertyView` ein Verweis auf ein Objekt der Klasse `FPropertyView` übergeben. Somit wird durch die Ausführung des dargestellten Codes ein Formular der Klasse `FPropertyView` innerhalb des Formulars der Klasse `FVSS` als Dockingfensters angezeigt. Diese Vorgehensweise wird bei allen anderen Formularen, die über das Menü <Ansicht> erreichbar sind, auf gleiche Weise angewendet.

6.1.2.2 Die Klasse FProjectExplorer

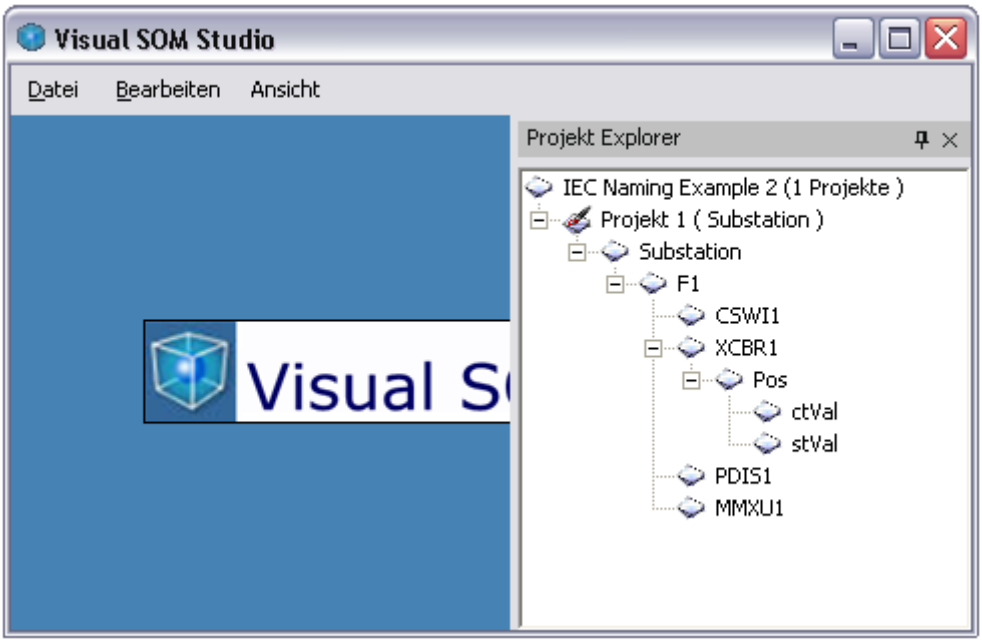

Abbildung 33: Instanz der Klasse FProjektExplorer

Die Instanz der Klasse `FProjectExplorer` hat die Aufgabe eine Projektmappe mit ihrem gesamten Inhalt, bestehend aus enthaltenen Projekten und deren SOMObjekten, in einer hierarchischen Form darzustellen. Dazu verwendet die Klasse `FProjectExpolorer` ein TreeView Steuerelement. Wie in Abbildung 33 zu sehen wird in der obersten Hierarchie des TreeViews die Projektmappe angezeigt. Dafür wird deren Name angezeigt, so wie dahinter in Klammern die Anzahl der enthaltenen Projekte. Unterhalb der Projektmappe werden die in der Projektmappe enthaltenen Projekte aufgelistet. Dazu wird der

Projektname angezeigt sowie in Klammern dahinter der Name des verwendeten SOMServers. Unterhalb des Projektknoten werden die SOMObjecte der Typen Server, LogicalDevice, LogicalNode, DataObject und DataAttribute in hierarchischer Weise dargestellt.
Zugriff auf die darzustellenden Daten erhält die Klasse `FProjectExplorer` über die Eigenschaft Solution der Klasse FVSS, auf die sie über die private Eigenschaft `m_parent` einen Verweis hält. Der Objektverweis auf die Instanz der Klasse `FVSS` wird der Klasse `FProjectExplorer` über den Konstruktor

```
Public Sub New(ByRef Parent As FVSS)
```

übergeben.

Über die Methode `Refresh` der Klasse `FProjectExplorer` wird die aktuell geladene Projektmappe im TreeView des Objektes der Klasse `FProjectExplorer` angezeigt. Die Methode Refresh ruft die private Methode `ViewSolution` auf.
Innerhalb der Methode `ViewSolution` wird zunächst der gesamte Inhalt des TreeViews mit `Me.TreeView.Nodes.Clear()` gelöscht. Anschließend wird überprüft, ob eine Projektmappe vorhanden ist. Ist dies der Fall wird dem TreeView ein neuer Knoten hinzugefügt und dieser mittels der privaten Funktion `CaptionedSolutionNode`, wie in Abbildung 33 dargestellt, beschriftet. Über die Eigenschaft `Tag` des neu angelegten Knotens wird ein Verweis auf die aktuelle Projekmappe gesetzt.
Damit wird eine Beziehung zwischen TreeView-Knoten und dem darzustellenden Objekt, in dem Fall der Projektmappe hergestellt. Diese Methode erleichtert den späteren Zugriff auf das durch den TreeView-Knoten repräsentierte Objekt, wenn z.B. lediglich ein Verweis auf einen TreeView-Knoten vorliegt. Diese Methode wird bei allen TreeView-Knoten angewendet. Somit kann über die Eigenschaft Tag des jeweiligen TreeView-Knotens auf das entsprechend dargestellte Objekt zugegriffen werden.
Nachdem die Tag-Eigenschaft des neuen TreeView-Knotens gesetzt wurde, wird durch den gesamten Projects-Container der Projektmappe durchiteriert und zu jedem vorhandenen Projekt ein neuer Knoten dem TreeView hinzugefügt, der mit der privaten Methode `CaptionedProjectNode` beschriftet wird. Während der Iterration wird die Methode `FillTreeWithSOMObjects` aufgerufen. Dieser Methode wird der aktuelle TreeView-Knoten, sowie der `SOMObjects`-Container des aktuellen Projekts als Parameter mit übergeben.

```
Private Sub FillTreeWithSOMObjects(ByRef Node As TreeNode, ByVal SOMObjects
                                                    As SolutionMap.SOMObjects)

    Dim SOMObject As SolutionMap.SOMObject
    Dim tmpNode As TreeNode
    If Not SOMObjects Is Nothing Then
        For Each SOMObject In SOMObjects.Values
            tmpNode = Node.Nodes.Add(SOMObject.Name)
            tmpNode.Tag = SOMObject
            FillTreeWithSOMObjects(tmpNode, SOMObject.SOMObjects)
        Next
    End If
End Sub
```

Die Methode `FillTreeWithSOMObjects` durchläuft den gesamten `SOMObjects`-Container und fügt dem aktuellen TreeView-Knoten je enthaltenem `SOMObject` einen neuen Knoten hinzu. Die Methode `FillTreeWithSOMObjects` wird rekursiv aufgerufen und ihr der aktuelle Knoten, sowie der `SOMObjects`-Container des aktuellen `SOMObject` mit übergeben.
Dadurch entsteht die in Abbildung 33 dargestellte hierarchische Abbildung der Projektmappe.

6.1.2.3 Die Klasse FPropertyView

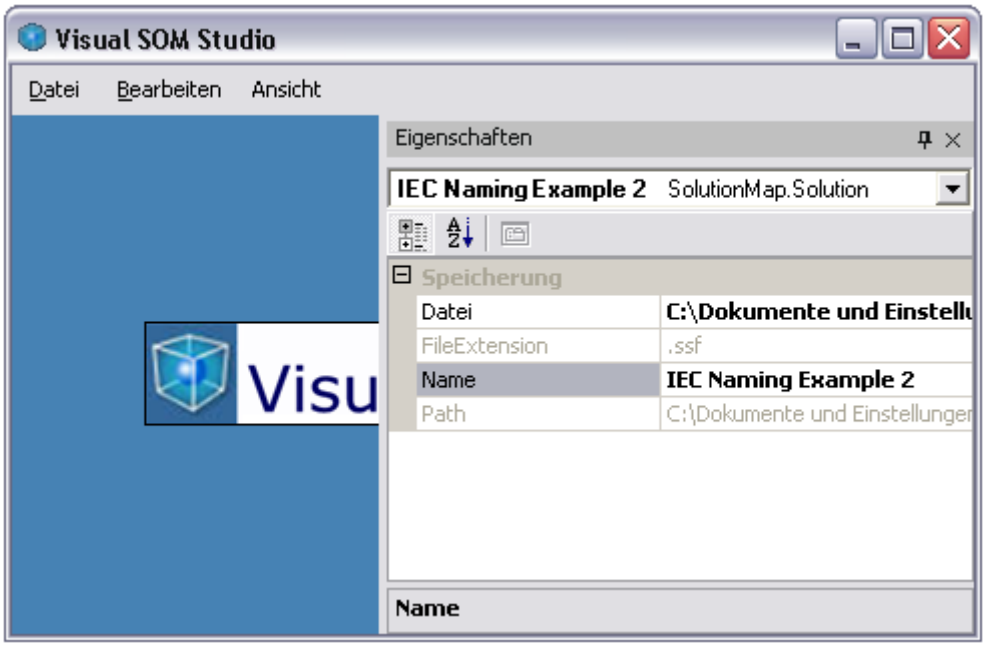

Abbildung 34: Instanz der Klasse FPropertyView

Die Instanz der Klasse `FPropertyView` hat die Aufgabe, Eigenschaften eines ausgewählten Objektes darzustellen und, soweit diese Eigenschaften nicht schreibgeschützt sind, editierbar zu machen. Schreibgeschützte Eigenschaften werden durch eine gräulich, transparente Darstellung gekennzeichnet. Um die Eigenschaften eines beliebigen Objektes wie beschrieben darzustellen, bedient sich die Klasse `FPropertyView` des Steuerelements PropertyGrid des .NET-Frameworks. Eine Instanz der Klasse `FPropertyView` stellt innerhalb der Anwendung Visual SOM Studio das Eigenschaftsfenster dar. Neben dem Anzeigen und Editieren von Eigenschaften eines Objektes zeigt die Instanz der Klasse `FPropertyView` im oberen Bereich den Namen, sowie den Typ des im PropertyGrid dargestellten Objektes an. Über die öffentliche Methode `SelectedObject` wird der Instanz der Klasse `FPropertyView` ein Verweis auf das anzuzeigende Objekt übergeben.
Die Methode `SelectedObject` ruft die private Methode `ShowObj` auf und übergibt dieser das neu anzuzeigende Objekt. Die private Methode `ShowObj` setzt die Eigenschaft `SelectedObject` des PropertyGrid-Steuerelements auf das neue Objekt. Daraufhin werden die Eigenschaften des Objektes im PropertyGrid-Steuerelement sichtbar. Anschließend wird über die Methode `GetObjKey` ein selbstdefinierte Objektschlüssel erstellt und dieser in der Combobox der Klasse `ObjectNameCombobox.Combobox` über die Eigenschaft `Text` dargestellt. Der über die Methode `GetObjKey` ermittelte Objektschlüssel besteht aus einer Zeichenkette, die wie folgt zusammengesetzt ist: Objektname + ASCII-Zeichen Nr. 255 + Objekttyp.

```
key = Obj.GetType.InvokeMember(KeyProperty, BindingFlags.GetProperty, _
                    Nothing, Obj, New Object() {})

key += Chr(255) + Obj.GetType.ToString
```

Zunächst ermittelt die Methode `GetObjKey` mittels des .NET-Reflectionframeworks den Wert der Eigenschaft Name (`KeyProperty = "Name"`) des Objektes, sofern diese Eigenschaft existiert. Daran wird das ASCII-Zeichen Nr. 255 angehängt, gefolgt vom Typnamen des Objektes. Dadurch, dass der Objektschlüssel zusammengesetzt ist aus

dem Namen und dem Typen des Objektes getrennt durch das ASCII-Zeichen Nr.255 und die Combobox der Klasse `ObjectNameCombobox.Combobox` das ASCII-Zeichen als Trennzeichen interpretiert, wird der Objektschlüssel in der Combobox wie in Abbildung 34 dargestellt. Der Objektname erscheint in fetten Lettern.
Unter der Zuhilfenahme der Combobox können im Eigenschaftsfenster mehrere Objekte angezeigt werden. Welches Objekt aktuell angezeigt werden soll, kann dabei über die Combobox gewählt werden. Die Combobox wird über die öffentliche Methode `AddObject` der Klasse `FPropertyView` gefüllt. Ebenso können über die öffentliche Methode `RemoveObject` in der Combobox dargestellte Objekte entfernt werden. So wird z.B. innerhalb der Anwendung Visual SOM Studio das Eigenschaftsfenster so benutzt, dass wenn die Eigenschaften einer Projektmappe angezeigt werden, gleichzeitig in der Combobox alle in der Projektmappe enthaltenen Projekte ausgewählt werden können.

6.1.2.4 Die Klasse FToolbox

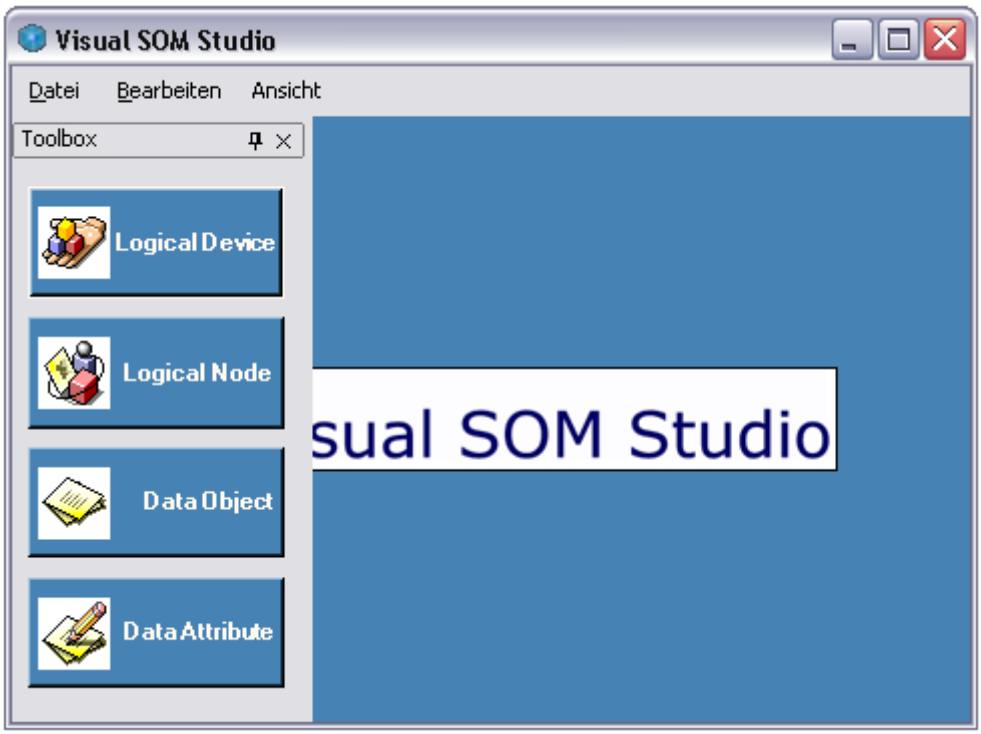

Abbildung 35: Instanz der Klasse FToolbox

Die Instanz der Klasse FToolbox stellt zur Laufzeit eine Werkzeugsammlung dar, über die bestimmte Objekte neu angelegt werden können. Diese Werzeugsammlung enthält vier Schaltflächen über die einem bestehendem SOMObject weitere SOMObjecte hinzugefügt werden können. Durch Auswahl der jeweiligen Schaltfläche wird der SOMType des neuen SOMObjects festgelegt.
Der folgende Codeausschnitt zeigt den Prozess des Hinzufügens eines neuen SOMObjects über die Schaltfläche LogicalDevice.

```
Private Sub btnAddLD_Click(ByVal sender As System.Object, ByVal e As
                           System.EventArgs) Handles btnAddLD.Click

    If Not m_parent.SelectedSOMObject Is Nothing Then
        CreateAndAddSOMObject(SolutionMap.SOMType.enType.stILogicalDevice,
                              m_parent.SelectedSOMObject, "LD")

    End If
```

```
End Sub
```

