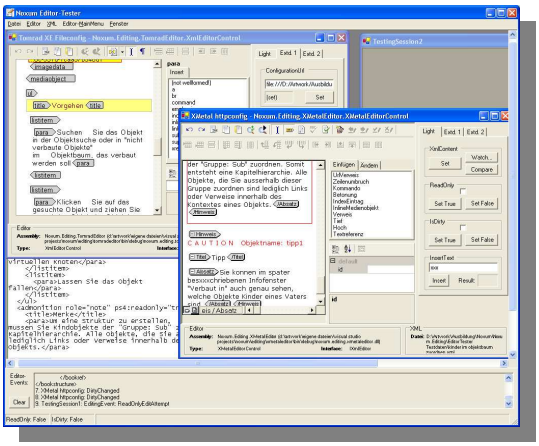
noxum

EDITOR-TESTUMGEBUNG

ANWENDUNG ZUM TESTEN VON XML-EDITOREN FÜR EIN CONTENT MANAGEMENT SYSTEM



Dokumentation einer betrieblichen Projektarbeit

für Fachinformatiker der Fachrichtung Anwendungsentwicklung

Vorgelegt von
Armin Willerding, Noxum GmbH

Für das
Städtische Kaufmännische Berufsbildungszentrum Würzburg

im Fach PTP,
Schuljahr 2005/06, Klasse 12 FI 1

noxum

Inhalt

1. Projektdefinition

1.1. Einführung: Noxum Publishing Studio

Die *Noxum GmbH* entwickelt standardisierte Software und individuelle Lösungen in den Bereichen Content Management, Cross Media Publishing, Technische Dokumentation und E-Business, und setzt dabei auf die XML-Technologie, die sich als Standard für Web- und Publishing-Lösungen durchgesetzt hat. In XML zentral und medienneutral abgelegte Inhalte können in einem weitgehend automatisierten Prozess in unterschiedlichen Medien publiziert werden, wobei die Inhalte in Design und Umfang mediengerecht aufbereitet werden können.

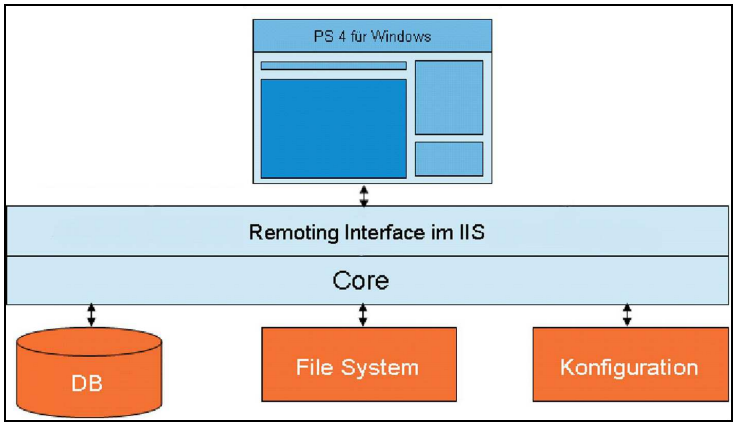


Abb. 1 – Systemarchitektur des *Noxum Publishing Studio* (vereinfacht)

Das *Noxum Publishing Studio* ist ein modular aufgebautes, auf Microsoft .NET 1.1 basiertes Content Management System (CMS) in Client-Server Architektur. Konfiguration, Datenbank und Filesystem liegen auf einem oder mehreren, i. d. R. physisch getrennten, Servern. Die Benutzeroberfläche, der Windows-basierte Rich Client, greift per Remoting-Schnittstelle auf den Server zu. Er wird lokal auf den einzelnen Arbeitsplatz-PCs installiert, während die Daten und die Kernapplikation selbst auf dem Server gehalten werden.
Über den Rich Client können alle Standard-Funktionen des *Noxum Publishing Studios* genutzt werden: Inhalte werden erfasst, nach logischen Kriterien verwaltet und in verschiedenen Medien automatisiert publiziert. Der konsequente Einsatz der XML-Technologie ermöglicht ein effizientes und flexibles Cross Media Publishing in mehreren Sprachen und Medienformaten.
Textinhalte können im *Noxum Publishing Studio* mit verschiedenen XML-Editoren erfasst werden, die vollständig in die Benutzeroberfläche integriert werden. Dabei kommen verschiedene Standardprodukte, wie *Blastradius*® *XMetaL*™, *Tomrad XE* für technische Dokumentation, individuell konfigurierte Formulareditoren für Produktdatenerfassung oder einfache Texteditoren für das simple Erfassen von Texten zum Einsatz.
Nach den Regeln einer bestimmten Document Type Definition (DTD) gelangen die Inhalte in das System. Sie werden dabei immer in logischen Informationseinheiten – sog. XML-Objekte

– gehalten. Diese können dabei verschiedene Ausprägungen haben, wie z. B. Objekttyp (Kapitel, Bild, Seite, etc.), Sprache oder Bearbeitungsbereich. Mittels dieser Objekte ist eine einfache Wiederverwendbarkeit von Textbausteinen und Grafikelementen sowie eine lückenlose Archivierung der gepflegten Daten möglich. Somit können Inhalte granular und kontextunabhängig wieder verwendet werden.