Durch Betätigen der Schaltfläche LogicalDevice wird die Ereignisbehandlungsroutine `btnAddLD_Click` aufgerufen. Diese überprüft zunächst, über den Verweis `m_parent` auf die Instanz der Klasse `FVSS`, mittels deren Eigenschaft `SelectedSOMObject`, ob ein SOMObject ausgewählt wurde. Ist dies der Fall, wird die private Methode `CreateAndAddSOMObejct` aufgerufen. Diese Methode empfängt drei Parameter. Über den ersten Parameter wird der `SOMType` des neuen SOMObjects angegeben. Über den zweiten Parameter wird das ausgewählte SOMObject, dem das neue SOMObject hinzugefügt werden soll, übergeben. Der dritte Parameter enthält den Default-Namen des neuen SOMObjects.
Durch diese Funktionssignatur kann die Methode `CreateAndAddSOMObject` von den Ereignisbehandlungsroutinen der anderen Schaltflächen aufgerufen werden. Dort müssen lediglich die Parameter angepasst werden.

```
Private Sub CreateAndAddSOMObject(ByVal SOMType As
                  SolutionMap.SOMType.enType, ByVal SelectedSOMObject As
                  SolutionMap.SOMObject, ByVal Name As String)

    Dim SOMColors As SolutionMap.SOMObjectColors = New
                                        SolutionMap.SOMObjectColors()

    Dim SOMObject As SolutionMap.SOMObject = New
                                        SolutionMap.SOMObject(SOMType)

    SOMObject.BackColor = SOMColors.Item(SOMType).Color
    SOMObject.Size = New Size(40, 30)
    SOMObject.Location = New Point(10, 30)
    SOMObject.Visible = True

    SOMObject.Name = Name & SelectedSOMObject.SOMObjects.Count + 1
    SelectedSOMObject.SOMObjects.Add(SOMObject)

    m_parent.ShowSolutionInViews()

End Sub
```

Die private Methode `CreateAndAddSOMObject` instanziert zunächst ein Objekt der Klasse `SolutionMap.SOMObjectColors`. Mittels des SOMTypen und des Objekts der Klasse `SolutionMap.SOMObjectColors` lässt sich eine, dem SOMTypen entsprechend, vordefinierte Farbe auswählen (`SOMColors.Item(SOMType).Color`).
Die Methode `CreateAndAddSOMObject` instanziert ein neues Objekt der Klasse `SOMObject` und legt Größe, sowie Position und Objektname fest. Anschließend wird das neue `SOMObject` dem `SOMObjects`-Container des ausgewählten SOMObjects über die Methode `Add` hinzugefügt. Im letzten Schritt ruft die Methode `CreateAndAddSOMObject` die Methode `ShowSolutionInViews` der Instanz der Klasse `FVSS` auf. Damit wird die aktuelle Konfiguration des Objektmodells in allen Formularen der Anwendung Visual SOM Studio aktualisiert.

6.1.2.5 Die Klasse FProjectView

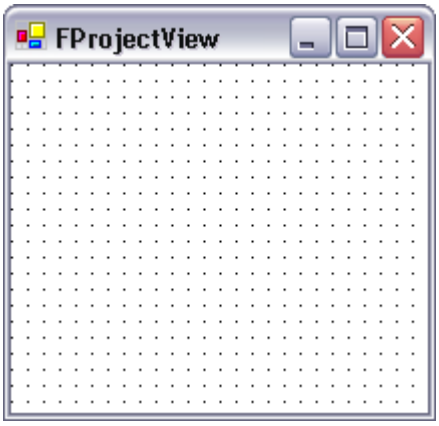

Abbildung 36: Instanz der Klasse FProjectView

Eine Instanz der Klasse `FProjectView` stellt die graphische Objektansicht eines Projekts mit den darin enthaltenen SOMObjecten dar. Zudem hat sie die Aufgabe des Objektkonfigurationseditors, mit dem Objekte der Klasse `SOMObject` in Größe, Position und Hierarchie modelliert werden. Die gesamte Objektkonfiguration eines Projektes wird mit dem Steuerelement der Klasse `GODesigner.UIObject` dargestellt (siehe Kapitel Die Komponente GODesigner). Das `UIObject` ist über die Eigenschaft Dock `(Me.UiObject.Dock = System.Windows.Forms.DockStyle.Fill )` an das Formularfenster der Klasse `FProjectView` angebunden. Dadurch füllt das Steuerelement der Klasse `UIObject` das gesamte Fenster aus. Zudem ist das Steuerelement durch ein Hintergrundbild mit einem Raster hinterlegt.

Ein Objekt der Klasse `FProjectView` hält über die private Eigenschaft `m_Project` einen Verweis auf das darzustellende Projekt. Diese Eigenschaft wird beim Instanzieren über den Konstruktor

```
Public Sub New(ByVal Project As SolutionMap.Project, ByVal Parent As FVSS)
```

gesetzt. Über die Methode `Refresh` der Klasse `FProjectView` wird die Objektkonfiguration des Projektes, auf das das Objekt der Klasse `FProjectView` verweist, angezeigt.

```
Public Overrides Sub Refresh()
    Dim SOMObject As SolutionMap.SOMObject
    For Each SOMObject In m_Project.SOMObjects.Values
        If Not Me.UiObject.Controls.Contains(SOMObject) Then
            Me.UiObject.Controls.Add(SOMObject)
        End If
        AddHandlerAndShow(SOMObject)
    Next
End Sub
```

Innerhalb der Methode `Refresh` wird der gesamte `SOMObjects-Container` des m_Project-Objekts durchlaufen. Jedes `SOMObject` des `SOMObjects-Containers` wird dem `Controls-Containers` des `UIObject`, sofern es in diesem noch nicht vorhanden ist, hinzugefügt. Anschließend wird die private Methode `AddHandlerAndShow` mit dem aktuellen `SOMObejct`-Objekt als Parameter aufgerufen. Die Methode `AddHandlerAndShow` registriert die Ereignisbehandlungsroutine

```
Private Sub SOMObjects_Selected(ByVal sender As GODesigner.UIObject)
```

für das Ereignis `Selected` des aktuellen SOMObjectes.
Die Ereignisbehandlungsroutine wird ausgeführt, wenn ein SOMObject mit der Maus selektiert wird. Somit wird innerhalb der Ereignisbehandlungsroutine die Aktualisierung des Eigenschaftsfensters vorgenommen, so dass immer das selektierte Objekt im Eigenschaftsfenster angezeigt wird.
Anschließend wird die Methode `Refresh` des `SOMObjects` aufgerufen.

```
Private Sub AddHandlerAndShow(ByVal SOMObject As SolutionMap.SOMObject)
    Dim tmp As SolutionMap.SOMObject
    AddHandler SOMObject.Selected, AddressOf SOMObjects_Selected
    SOMObject.Refresh()
    SOMObject.Visible = True

    For Each tmp In SOMObject.SOMObjects.Values
        AddHandlerAndShow(tmp)
    Next
End Sub
```

Durch den Aufruf der Methode `Refresh` am SOMObject werden alle, in dessen `SOMObjects`-Container enthaltenen SOMObjecte dargestellt. Die in dem SOMObject enthaltenen SOMObjecte werden in dessen graphischen Rahmen dargestellt.
Anschließend wird der gesamte `SOMObjects`-Container des aktuellen `SOMObject` durchiteriert und rekursiv die Methode `AddHandlerAndShow` mit jeweils aktuellem `SOMObject` aufgerufen. Durch den rekursiven Aufruf der Methode `AddHandlerAndShow` wird die Anzeige aller in dem Projekt enthaltener SOMObjecte erreicht.

6.1.2.6 Die Klasse FProjectViewContainer

Die Klasse `FProjectViewContainer` hat die Aufgabe einen visuellen Container für Objekte der Klasse `FProjectView` zur Verfügung zustellen.

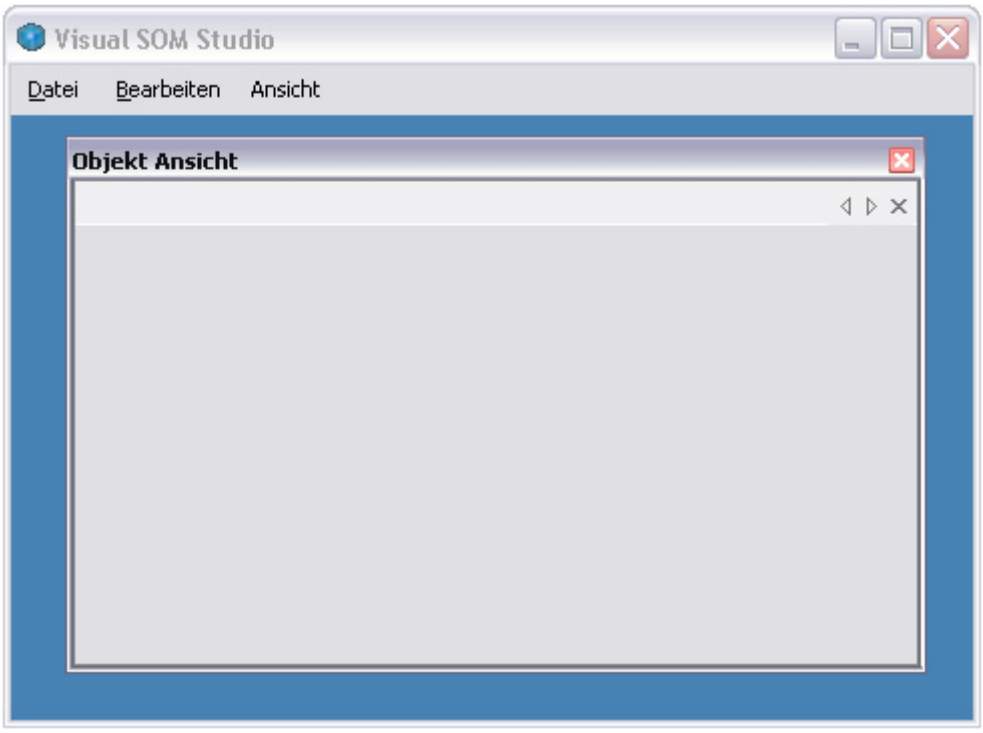

Abbildung 37: Instanz der Klasse FProjectViewContainer

Da mit der Anwendung Visual SOM Studio beliebig viele Projekte innerhalb einer Projektmappe verwaltet werden, die Klasse `FProjectView` jedoch nur für die Darstellung eines Projektes ausgelegt ist, wird die Klasse `FProjectViewContainer` verwendet.
Die Darstellung der einzelnen Projekte erfolgt mit Hilfe von Registerkarten.
Dafür bedient sich die Klasse `FProjectViewContainer` des Steuerelements `Crownwood.Magic.Controls.TabControl` **[2]**. Das Steuerelement `TabControl` verwaltet über die Eigenschaft `TabPages` die einzelnen Dialogseiten. Die Dialogseiten sind Objekte der Klasse `TabPage`, die einen Verweis auf ein Objekt der Klasse `FProjectView` halten. Somit werden die Instanzen der Klasse `FProjectView` als Dialogseiten innerhalb der Registerkarte dargestellt.
Über die Methode `Refresh` wird der Inhalt einer Projektmappe angezeigt. Für jedes in der Projektmappe enthaltenen Projekt wird eine Registerkarte der Klasse `TabPage` angelegt. Zunächst wird überprüft, ob eine Projektmappe vorhanden ist. Ist dies nicht der Fall wird der gesamte Inhalt des Registerkartensteuerelement gelöscht. Ist eine Projektmappe vorhanden wird durch den gesamten `Projects`-Container der Projektmappe durchiterriert. Innerhalb der Iterrartionsschleife wird zunächst überprüft, ob das aktuelle Projekt eventuell bereits in dem TabPages-Container des TabControl-Steuerelements enthalten ist. Ist dies nicht der Fall wird ein Objekt der Klasse `FProjectView` instanziert.

```
Dim ProjectView As New FProjectView(Project, m_Parent)
TabPage = New Crownwood.Magic.Controls.TabPage(Project.Name, ProjectView)
TabControl.TabPages.Add(TabPage)
```

Dem zu instanzierenden Objekt der Klasse `FProjectView` wird über dessen Konstruktor ein Verweis auf das aktuelle Projekt übergeben. Danach wird ein Objekt der Klasse `TabPage` instanziert. Diesem wird im Konstruktor ein Verweis auf das zuvor neu instanziert Objekt der Klasse `FProjectView` übergeben. Anschließend wird das neue Objekt der Klasse `TabPage` dem `TabPages`-Container des Steuerelementes `TabControl` über die Methode `Add` hinzugefügt.

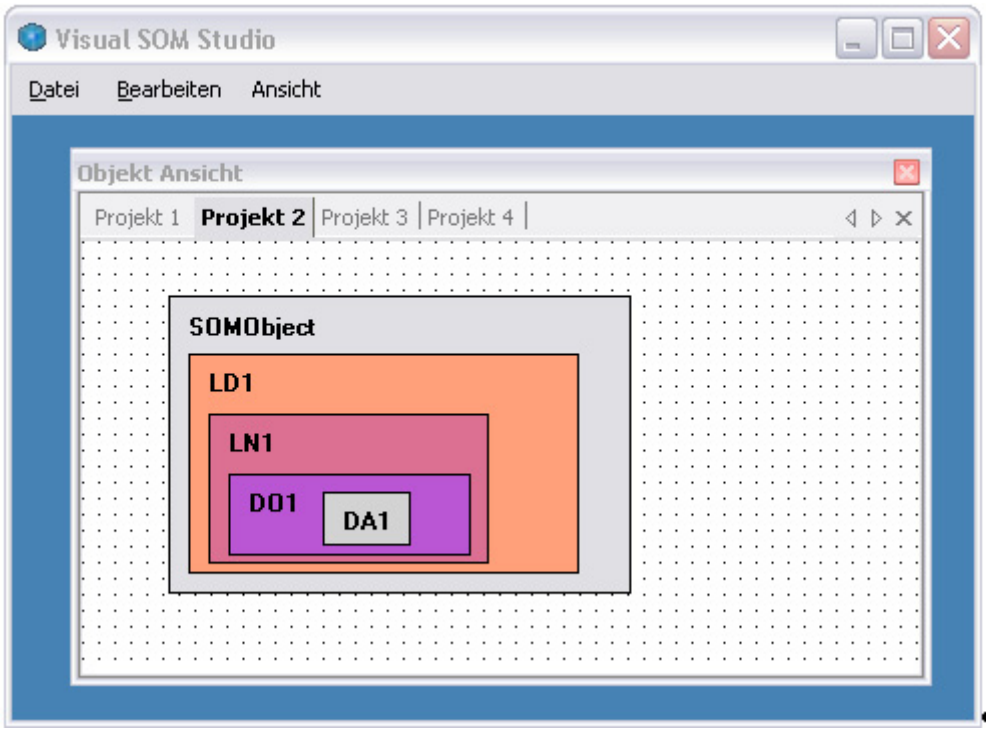

Abbildung 38: mehrere Projekte, dargestellt über Registerkarten

Somit werden Objekte der Formularklasse `FProjectView` als Dialogseite innerhalb des Registerkartensteuerelement `TabControl` dargestellt.

6.2 Die Komponente SolutionMap

6.2.1 Aufgaben der Komponente SolutionMap

Die Komponente SolutionMap stellt Klassen bereit mit deren Hilfe eine projektbezogene Modellierung von Unterstationen der elektrischen Energieversorgung, basierend auf dem Schnittstellenmodell SOM, realisiert werden kann. Die Komponente übernimmt in ihrer Funktion innerhalb der gesamten Anwendung Visual SOM Studio (VSS) die Aufgaben der Geschäftslogik.

Die zentralen Klassen der Komponente sind in einem hierarchischen Objektmodell angeordnet. Das Wurzelelement der Klassenhierarchie ist die Klasse `SolutionMap.Solution`. Objekte der Klasse `Solution` verfügen über einen Container `Projects` über den beliebige Objekte der Klasse `Project` verwaltet werden können. Die Klasse `Project` verfügt über einen Container `SOMObjects` zur Aufnahme von Objekten der Klasse `SOMObject`. Die Klasse `SOMObject` selber enthält ebenfalls einen Container `SOMObjects` der beliebige Objekte der Klasse `SOMObject` enthalten kann. Die hierarchische Struktur eines Project-Objektgraphen ist somit nach unten hin beliebig erweiterbar. Jedoch werden im Kontext des SOM lediglich fünf Hierarchieebenen gefordert um die einzelnen Grundtypen, spezifiziert in IEC 61850 Teil 7 und durch die abstrakten Klassen des SOM's implementiert (`IServer, IlogicalDevice, IlogicalNode, IDataObject, IDataAttribute`), zu modellieren.
Es wird deutlich, dass die hierarchische Struktur der Komponenten einer zu modellierenden Unterstation nicht direkt durch Objekte, die Schnittstellen des SOM implementieren, realisiert wird, sondern über die Containerklasse `SOMObjects` sowie die Klasse `SOMObject`. Objekte der Klasse `SOMObject` haben dabei die Aufgabe zwischen graphischer Anwendungsebene und SOM-Objekten zu vermitteln. Sollten Objekte, die SOM-Schnittstellen implementieren, direkt graphisch darstellbar und modellierbar sein, so müssten Implementierung oder sogar Spezifizierungen die graphische Darstellung eines Objekts betreffend, bereits im SOM enthalten sein. Dies ist nicht der Fall, da dies auch nicht Aufgabe von SOM ist. Die Verbindung zwischen graphischer Anwendungsebene und SOM übernehmen Objekte der Klasse `SOMObject`. Die Klasse `SOMObject` ist abgeleitet von der Klasse `GODesigner.UIObject` und stellt somit von sich aus eine graphische Oberfläche bereit. `UIObject` stellt dabei einen graphischen Container breit in dem beliebige andere Objekte abgeleitet von `System.Windows.Forms.Control` in Form und Position verändert werden können. Neben der graphischen Oberfläche besitzt jedes Objekt der Klasse `SOMObject` die Eigenschaft `SOMObject`

```
Public Property SOMObject() As Object
```

Über diese Eigenschaft kann jedes Objekt der Klasse SOMObject einen Verweis auf ein anderes Objekt halten, vornehmlich solche, die Schnittstellen des SOM's implementieren. Somit kann eine hierarchische Struktur auf Grundlage einer graphischen Oberfläche modelliert werden in der sich bereits zur Modellierungsphase konkrete Objektinstanzen von Klassen, die SOM-Schnittstellen implementieren, befinden, ohne dass das SOM über graphische Darstellungsspezifikationen verfügen muss.

Die nachfolgenden Abbildungen verdeutlichen den beschriebenen Sachverhalt.
In Abbildung 39 ist ein exemplarischer Objektgraph modelliert. Die oberste Ebene stellt ein Objekt der Klasse `Solution` dar. In der Ebene darunter wird ein Objekt der Klasse `Project` dargestellt. Alle weiteren dargestellten Objekte sind Objekte der Klasse `SOMObject`.

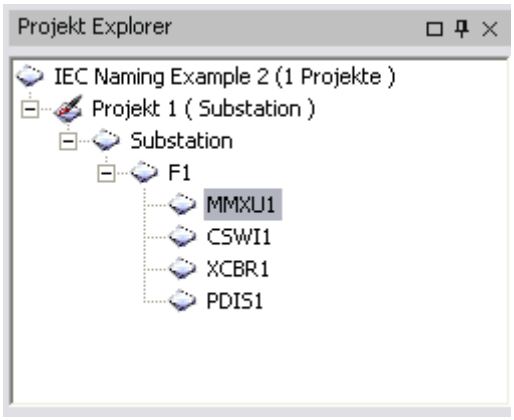

Abbildung 39: Objektgraph dargestellt in einem TreeView

In Abbildung 40 sind die Objekte der Klasse `SOMObject` in graphischer Darstellung gezeigt.
Zu sehen ist, wie Objekte der Klasse `SOMObject` andere Objekte aufnehmen können.

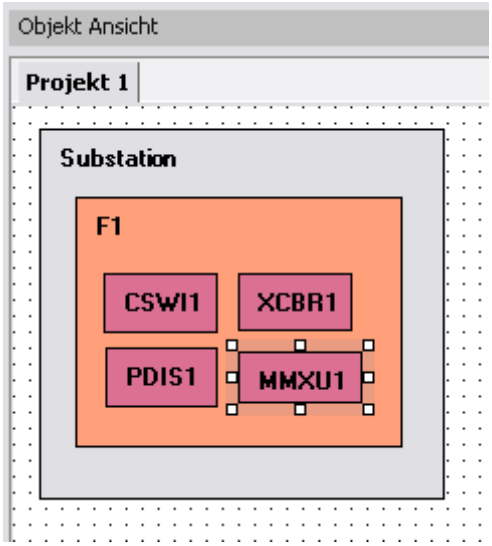

Abbildung 40: graphische Darstellung der SOMObjekte

In Abbildung 41 ist ein Ausschnitt von Eigenschaften des markierten SOMObject's MMXU1 dargestellt. Im dargestellten Bereich Misc des Eigenschaftsfensters sind unter den Eigenschaften AssemblyName und ClassName die Informationen zum Instanzieren des SOM-Objekts enthalten, somit wird der eigentliche Verweis auf das Objekt über die Eigenschaft SOMObject gehalten.

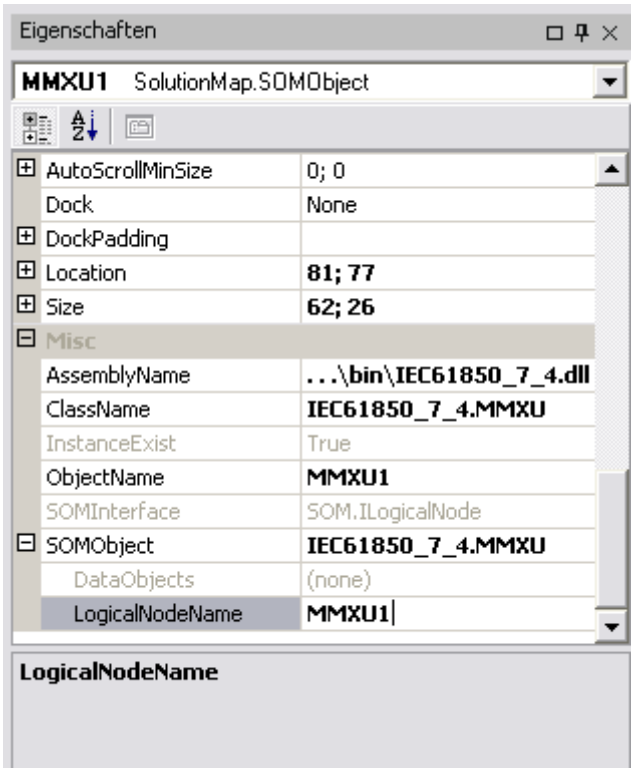

Abbildung 41: Darstellung von Eigenschaften eines SOMObjekts

Um einen konkreten SOM-Objektgraph zu instanzieren ist ein weiterer Schritt nötig. In diesem muss die Information der Hierarchie, enthalten in dem Project-Objektgraph, und die einzelnen Objekte des SOM's, referenziert durch ein jeweiliges SOMObject verknüpft werden. Dieses leistet die Anwendung Visual SOM Studio in ihrer vorliegenden Version nicht.

Alle Klassen der beschriebenen Klassenhierarchie sind serialisierbar. Durch die Serialisierbarkeit jeder einzelnen Klasse wird die Persistenz eines jeden Objektgraph erreicht. Neu modellierte Objektgraphen können gespeichert werden oder bereits bestehende Objektgraphen geladen und geändert werden.
Somit wird ermöglicht, dass die Modellierung einer Unterstation gespeichert und zu späterer Zeit weiterbearbeitet werden kann.
Die Geschäftslogikkomponente SolutionMap stellt somit direkt die Verbindung zur Datenhaltungsschicht her.

6.2.2 Das Objektmodell

Die Klassen der Komponente SolutionMap, die zur Erstellung von Objektkonfigurationen notwendig sind, sind in einer hierarchischen Struktur implementiert.

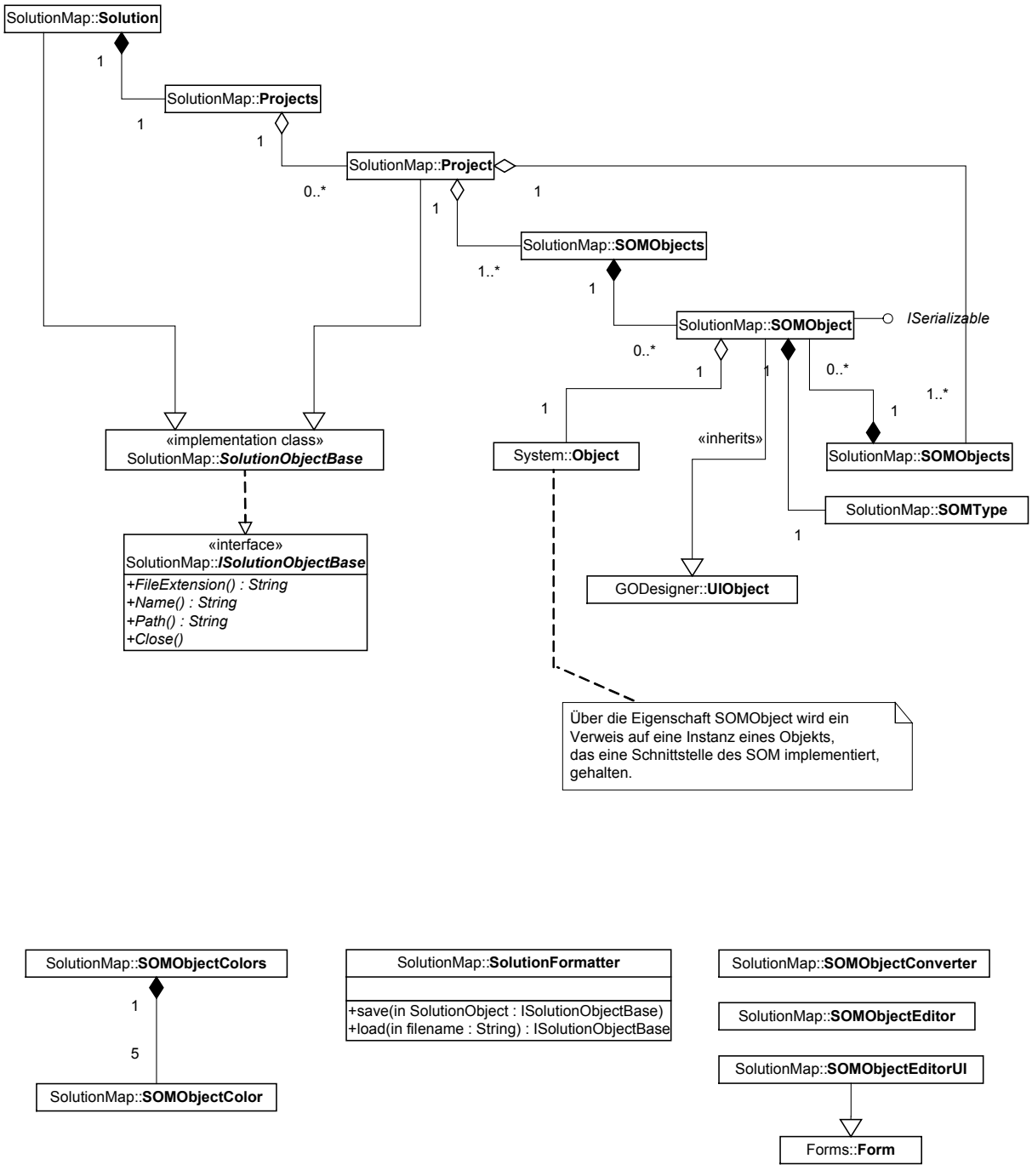

Abbildung 42: Objektmodell der Komponente SolutionMap

6.2.2.1 Die Klassen Solution, SolutionObjectBase und die Schnittstelle ISolutionObjectBase

Das Element der höchsten Hierarchiestufe, das sogenannte Wurzelelement, ist die Klasse `Solution`. Sie enthält einen dynamischen Container `Projects` zur Aufnahme von Objekten der Klasse `Project`. Sowohl die Klasse Solution als auch die Klasse Project ist abgeleitet von der Implementierungsklasse `SolutionObjectBase`, welche die Schnittstelle `ISolutionObjectBase` implementiert.

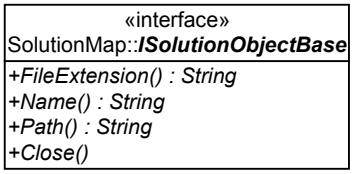

Abbildung 43: Die Schnittstelle `ISolutionObjectBase`

Dadurch werden die beiden Klassen mit Eigenschaften ausgestattet, mittels derer sie Informationen über den eigenen Speicherort auf einem Dateisystem speichern können. Der Speicherort setzt sich zusammen aus der Pfadangabe, dem Dateinamen und der Dateiendung.

6.2.2.2 Die Klasse Project

Jedes Objekt der Klasse `Project` enthält einen Container `SOMObjects` der Objekte der Klasse `SOMObject` aufnimmt. Im Kontext von Visual SOM Studio hat ein Projekt die Aufgabe einen Server (SOM Server) bereitzustellen. Da zu jeder Modellierung mindestens ein Server benötigt wird, wird standardmäßig zu jedem Projekt ein Server angelegt. Aus diesem Grund wird beim Instanzieren eines Objekts der Klasse `Project` ein Objekt der Klasse `SOMObject` instanziert und dem Project-Container `SOMObjects` hinzugefügt. Durch die flexible Gestaltung der Implementierung ist es theoretisch möglich einem Projekt mehrer SOM Server zuzuordnen, zum derzeitigen Entwicklungsstand ist dies jedoch durch die Anwendung Visual SOM Studio nicht vorgesehen.

6.2.2.3 Die Klasse SOMObject

Objekte der Klasse `SOMObject` stellen die Verbindung zwischen beliebigen Objekten SOM-Schnittstellen implementierender Klassen und einer graphischen Darstellung eines Objektes dar.
Sie können somit als graphischer Repräsentant eines SOM-Objekts angesehen werden.

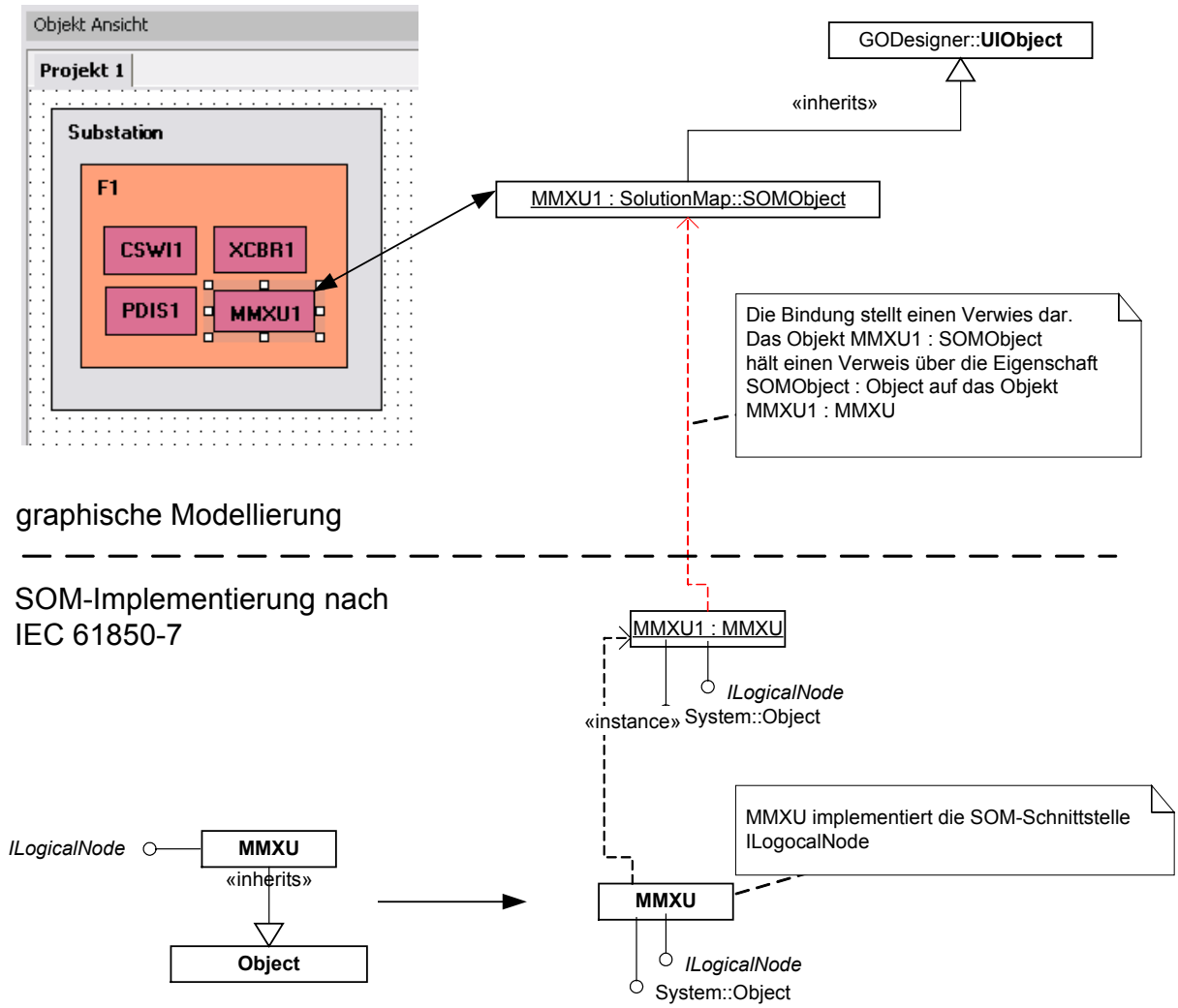

Abbildung 44: Verbindung zwischen SOMObjekten und Objekten die eine Schnittstelle des SOM's implementieren

Die Verbindung zwischen Graphik und SOM-Objekten wird dadurch hergestellt, dass Objekte der Klasse `SOMObject` über die Eigenschaft `SOMObject` auf Objekte einer Klasse, die eine SOM-Schnittstelle implementiert, verweisen. Zum anderen ist die Klasse `SOMObject` abgeleitet von der Klasse `GODesigner.UIObject` und stellt somit eine graphische Oberfläche bereit.

Objekte der Klasse `SOMObject` haben neben der Aufgabe, als graphische Präsenz eines SOM-Objektes zu agieren, zusätzlich die Aufgabe, einen hierarchischen Objektbaum verwalten zu können. Aus diesem Grund besitzt die Klasse `SOMObject` einen Container `SOMObjects`. Dieser Container kann weitere Objekte der Klasse `SOMObject` verwalten.

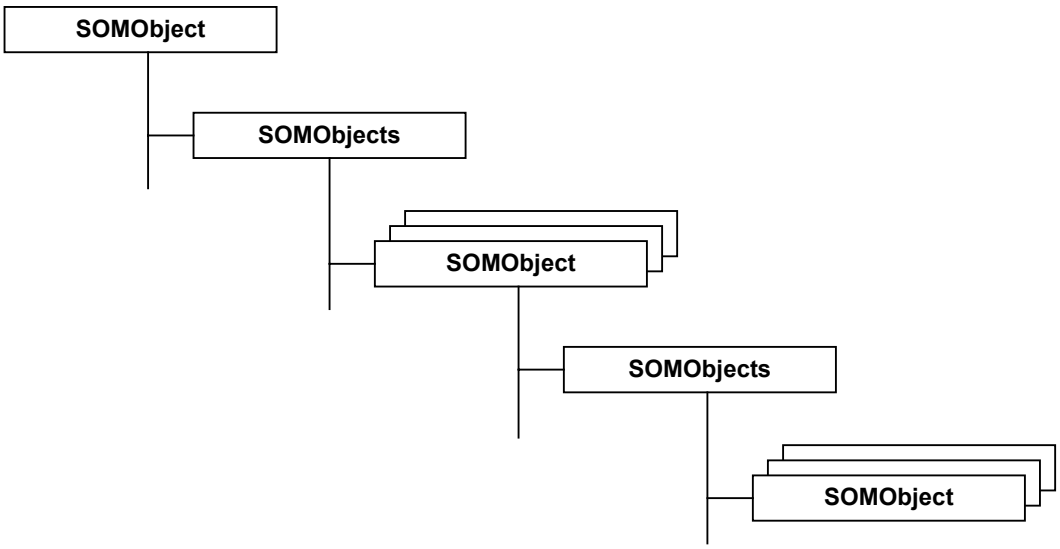

Abbildung 45: SOMObjet-Objektbaum

Somit läst sich ein nach unten beliebig erweiterbarer Objektbaum modellieren.
Damit die virtuelle hierarchische Struktur auch graphisch dargestellt werden kann, müssen die Objekte nicht nur in dem Container `SOMObjects` enthalten sein, sondern auch in dem Container Controls der Klasse `SOMObject`. Jedes Objekt, das abgeleitet ist von `System.Windows.Forms.Control` wird auf der Graphikoberfläche des Objektes der Klasse `SOMObject` dargestellt, sobald es sich im Container Controls befindet.
Das Übertragen von Objektverweisen von dem Container `SOMObjects` in den Container Controls erledigt das Objekt der Klasse `SOMObject` eigenständig. Das nachfolgende Sequenzdiagramm veranschaulicht diesen Vorgang. Zudem ist im oberen Teil des Diagramms ein statischer Ausschnitt der Klasse `SOMObject` dargestellt, das die Lebenszyklen aller beteiligter Objekte darstellt. Die beiden einzeln aufgeführten Objekte SOMObject : UIObject und SOMObject : SOMObject sind als ein und das selbe Objekt zu interpretieren, da die Klasse `SOMOBject` von `UIObject` erbt.

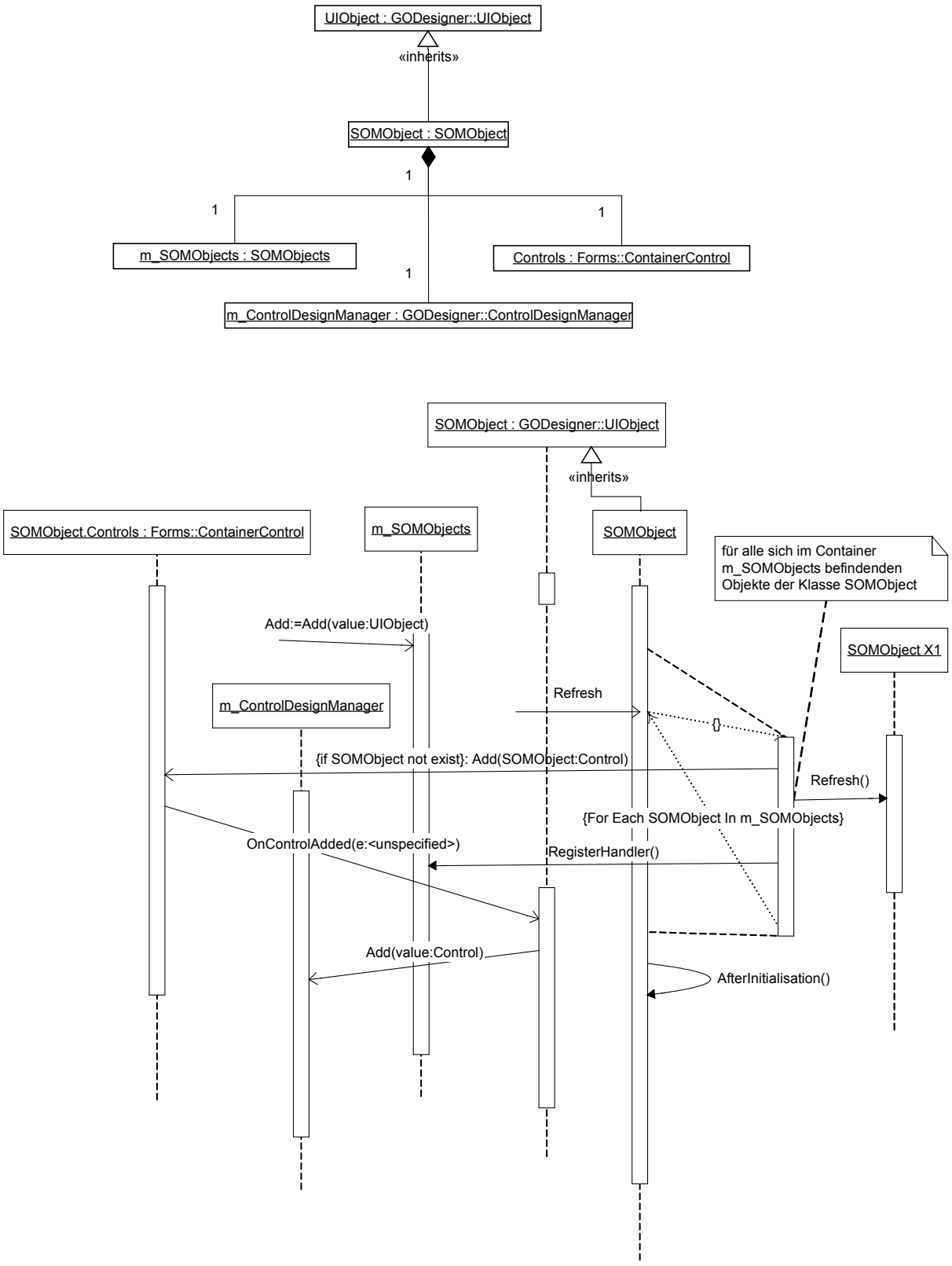

Abbildung 46: Sequenzdiagramm für das Hinzufügen eines UIObjects

Durch Aufruf der Methode `Add()` am SOMObjects-Container wird diesem ein neues Objekt der Klasse `SOMObject` hinzugefügt. Anschließend wird es jedoch nicht direkt in

den Controls-Container übertragen. Dies erfolgt erst mit dem Methodenaufruf `Refresh()` des jeweiligen Objekts der Klasse `SOMObject`. Innerhalb der Methode `Refresh()` wird der gesamte SOMObjects-Container durchlaufen und jedes sich darin befindende Objekt, falls es sich noch nicht in dem Controls-Container befindet, in diesen eingefügt. Zudem wird für jedes Objekt der Klasse `SOMObject` dessen Methode `Refresh()` aufgerufen, um somit rekursiv den gesamten Objektbaum anzuzeigen.
Weiterhin ist es Aufgabe der Objekte der Klasse `SOMObject`, Informationen über das jeweils zu referenzierende SOM-Objekt bereitzuhalten um diese eigenständig instanzieren zu können. Dafür sind folgende Informationen notwendig: der Klassenname, sowie der Speicherort von dem aus das Objekt instanziert werden kann; sprich der Assemblyname. Zudem ist für die gängigen SOM-Objekte ein Name anzugeben. Dafür stellt die Klasse `SOMObject` die Eigenschaft `AssemblyName`, `ClassName` und `ObjectName` bereit.
Neben diesen Informationen kann ein Object der Klasse `SOMObject` über die Eigenschaft SOMType dahingehend spezialisiert weren, dass es nur einen bestimmten SOM-Objekttypen referenzieren kann. Gültige SOMTypen sind durch die Enumeration `enType` spezifiziert.

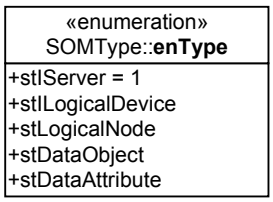

Abbildung 47: Die Enumeration enType

Die Klasse `SOMObject` ist so implementiert, das sobald die Informationen zur Instanzierung eines SOM-Objektes vorhanden sind, ein solches Objekt instanziert wird. Dabei wird über die Eigenschaft SOMObject : Object direkt eine Referenz auf das neu instanzierte Objekt gesetzt. Im folgenden Sequenzdiagramms ist der Vorgang der Instanzierung eines SOM-Objekts mit Hilfe des .NET-Reflectionframeworks verdeutlicht.

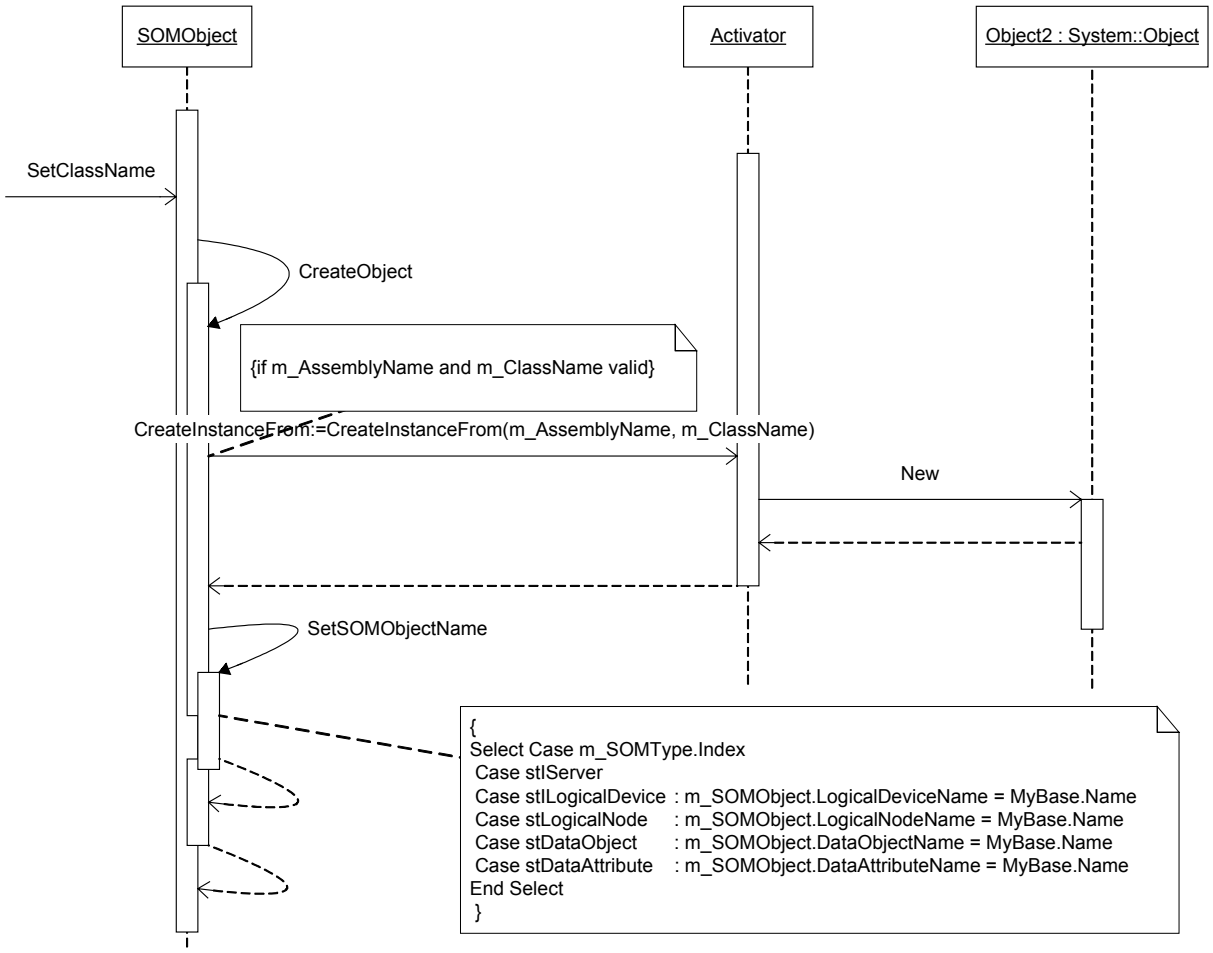

Abbildung 48

Zunächst wird ein Name über die Eigenschaft ClassName gesetzt. Daraufhin ruf das Objekt `SOMObject` die private Methode `CreateObject()` auf. Enthalten die Variablen m_AssemblyName und m_ClassenName gültige Werte, so wird über das Activator-Objekt des .NET-Reflectionframeworks die Methode `Activator.CreateInstanceFrom()` mit den entsprechenden Parametern aufgerufen. Die Methode liefert einen Verweis auf das neu instanzierte Objekt. Dieser Verweis wird in der privaten Variable `m_SOMObject` gespeichert. Anschließend wird die private Methode `SetSOMObjectName` aufgerufen. Hier wird selektiv nach dem SOMTyp unterschieden und jeweils der eigene Objektname dem neuen SOM-Objekt zugewiesen.

Wie bereits erwähnt sind alle Klassen des dargestellten Objektmodells serialisierbar. Um eine Klasse serialisierbar und damit Instanzen dieser Klasse persistent zumachen, stellt das .NET-Framework eine einfach anzuwendende Methode bereit. Die Klasse die serialisiert werden soll, muss lediglich mit dem Attribut [`Serializable`] gekennzeichnet werden.

```
<Serializable()> Public Class SOMObject
      .
      .
End Class
```

Eine als serialisierbar deklarierte Klasse kann mit vorhandenen Serialisierungsformattern des .NET-Frameworks serialisiert werden. Diese Methodik wird bei den Klassen `Solution`, `Projects`, `Project` und `SOMObjects` verwendet. Die dargestellte simple Serialisierung

[1] kann bei der Klasse `SOMObject` nicht angewendet werden. Dies liegt darin begründet, dass die Klasse `SOMObject` von der Klasse `UIObject` und damit indirekt abgeleitet ist von der Klasse `System.Windows.Forms.Control`. Die Klasse `System.Windows.Forms.Control` ist nicht mit dem `[Serializable]`-Attribut implementiert. Da sie integraler Bestandteil des .NET-Frameworks ist, kann diese Implementierung der Klasse nachträglich nicht geändert werden.
Um eine Klasse serialisierbar zu machen, die von einer nicht serialisierbaren Klasse erbt, stellt das .NET-Framework eine alternative Methode bereit. Bei der sogenannten benutzerkontrollierten Serialisierung muss die entsprechend zu serialisierende Klasse den Serialisierungsprozess selbst steuern. Der Serialisierungsprozess wird dabei über Methoden der Schnittstelle `ISerializable` gesteuert.
Aus diesem Grund implementiert die Klasse `SOMObject` die Schnittstelle `ISerializable`.

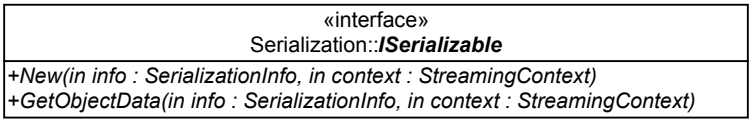

Abbildung 49: Das Interface `ISerializable`

Die Schnittstelle `ISerializable` erfordert die Implementierung der Methode `GetObjectData()`. Diese Methode wird beim Serialisierungsvorgangs von dem beteiligtem Serialisierungformatter aufgerufen.

Während der Serialisierung ruft der Formatter die Methode `GetObjectData()` des Objekts der Klasse `SOMObject` auf. Innerhalb der Methode `GetObjectData` wird das info-Objekt mit den entsprechend zu serialisierenden Daten gefüllt. Diese Daten bestehen aus Schlüssel-Werte-Paaren, die sich aus der zu serialiserenden Eigenschaft und deren Wert zusammensetzt.

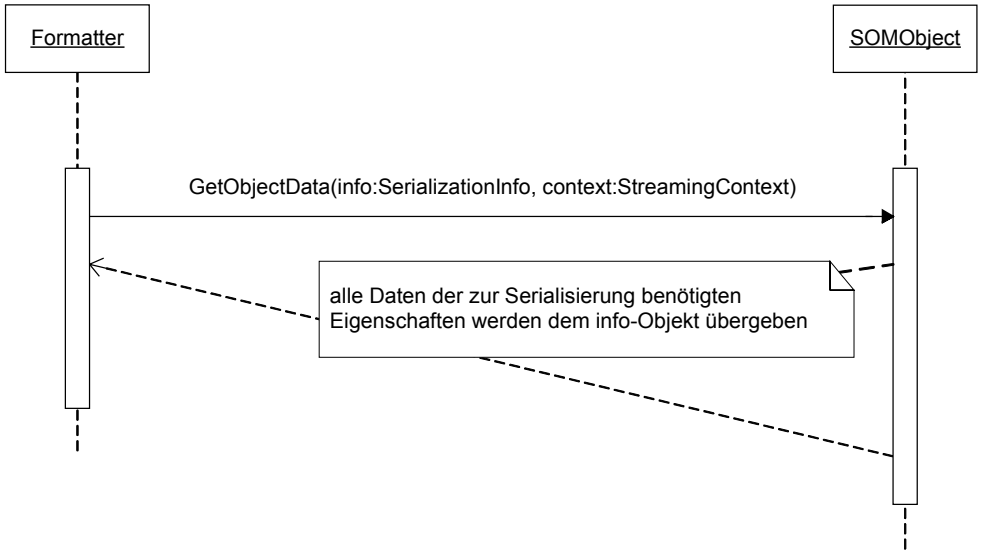

Abbildung 50: Objektserialisierung

Innerhalb der Methode `GetObjectData()` der Klasse `SOMObject` werden die benötigten Schlüssel-Werte-Paare mittels des .NET-Reflectionframeworks ermittelt und über die Methode `AddValue` dem Info-Objekts hinzugefügt.
Die Eigenschaft `Font`, sowie die Eigenschaft `SOMObject` des Objektes der Klasse `SOMObject` werden gesondert behandelt. Objekte der Klasse `Font` können nicht serialisiert werden. Um die in einem Font-Objekt enthaltenen Daten dennoch speichern

zu können, wird die Struktur[1] `XmlFont` der Komponente `SerializationHelperLib` verwendet. Objekte vom Typ `XmlFont` sind serialisierbar. Vor der Serialiseirung eines Objektes der Klasse `Font` wird dieses in ein Objekt vom Typ `XmlFont` konvertiert.
Die Werte der Eigenschaft `SOMObject` eines Objekts der Klasse `SOMObject` wird nicht serialisiert. Die Eigenschaft hält, wie oben beschrieben, einen Verweis auf ein SOM-Objekt. Da nicht davon ausgegangen werden kann, dass ein SOM-Objekt serialisierbar ist und es in solch einem Fall zu einer SerializationExeption kommen würde, wird diese Eigenschaft von der Serialisierung ausgeschlossen.

Damit die eine Klasse nicht nur serialisierbar, sondern auch deserialisierbar ist, muss ein entsprechender mit Parametern versehene Konstruktor implementiert werden. Beim Vorgang des Deserialisierens wird der Konstruktor

```
Public Sub New(ByVal info As
      System.Runtime.Serialization.SerializationInfo, ByVal context As
      System.Runtime.Serialization.StreamingContext)
```

aufgerufen.

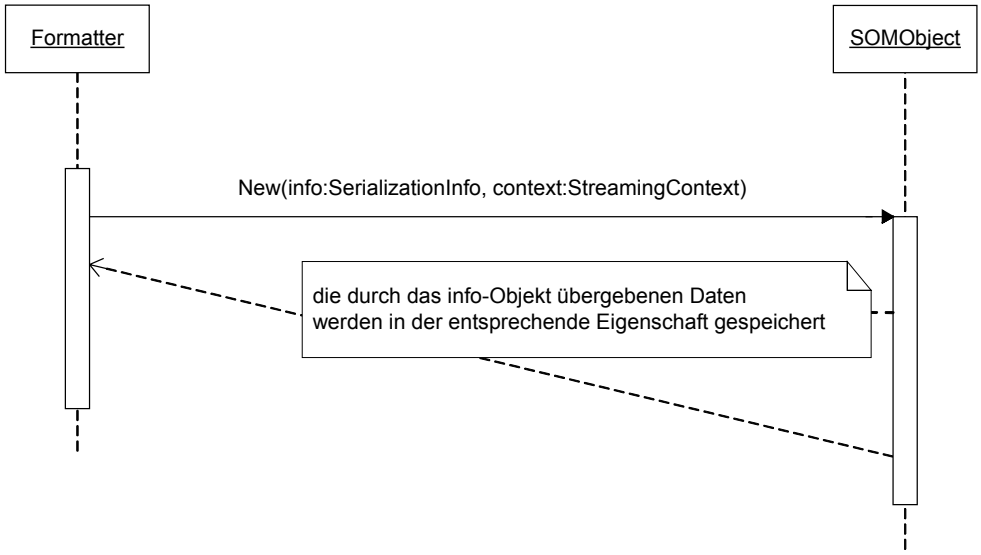

Abbildung 51: Objektdeserialisierung

Über den Parameter `info` werden die entsprechend zu deserialisierenden Werte übergeben. Innerhalb des Konstruktors werden die entsprechenden Eigenschaften des Objekts der Klasse `SOMObject` mit den übergebenen Werten gesetzt.

Um Objekte, wie hier vorgestellt, über den Weg der benutzerkontrollierten Serialisierung serialisierbar zu machen, müssen deren Klassen neben der Implementierung der Schnittstelle `ISerializable` zusätzlich wie bei der simplen Serialisierung mit dem `[Serializable]`-Attribut deklariert werden.

6.2.2.3.1 Besonderheiten der Klasse SOMObject

Die Anwendung Visual SOM Studio verwendet zur Parametrierung von Objekten jeglicher Art, das sogenannte Eigenschaftsfenster (PropertyGrid). So werden auch Objekte der Klasse `SOMObject` über dieses Steuerelement, wie in Abbildung 41 dargestellt, angezeigt

[1] C# Programmer's Reference: A **struct** type is a value type that can contain constructors, constants, fields, methods, properties, indexers, operators, and nested types.

und konfiguriert. Damit die Parametrierung eines Objektes der Klasse `SOMObject` über das Eigenschaftsfenster komfortabler wird, verfügen bestimmte Eigenschaften der Klasse `SOMObject` über Informationen, die dem Eigenschaftsfenster mitteilen, wie es sich bei der Editierung der entsprechenden Eigenschaft verhalten soll. Bei der Parametrierung der Eigenschaft `AssemblyName` eines SOMObjects, soll es dem Anwender ermöglicht werden den Wert über einen Dateiauswahldialog zu wählen. Zu diesem Zweck wird die Eigenschaft `AssemblyName` der Klasse `SOMObject` mit dem folgenden Attribut erweitert.

```
<Editor(GetType(UITypeEditorLib.OpenFileEditor), GetType(UITypeEditor))> _
Public Property AssemblyName() As String
```

Dieses Attribut gibt dem Eigenschaftsfenster bekannt, dass es im Falle der Editierung der AssemblyName-Eigenschaft den `UITypeEditor UITypeEditorLib.OpenFileEditor` verwenden soll. Der Editor `UITypeEditorLib.OpenFileEditor` sorgt dafür, dass im Eigenschaftsfenster neben dem Editierfeld ein Schalter eingeblendet wird.

Abbildung 52: Die Eigenschaft AssemblyName im Eigenschaftsfenster

Über diesen Schalter lässt sich der Dateiauswahldialog öffnen.
Wird die ClassName-Eigenschaft eines Objektes der Klasse `SOMObject` über das Eigenschaftsfenster editiert, so verhält sich das Eigenschaftsfenster so, dass lediglich solche Klassen zu Auswahl angezeigt werden, die sich in dem zuvor ausgewähltem Assembly befinden und gleichzeitig die SOM-Schnittstelle implementieren, die über die Eigenschaft `SOMInterface` des `SOMObjects` angegeben ist.

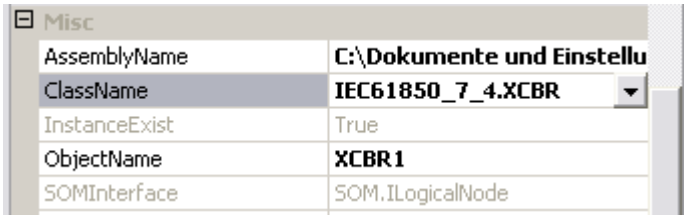

Abbildung 53:

In der folgenden Abbildung sind alle Klassen aufgelistet, die die Schnittstelle SOM.IlogicalNode implementieren und im Assembly IEC61850_7_4.dll enthalten sind.

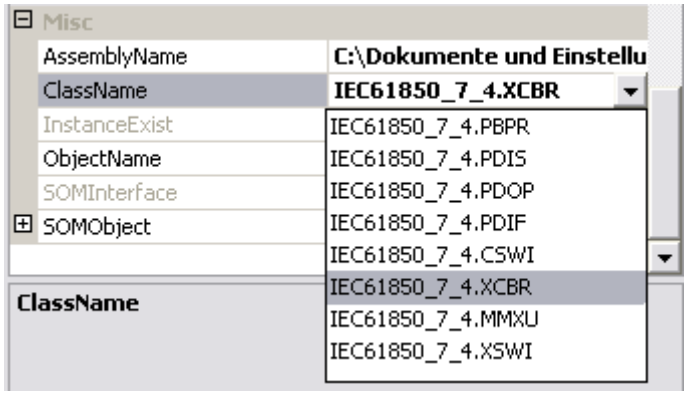

Abbildung 54: Auswahlliste aller Klassen, die `ILogicalNode` implementieren

Diese Auswahlmöglichkeit der einzelnen Klassen wird dadurch erreicht, dass die Eigenschaft `ClassName` der Klasse `SOMObject` mit dem folgenden Attribut erweitert wird.

```
<TypeConverter(GetType(ClassNameConverter))> _
Public Property ClassName() As String
```

Während der Editierung der ClassName-Eigenschaft ist die Klasse `ClassNameConverter` für die Zusammenstellung der Listenfeldeinträge und deren Darstellung wie in Abbildung 54 gezeigt zuständig.
Somit kann einem graphischen SOMObject ein SOM-Objekt zugeordnet werden.
Ein anderer Weg ein SOM-Objekt einem graphischen SOMObject zuzuweisen, ebenfalls mit Hilfe des Eigenschaftsfensters, kann über die Eigenschaft `SOMObject` des jeweiligen SOMObjects erfolgen. Dafür ist die Eigenschaft `SOMObject` der Klasse `SOMObject` mit den folgend dargestellten Attributen erweitert.

```
<Editor(GetType(SOMObjectEditor), GetType(UITypeEditor)), _
TypeConverter(GetType(SOMObjectConverter))> _
Public Property SOMObject() As Object
```

Die Klasse `SOMObjectEditor` ist dafür zuständig, dass beim Editieren der Eigenschaft `SOMObject` im Eigenschaftsfenster eine Schaltfläche neben dem Editierfeld angezeigt wird. Durch Betätigen des Schalters wird ein Formular der Klasse `SOMObjectEditorUI` angezeigt.

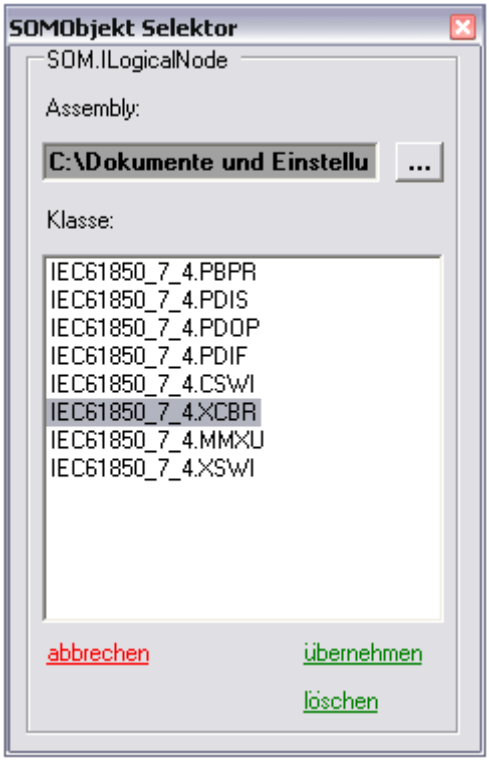

Abbildung 55: Der SOMObjekt Selektor

Über das aufgeschaltete Formular lassen sich dann Assemblyname und entsprechende Klasse auswählen.
Der TypeConverter `SOMObjectConverter` ist lediglich für die Anzeige des Klassennamens im Eigenschaftsfenster neben der Eigenschaft `SOMObject` zuständig.

Abbildung 56: Anzeige des Klassennamens im SOMObject-Feld

6.2.2.4 Die Klasse SolutionFormatter

Die Klasse `SolutionFormatter` hat die Aufgabe Objekte, die die Schnittstelle `ISolutionObjectBase` implementieren, zu serialisieren, sowie zu deserialisieren um damit solche Objekte persistent auf einem Datenträger zu speichern, bzw. von einem Datenträger in den Speicher zu laden.

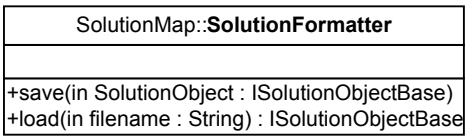

Abbildung 57: Die Klasse `SolutionFormatter`

Dazu stellt die Klasse `SolutionFormatter` zwei Methoden zur Verfügung. Um Objekte vom Typ `ISolutionObjectBase` zu serialisieren, wird die Methode `save()` aufgerufen. Als Parameter wird an die Methode `save()` das zu speichernde Objekt übergeben. Eine zusätzliche Angabe über den Speicherort des zu speichernden Objekts ist der Methode nicht zu übergeben, da Objekte vom Typ `ISolutionObjectBase` diese Information selbst bereitstellen. Innerhalb der Methode `save()` werden diese Informationen aus dem übergebenen Objekt gelesen und mit Hilfe dieser und einem .NET-Formatter das Objekt serialisiert und gespeichert.
Die Methode `load()` stellt die Funktionalität des Deserialisierens eines Objektes vom Typ `ISolutionObjectBase` bereit. Der Methode `load()` muss per Parameter der entsprechende Speicherort des zu deserialisierenden Objekts mit übergeben werden. Die Methode `load()` deserialisiert mittels eines .NET-Formatter und der Angabe des Speicherorts das entsprechende Objekt von seinem Speicherort. Als Rückgabewert liefert die Methode `load()` einen Verweis auf das entsprechend deserialisierte Objekt.
Bei der Serialisierung, sowie bei der Deserialisierung wird der .NET-BinaryFormatter benutzt. Theoretisch könnten jedoch auch andere Formatter eingesetzt werden. Das .NET-Framework stellt neben dem verwendetem BinaryFormatter einen SoapFormatter bereit. In Tests hat sich jedoch gezeigt, dass beim Deserialisieren mittels des SoapFormatters Probleme beim Deserialisieren von Objekten der Klasse `SOMObject` auftreten. Aus diesem Grund wird ausschließlich der BinaryFormatter des .NET-Frameworks verwendet. Dadurch, dass der SolutionFormatter Objekte, die die Schnittstelle `ISolutionObjectBase` implementieren, serialisieren kann, können sowohl Objekte der Klasse `Solution` als auch der Klasse `Project` serialisiert und deserialisiert werden.

6.2.2.5 Die Klassen SOMObjectColors und SOMObjectColor

Die Klasse `SOMObjectColors` dient als statischer Container für Objekte der Klasse `SOMObjectColor`. Durch die Objekte der Klasse `SOMObjectColor` wird eine Zuordnung zwischen dem SOMTyp eines SOMObject und einer entsprechenden Farbe, die als Hintergrundfarbe bei der graphischen Darstellung verwendet wird, hergestellt. Somit können beim Instanzieren eines SOMObjectes deren Hintergrundfarbe aus dem Container `SOMObjectColors` bezogen werden.

```
SOMObject.BackColor = SOMColors.Item(SOMType).Color
```

Der Container `SOMObjectColors` enthält fünf Objekte der Klasse `SOMObjectColor`. Jedes Objekt der Klasse `SOMObjectColor` besitzt eine Farbe und ist einem der fünf SOMTypen (s. Abbildung 47) über einen Schlüssel des Containers `SOMObjectColors` zugeordnet.
Die Farbfestlegung ist fest codiert. Um unterschiedliche Farbkonfigurationen eines Projekts festlegen zu können, wäre eine Erweiterung der SOMObjectColors-Klasse denkbar, die die Farbfestlegungen aus einer Konfigurationsdatei bezieht.

In der folgenden Abbildung sind die standardmäßig festgelegten Farben eines jeden SOMObjects dargestellt.

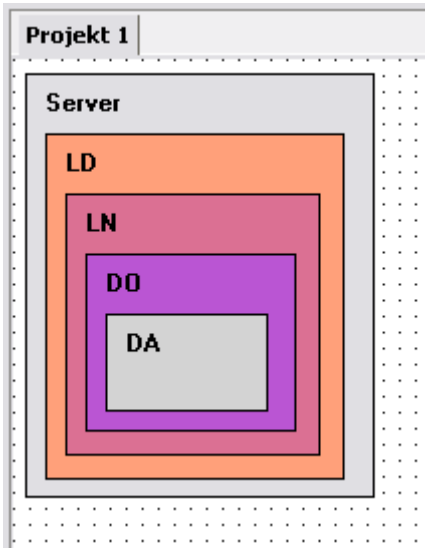

Abbildung 58: Festlegung der Farben für entsprechende SOMTypen

6.3 Die Komponente GODesigner

6.3.1 Aufgabe der Komponente GODesigner

Die Komponente GODesigner stellt Klassen zur Verfügung mittels derer, zur Laufzeit einer Anwendung, ein Designer für graphische Objekte realisiert werden kann. Die Komponente GODesigner enthält vier Klassen. Die zentrale Klasse ist die Klasse `UIObject`. Diese ist abgeleitet von `System.Windows.Forms.UserControl`. Instanzen dieser Klasse stellen zur Laufzeit eine graphische Benutzeroberfläche bereit. Auf dieser graphischen Oberfläche können andere graphische Objekte bearbeitet werden. D.h. sie können direkt mit der Maus in Position und Größe verändert werden. Vorraussetzung dafür ist, dass ein solches Objekt abgeleitet ist von `System.Windows.Forms.Control`. Um ein graphisches Objekt mit Hilfe der Benutzeroberfläche eines Objekts der Klasse `UIObject` bearbeiten zu können, muss das zu bearbeitende Objekt in den Container `Controls` des UIObject eingefügt werden. Dies geschieht über die Methode `Add` oder `AddRange` des Controls-Container. Z.B.: `Me.UIObject1.Controls.Add(Me.Button1)`.

Abbildung 59: Button-Instanz auf der graphischen Oberfläche eines UIObjects

Objekte die sich auf der Designfläche eines Objektes der Klasse `UIObject` befinden, können durch Selektion mit der linken Maustaste markiert werden.

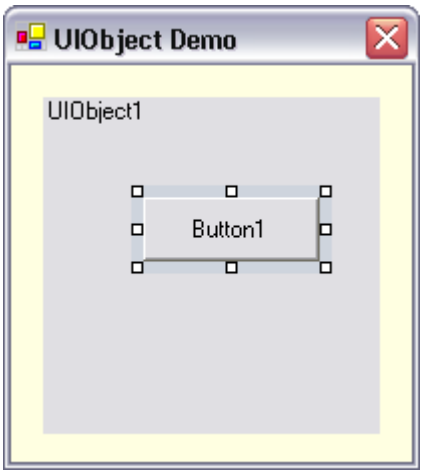

Abbildung 60: markierte Button-Instanz

Durch gleichzeitiges drücken der STRG-Taste können auch mehrer Objekte gleichzeitig selektiert werden. Nachdem ein Objekt markiert wurde, erscheinen um dieses herum so genannte Objektgriffe mit denen sich das Objekt „anfassen" lässt. An jeder der vier Ecken des Objekts einer, um das Objekt in seiner Größe zu ändern, sowie jeweils ein Griff in der Mitte einer jeden Seitenkante um dessen Größe in horizontaler, sowie in vertikaler Richtung zu ändern.

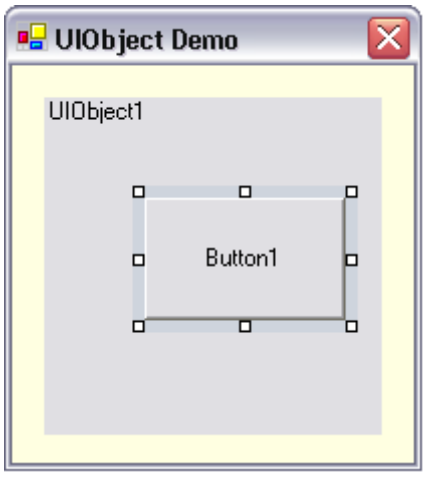

Abbildung 61

Um die Größe eines Objektes zu ändern, muss die Maus auf den entsprechenden Griff geführt werden und anschließend bei gedrückt gehaltener linker Maustaste bewegt werden.
Um die Position eines Objektes zu verändern muss die Maus über das entsprechenden Objekt bewegt werden und anschließend bei gedrückt gehaltener linker Maustaste bewegt werden.

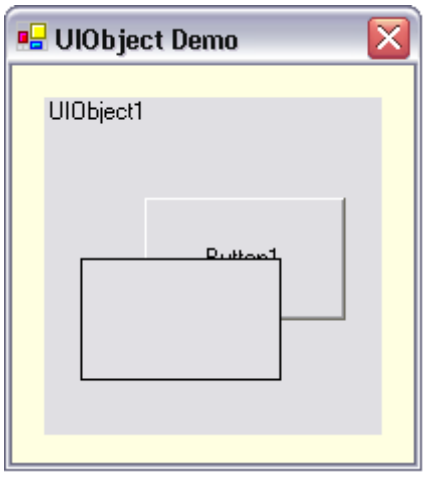

Abbildung 62: verschobenes Objekt

6.3.2 Das Objektmodell

Die anderen drei Klassen der Komponente `GODesigner` sind nicht für die direkte Verwendung außerhalb der Komponente `GODesigner` bestimmt. Lediglich die Klasse `UIObject` verwendet Instanzen der Klassen `ControlDesignManager` und `ControlDesigner`.
Das Objektmodell der Komponente `GODesigner` sieht wie folgt aus:

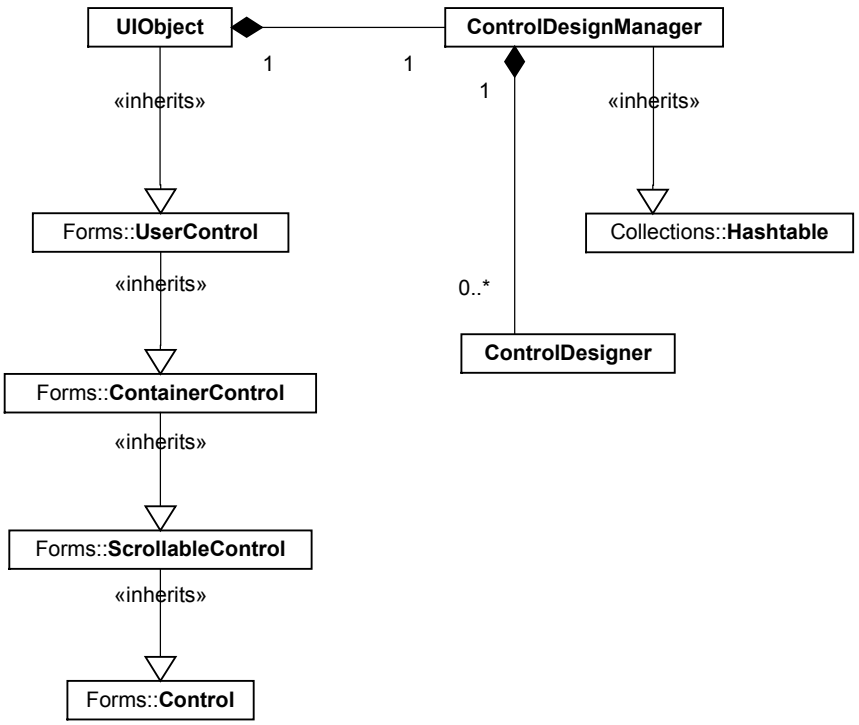

Abbildung 63: Objektmodell der Komponente `GODesigner`

6.3.2.1 Die Klasse UIObject

Die Klasse `UIObject` wird direkt abgeleitet von der .NET-Framework-Klasse `System.Windows.Forms.UserControl`. Gleichzeitig besteht eine Beziehung zwischen der Klasse `UIObject` und der Klasse `ControlDesignManager`. Diese Beziehung ist als Komposition mit einer 1 zu 1 Kardinalität spezifiziert. Dies bedeutet, dass während des gesamten Lebenszykluses eines Objektes der Klasse `UIObject` gleichzeitig eine Instanz eines Objektes der Klasse `ControlDesignManager` existiert. Gleichzeitig hält das Objekt der Klasse `UIObject` mit der Membervariablen `m_ControlDesignManager` einen Verweis auf das Objekt der Klasse `ControlDesignManager`.

6.3.2.2 Die Klasse ControlDesignManager

Die Klasse `ControlDesignManager` ist direkt abgeleitet von der Klasse `System.Collections.Hashtable` und stellt somit eine Containerklasse dar. Die Klasse `ControlDesignManager` agiert dabei als Container für Objekte der Klasse ControlDesigner. Die Klasse `ControlDesignManager` ist Verwalter aller graphischer Objekte, die auf der Benutzeroberfläche des Objektes der Klasse `UIObject` vorhanden sind. Der ControlDesignManager ist dafür zuständig, welches Objekt bewegt, bzw.

welches Objekt markiert oder unmarkiert wird. Da die Selektion eines Objektes mittels Mausklicks geschieht und der ControlDesignManger darüber informiert sein muss, wann dies geschieht, abonniert er einige für ihn wichtige Ereignisse eines jeden graphischen Objektes das sich in seiner Verwaltung befindet.

6.3.2.3 Die Klasse ControlDesigner

Die Klasse `ControlDesigner` ist dafür zuständig, wie die Markierung eines graphisches Objekt auf der Benutzeroberfläche eines Objektes der Klasse `UIObject` dargestellt wird. Der ControlDesigner zeichnet um das markierte Objekt, auf der Oberfläche des Objektes der Klasse `UIObject`, einen Rahmen bestehend aus Objekten der Klasse `System.Windows.Forms.Panel`. Da jedes graphische Objekt über einen Markierungsrahmen verfügt, ob dieser sichtbar oder unsichtbar ist, wird zu jedem graphischen Objekt ein entsprechendes Objekt der Klasse `ControlDesigner` angelegt. Die Zugehörigkeit zwischen einem graphischen Objekt und dem entsprechenden ControlDesigner wird im ControlDesignManager verwaltet. Der ControlDesigner ist dafür zuständig ob das entsprechende graphische Objekt als markiert oder als unmarkiert angezeigt werden soll.

6.3.3 Implementierung und Funktionsweise

6.3.3.1 Anzeigen eines graphischen Objekts in der Benutzeroberfläche eines UIObjects

Das folgende Sequenzdiagramm verdeutlicht den Ablauf des Hinzufügens eines graphischen Objekts zu einem UIObject.

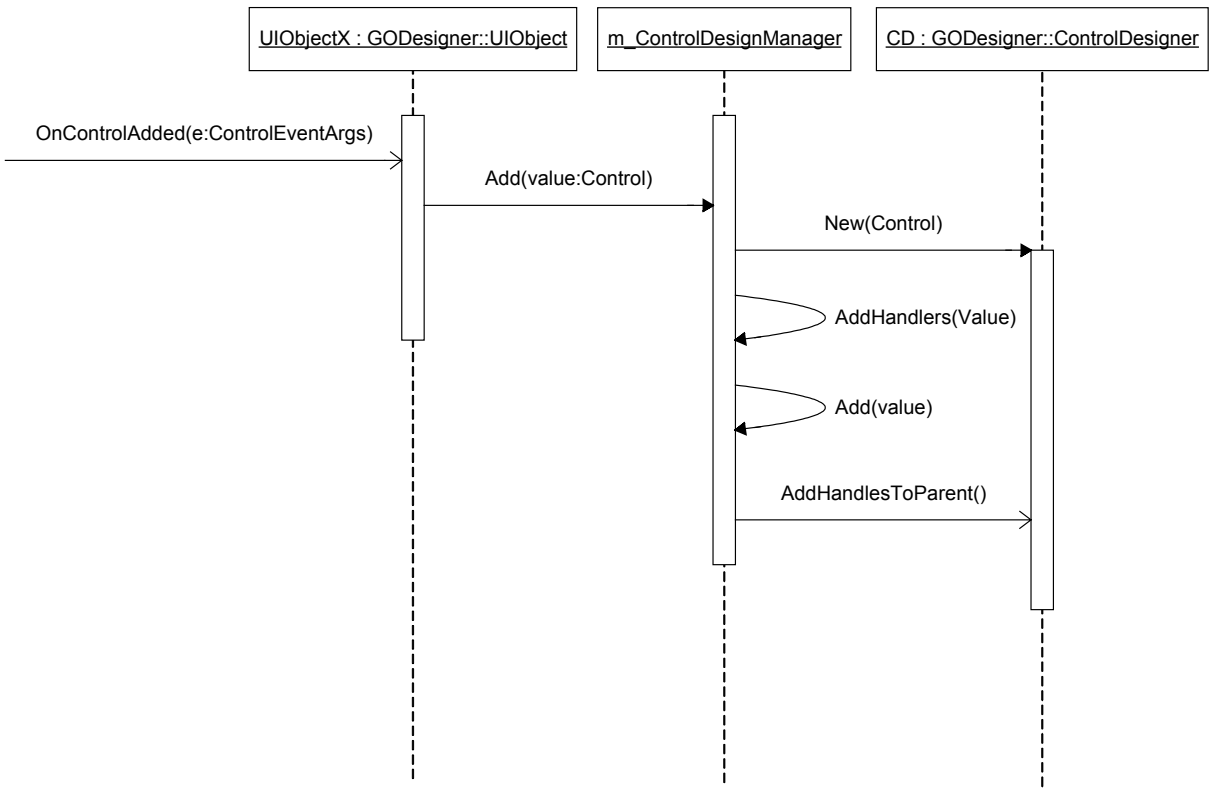

Abbildung 64: Hinzufügen eines graphischen Objekts zu einem UIObject

Wird ein graphisches Objekt in den `Controls`-Container des Objekts der Klasse `UIObject`, über die Methoden `Add` oder `AddRange`, hinzugefügt, wird die Ereignisbehandlungsroutine `OnControlAdded(ByVal e As System.Windows.Forms.ControlEventArgs)` des Objekts der Klasse `UIObject` aufgerufen. Als Parameter empfängt die Methode eine Variable vom Typ `ControlEventArgs`. Über diesen Parameter wird eine Referenz auf das neu hinzugefügte Objekt mitgeliefert. Dieser Verweis wird über die Methode `Add` des ControlDesignManagers an die Verwaltung des ControlDesignManagers übergeben. Handelt es sich um ein Objekt, das sich noch nicht in der Verwaltung des ControlDesignManagers befindet, wird es in dessen Liste aufgenommen. Zudem wird ein neues Objekt der Klasse `ControlDesigner` instanziert. Über den Konstruktor wird dem neuen Objekt der Klasse `ControlDesigner` ein Verweis auf das graphische Objekt mitgeliefert. Damit hat der ControlDesigner direkten Zugriff auf das graphische Objekt, sowie über die Eigenschaft `Parent` des graphischen Objekts, Zugriff auf das übergeordnete Objekt der Klasse `UIObject`. Anschließend abonniert der ControlDesignManager bestimmte für ihn wichtige Ereignisse des graphischen Objekts und dessen übergeordnetes Objekt der Klasse `UIObject`.

```
Private Sub AddHandlers(ByVal Value As Control)
    AddHandler Value.MouseDown, AddressOf Me.Controls_MouseDown
    AddHandler Value.MouseMove, AddressOf Me.Controls_MouseMove
    AddHandler Value.MouseUp, AddressOf Me.Controls_MouseUp
    AddHandler Value.KeyDown, AddressOf Me.Controls_KeyDown
    AddHandler Value.KeyUp, AddressOf Me.Controls_KeyUp
    AddHandler Value.Parent.KeyDown, AddressOf Me.Controls_KeyDown
    AddHandler Value.Parent.KeyUp, AddressOf Me.Controls_KeyUp
    AddHandler Value.Parent.MouseDown, AddressOf Me.ParentControl_MouseDown
End Sub
```

Damit erhält der ControlDesignManager die Möglichkeit auf bestimmte Ereignisse des graphischen Objekts oder dessen übergeordnetem Objekt der Klasse `UIObject` zu reagieren.
Anschließend werden das graphische Objekt und der dazugehörende ControlDesigner in die interne Liste des ControlDesignManagers aufgenommen. Die interne Liste ist als Hashtable implementiert. Die Hashtable fungiert als assoziatives Array bestehend aus Schlüssel-Werte-Paaren. Die Assoziation zwischen einem graphischen Objekt und dem dazugehörenden CondrolDesigner wird über diese Liste hergestellt, indem das graphische Objekt als Schlüssel und der ControlDesigner als Wert in der Hashtable gespeichert werden.

Schlüssel	Wert
:UIObject1	:ControlDesigner1
:UIObjectXYZ	:ControlDesignerXYZ
:UIObjectA	:ControlDesignerA

Abbildung 65: Beispielhafte Darstellung der Assoziation zwischen Schlüssel und Wert

Durch diese „lose" Verbindung zwischen graphischem Objekt und dem dazugehörenden ControlDesigner ist es möglich, jedes Objekt, das von `System.Windows.Forms.Control` abgeleitet ist, ohne dieses mit zusätzlichen Schnittstellen und Funktionalitäten ausstatten

zu müssen, auf der Benutzeroberfläche eines Objekts der Klasse `UIObject` bearbeiten und dargestellen zu können.
Danach wird die Methode `AddHandelsToParent()` am ControlDesigner aufgerufen. Der ControlDesigner fügt daraufhin dem übergeordnetem Objekt der Klasse `UIObject` des graphischen Objekts einige Objekte der Klasse `System.Windows.Forms.Panel` hinzu. Diese werden sichtbar wenn das entsprechende graphische Objekt markiert ist und stellen den Markierungsrahmen zusammen mit den Objektgriffen dar (siehe Abbildung 60).

6.3.3.2 Markieren eines graphischen Objekts in der Benutzeroberfläche eines UIObjects

Das folgende Sequenzdiagramm verdeutlicht den Ablauf des Markierens eines graphischen Objekts.

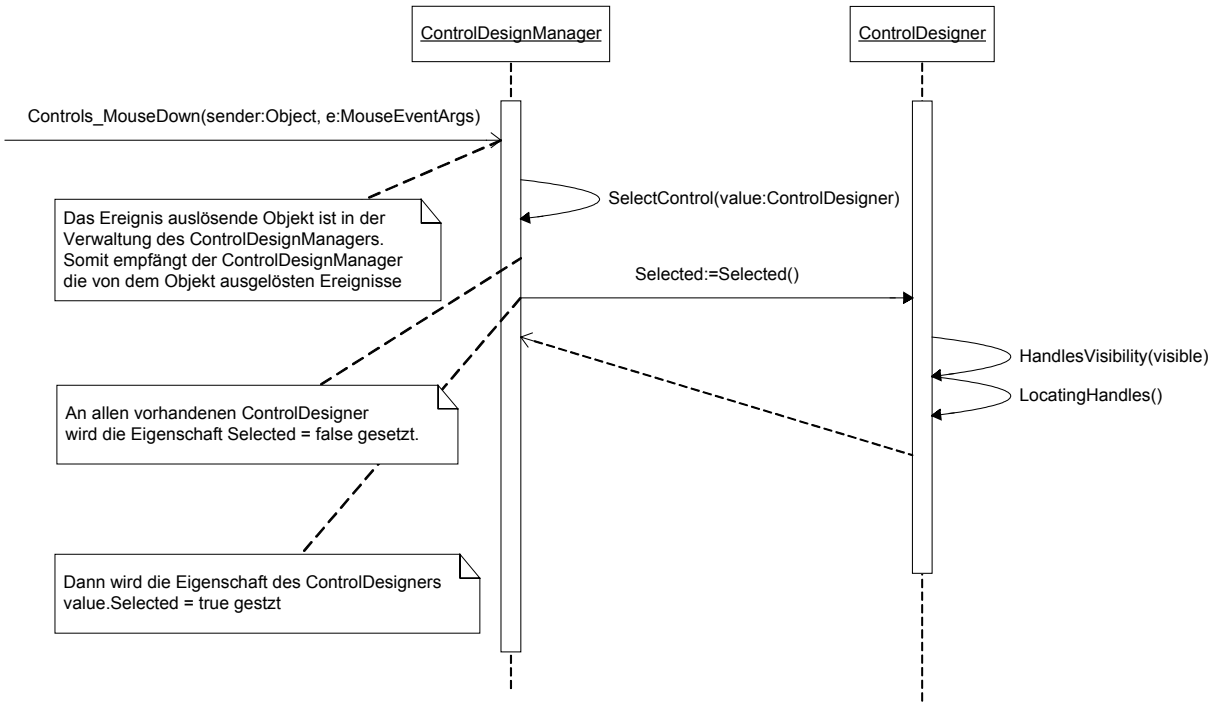

Abbildung 66: Markieren eines Objektes auf der Zeichenfläche eines UIObjects

Dadurch, dass der ControlDesignManager das MouseDown-Ereignis jedes einzelnen graphischen Objekts, welches sich in dem Controls-Container des UIObjects befindet, abonniert hat, wird bei auftreten dieses Ereignisses die Ereignisbehandlungsroutine
```
Private Sub Controls_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs)
```
des ControlDesignManagers aufgerufen. Über den Parameter `sender` wird dem ControlDesignManager mitgeteilt, welches Objekt das Ereignis ausgelöst hat. Da der ControlDesignManager eine assoziative Liste aller graphischen Objekte mit ihren zugehörigen ControlDesignern verwaltet, kann mit Hilfe dieses Parameters der entsprechende ControlDesigner ermittelt werden.
Wird die linke Maustaste betätigt, wird eine private boolesche Variable gesetzt, die wieder gelöscht wird, wenn die Maustaste losgelassen wird. Zudem wird die aktuelle Mausposition gespeichert. Anschließend wird die private Methode
```
Private Sub SelectControl(ByVal value As ControlDesigner)
```

aufgerufen und als Parameter des zuvor ermittelten ControlDesigners übergeben. Die Methode SelectControl setzt zunächst die Eigenschaft `Selected` jedes im ControlDesignManager verwaltetem ControlDesigner zurück (= false). Damit werden auf der Oberfläche des UIObjectes alle etwaigen Markierungen aufgehoben. Anschließend wird die Eigenschaft `Selected` des aktuellen ControlDesigners gesetzt (= true) .
Wird die Eigenschaft `Selected` des ControlDesigners gesetzt ruft dieser die privaten Methoden
`Private Sub HandlesVisibility(ByVal visible As Boolean)` und
`Private Sub LocatingHandles()` auf.
Die Methode `HandlesVisibility` setzt die Sichtbarkeit alle Objekte, die den Markierungsrahmen um das graphische Objekt darstellen, entsprechend dem Wert des übergebenem Parameters.
Die Methode `LocatingHandles()` positioniert alle Objekte, die den Markierungsrahmen um das graphische Objekt darstellen so, dass sie das graphische Objekt umschließen.

6.3.3.3 Besonderheiten der Klasse UIObject

Die Klasse UIObject implementiert ein Standard-PopUp-Menü, welches durch klicken der rechten Mausetaste aufgeschaltet wird.

Abbildung 67: Popup-Menü der Klasse UIObject

Das PopUp-Menü ist vorgesehen für Anwendungen in dem ein Objekt der Klasse `UIObject` als Container für weitere Objekte der Klasse `UIObject` eingesetzt wird.
Die Funktionalität des PopUp-Menüs ist jedoch nicht vollständig implementiert. Lediglich die Funktionen „in den Vordergrund" und „in den Hintergrund" sind konkret realisiert.
Die Funktionalität „löschen" ist so implementiert, dass das UIObject das Ereignis
`Public Event Delete(ByVal sender As UIObject)`
auslöst. Das bedeutet, dass das eigentliche Entfernen des UIObjects von einem anderen Ort aus durchgeführt werden muss. Dadurch wird jedoch eine größere Flexibilität und Kontrolle über das Löschen eines Objektes erzielt. Ebenso verhält es sich mit der Funktionalität „Eigenschaft". Diese löst das Ereignis
`Public Event Selected(ByVal sender As UIObject)` aus.

```
Private Sub mnuProperty_Click(ByVal sender As Object, ByVal e As
                    System.EventArgs) Handles mnuProperty.Click

    RaiseEvent Selected(Me)
End Sub

Private Sub mnuDelete_Click(ByVal sender As Object, ByVal e As
                    System.EventArgs) Handles mnuDelete.Click

    RaiseEvent Delete(Me)
End Sub
```

Somit kann von „außen" die Reaktion auf einen bestimmten Menüpunkt kontrolliert werden.
Die anderen PopUp-Menü-Punkte sind nicht implementiert. Hier ist jedoch ein ähnliches Vorgehen zu empfehlen.

Zu überdenken wären auch, ob nicht das vorgestellte PopUp-Menü im ControlDesigner implementiert werden kann, damit jedes graphische Objekt, das in einem UIObject dargestellt wird über ein solches PopUp-Menü direkt verfügt.

6.4 Die Komponente ObjectNameComboBox

6.4.1 Aufgabe der Komponente ObjectNameComboBox

Die Komponente ObjectNameComboBox enthält die Klasse `ComboBox`. Die Klasse Combobox ist direkt abgeleitet von der Klasse `System.Windows.Forms.Combobox`. Instanzen der Klasse `ObjectNameComboBox.Combobox` stellen Steuerelemente mit dem Verhalten einer DropDownList dar. Im Gegensatz zu der Standard-Combobox können in der `ObjectNameComboBox.Combobox` zwei Werte in einer Auswahlreihe dargestellt werden. Dabei wird der erste Wert fett dargestellt. Somit stellt dieses Steuerelement, die aus verschiedenen Entwicklungsumgebungen (VB 6.0, VisualInterDef, Visual Studio .NET ...) bekannte Auswahlliste des Eigenschaftsfensters, zur Auswahl bestimmter Objekte dar.

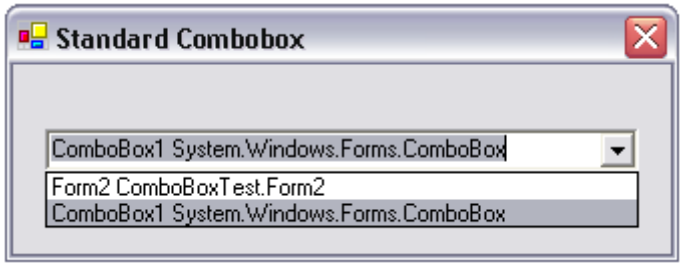

Abbildung 68: Die Standard-Combobox

Im Gegensatz zur oben dargestellten Standard-Combobox, stellt die `ObjectNameComboBox.Combobox` den ersten Wert **fett** dar.

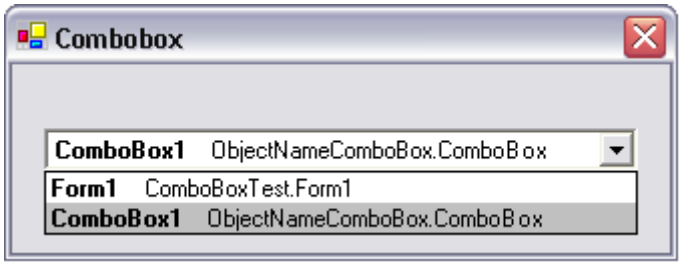

Abbildung 69: Die `ObjectNameCombobox.Combobox`

6.4.2 Das Objektmodell

Das folgende Klassendiagramm stellt das Objektmodell der Klasse `ObjectNameComboBox.Combobox` dar.

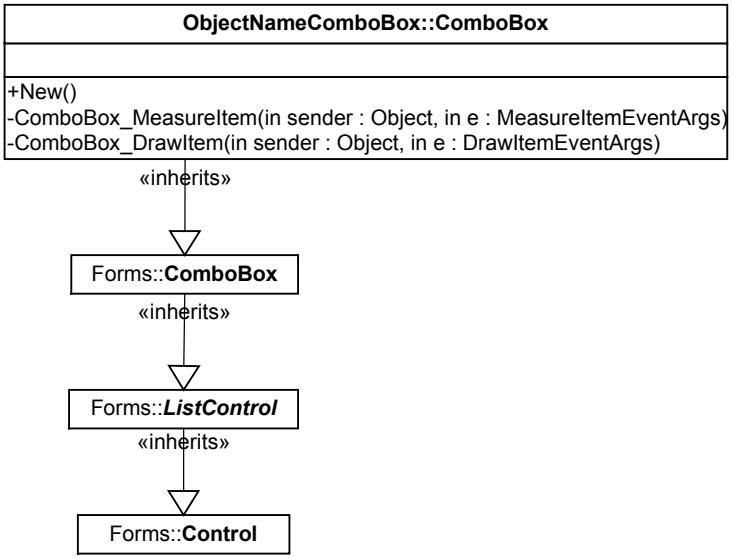

Abbildung 70: Die Klasse `ObjectNameComboBox.Combobox`

6.4.2.1 Die Klasse ObjectNameComboBox.Combobox

Die Klasse `ObjectNameComboBox.Combobox` unterscheidet sich von der Klasse Standard-Combobox dadurch, dass sie zwei Ereignisbehandlungsroutinen implementiert, die die Ereignisse der Basisklasse MeasureItem und DrawItem behandeln.

Die Klasse `System.Windows.Forms.Combobox` stellt eine Möglichkeit bereit, die Darstellung der Combobox und im Besonderen die der einzelnen Combobox-Einträge zu verändern und somit an besondere Bedürfnisse anzupassen.
Bevor ein Objekt der Klasse `Combobox` gezeichnet wird, löst es nacheinander die Ereignisse MeasureItem und DrawItem aus. Dabei werden jeweils entsprechende Parameter an die behandelnden Ereignisbehandlungsroutinen übergeben, die auf das aktuell zu zeichnende Combobox-Item verweisen.
In der Ereignisbehandlungsroutine, die das Ereignis MeasureItem behandelt, kann die Abmessung eines Combobox-Items gesetzt werden.
In der Ereignisbehandlungsroutine, die das Ereignis DrawItem behandelt, kann die Darstellung des Inhaltes der einzelnen Combobox-Items implementiert werden.

6.4.2.2 Implementierung und Funktionsweise

Die Abmessung der Combobox-Items der Klasse `ObjectNameComboBox.Combobox` wird in der Ereignisbehandlungsroutine `ComboBox_MeasureItem` festgelegt.

```
Private Sub ComboBox_MeasureItem(ByVal sender As Object, ByVal e As
      System.Windows.Forms.MeasureItemEventArgs) Handles MyBase.MeasureItem

    Dim CurrentFont As Font
    Dim ItemCaption As String, Fontname As String

    'Wenn nichts ausgewählt...
    If e.Index > 0 Then

        ItemCaption = MyBase.Items(e.Index)

        'Den Font erstellen
        CurrentFont = New System.Drawing.Font(MyBase.Font.Name,
            MyBase.Font.Size, FontStyle.Bold, GraphicsUnit.Pixel)


        'Rechteck mit den Ausmaßen des Textes erstellen
        Dim MySizeF As SizeF = e.Graphics.MeasureString(ItemCaption,
       CurrentFont)

        'Höhe und Breite setzen
        e.ItemHeight = MySizeF.Height + 6
        e.ItemWidth = MySizeF.Width

    End If
End Sub
```

Zunächst überprüft die Methode `ComboBox_MeasureItem` ob es sich bei dem übergebenden Index um ein gültige Assoziation mit einem vorhandenen Combobox-Item handelt. Ist dies der Fall wird der Text des Items gespeichert und ein neues Font-Objekt mit den Eigenschaften des Font-Objektes der Combobox instanziert. Dabei wird jedoch der Schriftstil auf fett gesetzt. Anschließend wird ein virtuelles Rechteck vom Typ `SizeF` erzeugt. Die Maße des Rechtecks werden durch die Länge des aktuellen Textes und der verwendeten Schrift bestimmt. Im letzten Schritt wird die Höhe, sowie die Breite des Rechtecks dem neu zu zeichnenden Combobox-Item zugewiesen. Dabei wird die Höhe um weitere sechs Pixel vergrößert, damit die darzustellende Schrift nicht direkt an den Rand des Combobox-Items stößt.

Das eigentliche Erscheinungsbild eines Combobox-Items wird in der Ereignisbehandlungsroutine `DrawItem` festgelegt. Hier kann z.B. der Hintergrund aber auch die Darstellung des Textes festgelegt werden.
Die Methode `DrawItem` der Klasse `ObjectNameComboBox.Combobox` ist so implementiert, dass sie einen Text, der durch das ASCII-Zeichen 255 separiert ist, zweigeteilt darstellt. Der erste Teil der Zechenkette wird **fett** dargestellt, der zweite in normaler Schriftgröße.

```
If MyBase.Items(e.Index).LastIndexOf(Chr(255)) > 0 Then
    FirstString = MyBase.Items(e.Index).Split(Chr(255))(0)
    SecondString = MyBase.Items(e.Index).Split(Chr(255))(1)
Else
    FirstString = MyBase.Items(e.Index)
End If
```

Um also einen Text wie in Abbildung 69 in der Combobox darzustellen, muss der Text der Combobox wie in folgendem Codesausschnitt gezeigt, der Combobox hinzugefügt werden.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
       System.EventArgs) Handles MyBase.Load

    Me.ComboBox1.Items.Add(Me.Name & Chr(255) & Me.GetType.ToString)
    Me.ComboBox1.Items.Add(Me.ComboBox1.Name & Chr(255) &
       Me.ComboBox1.GetType.ToString)

End Sub
```

Wichtig sind dabei die ASCII-Zeichen 255, die als Separatoren dienen.

7 Zusammenfassung und Ausblick

7.1 Stand der Arbeit

Die vorliegende Arbeit ist in zwei Teile, einen praktischen und einen schriftlichen Teil, gegliedert. Der praktische Teil umfasst die Implementierung der Softwareanwendung Visual SOM Studio. Die Software ist auf der .NET-Plattform der Firma Microsoft realisiert. Aufgabe der Anwendung Visual SOM Studio ist es ein Werkzeug bereit zustellen, mit dem Objektkonfigurationen auf Grundlage des SOM-Schnittstellenmodells erstellt werden können. Durch die Unterstützung des SOM-Schnittstellenmodells ist Visual SOM Studio in der Lage Objektkonfigurationen nach IEC 61850 Teil 7 zu erzeugen.
Der schriftliche Teil der Arbeit enthält eine ausführliche Dokumentation des Quellcodes der Anwendung Visual SOM Studio. Neben der Softwaredokumentation, wird das Verhalten und die Bedienung von Visual SOM Studio dokumentiert.
Der schriftliche Teil der Arbeit beleuchtet zudem die Protokollebene der Kommunikation zwischen Objekten über das .NET-Remotingframework, um das Kommunikationsverhalten von Serverobjekten des SOM-Schnittstellenmodells zu verdeutlichen.

7.2 Ideen zur Erweiterung von Visual SOM Studio

Wie in Kapitel ... dargestellt, sind nicht alle Menüpunkte der Annwendung Visual SOM Studio mit Funktionalität hinterlegt. Um Visual SOM Studio anwenderfreundlicher und ergonomischer in der Bedienung zu machen, ist es notwendig, die Funktionalitäten der einzelnen noch nicht unterlegten Menüpunkte zu realisieren.
Unabhängig von der dargestellten Differenzierung der Anwendungsbedienung ist ein Objektgenerator denkbar. In diesem Szenario ist Visual SOM Studio nicht als alleinstehende Anwendung zu betrachten. Mit Hilfe von Visual SOM Studio wird eine Objektkonfiguration projektiert und anschließend unter Verwendung des Objektgenerators die Konfiguration in ein konkretes System, wie in **[5]** eingesetzt, transformiert.
Neben der Editierfunktion zur Erstellung von Konfigurationen basierend auf dem SOM-Schnittstellenmodell ist es denkbar, die Anwendung Visual SOM Studio so zu erweitern, dass Konfigurationen in SCL-Notation (Substation Configuration Language spezifiziert in IEC 61850 Teil 6) erstellt werden können.
Dazu muss die Geschäftslogik angepasst und um mindestens ein, in Teil 6 der Norm IEC 61850 definiertes, Connectionobjekt erweitert werden.
Zusätzlich wäre die Implementierung eines Serialisierens notwendig, der auf Grundlage des Objektgraphen der Geschäftslogik von Visual SOM Studio das SCL-Format erzeugt.
Zur komfortableren Durchführung von Projektierungsaufgaben nach IEC 61850 Teil 7 im Bereich von Unterstationen, ist die Erweiterung von Visual SOM Studio dahingehend denkbar, dass ein graphischer Editor zur Eingabe der Primärtechnik in Unterstationen bereitgestellt wird. Ebenso ist es zweckmäßig, die abstrakte Darstellung der sekundärtechnischen Geräte zu konkretisieren.
Somit könnte die Anwendung Visual SOM Studio als Engineeringtool und für Projektierungen von netzleittechnischen Einrichtungen auf Grundlage der Norm IEC 61850 eingesetzt werden.

8 Literaturverzeichnis

[1] Ingo Hassler, Serienkiller: Die Fähigkeit der .NET-Serialisierung unter die Lupe genommen, dotnet Magazin 03.03

[2] The User Interface Library for NET, http://www.dotnetmagic.com/

[3] Dr. Michael Kofler, It's a kind of magic... Die Magic-Bibliothek, dotnetpro 7/2002

[4] Netzwerkmonitor von Jan-Arne Sobania, Der große Lauschangriff, dotnetpro 5/2002

[5] Ingenieurarbeit von Jan H. Arph, Grundlage der .NET-Technologie und beispielhafte Anwendung für Client/Server-Kommunikation logischer Knoten nach IEC 61850 in verteilten Stationsleitsystemen, http://fhdo.opus.hbz-nrw.de/volltexte/2003/35.

[6] Dr. Holger Schwichtenberg, www.it-visions.de, Lexikon/Glossar