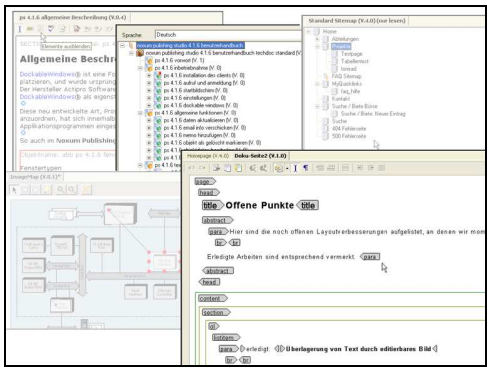


Abb. 2 – verschiedene Editoren im PS4

Auf Anfrage können auch kundenspezifische Editoren, wie Eingabemasken für Produktdaten, ein Image Map Editor usw. in die Oberfläche integriert werden. Durch das Multi Editing Konzept ist es möglich, den Autoren für jeden Objekttyp den passenden Editor anzubieten.

1.2. Projektinitiierung: Das Editor-Interface

Im Zuge der Weiterentwicklung des *Noxum Publishing Studios* (im Folgenden „PS4“ genannt[1]), für das es anfangs nur einen einzigen Editor gab (*Blastradius*® *XMetaL*™), sind inzwischen mehrere Editoren entstanden, die selbst ständigen Weiterentwicklungen unterliegen. Entsprechend der Kundenwünsche müssen sie immer wieder erweitert oder angepasst werden, und Fehler müssen behoben werden. Um diese Entwicklung zu vereinheitlichen, wurde eine Schnittstelle geschaffen – `IXmlEditor` –, welche von allen Editoren implementiert werden muss. Ausschließlich über dieses Interface wird ein Editor konfiguriert, gesteuert, und überwacht. Auch die Übergabe eines XML-Objekts an den Editor und die Entgegennahme eines solchen erfolgt allein über die Schnittstelle. Vom PS4 wird, abhängig von dessen Konfiguration und des Typs des zu editierenden XML-Objekts, der jeweils passende Editor, als `IXmlEditor` typisiert, per Late Binding im Rich Client eingebunden.

Das Interface liegt heute in 2 Versionen vor: Zusätzlich zum normalen `IXmlEditor` gibt es eine „abgespeckte“ Version, `IXmlEditorLight`, die aus dem ersten Entwurf des Interfaces hervorgegangen ist und später in diesen Namen umbenannt wurde. Es enthält nur die nötigsten Funktionen und ist für Testzwecke, bzw. besonders einfach gestrickte Editoren

vorgesehen. `IXmlEditor` ist von `IXmlEditorLight` abgeleitet. Eine detaillierte Beschreibung des Interfaces ist in Anhang ▶ A1 zu finden.
Die Editor-Schnittstelle bietet die Erleichterung, Editoren unabhängig programmieren zu können, ohne sich um die internen Abläufe des PS4 zu kümmern. Der Editor muss sich z. B. seine Daten nicht mehr selbst aus der Umgebung holen; auf Ereignisse der Umgebung muss er nicht selbständig reagieren. Diese Vorteile ermöglichen es prinzipiell, die Entwicklung einzelner kundenspezifischer Editoren an Dritthersteller abzugeben, oder die Editoren auch in anderen Produkten einzusetzen.
Dennoch bleibt das Entwickeln von Editoren, bedingt durch den Umstand, dass hierbei sehr viele Testläufe gemacht werden müssen, überaus umständlich und ineffizient. Die Analyse der im folgenden Abschnitt beschriebenen Probleme führte schließlich zur Initiierung des vorliegenden, unternehmensinternen Projekts.

1.3. Ist-Analyse: Editor-Tests im PS4

1.3.1. Ausgangssituation

Um an einem Arbeitsplatz-PC eines Entwicklers einen Editor im PS4 zum Laufen zu bringen, sind eine Reihe von Voraussetzungen notwendig. Zunächst müssen mindestens die folgenden 3 Komponenten vorhanden sein.

- Rich Client des PS4 (muss lokal installiert sein)
- PS4-Server (erfordert IIS)
- Datenbank (erfordert MS SQL-Server)

Zum Server muss entweder eine Netzwerkverbindung bestehen, oder er muss ebenfalls lokal installiert werden, was einen IIS-Webserver erfordert. Entsprechendes gilt für die Datenbank, die wiederum einen Microsoft SQL Server benötigt. Zudem muss das komplette PS4-System inklusive Server, IIS, Datenbank, und Remoting-Schnittstelle korrekt, und zum jeweiligen Datenbestand passend, konfiguriert werden. Dies setzt eine recht genaue Kenntnis des Systems und des Ineinandergreifens seiner Komponenten voraus. Schließlich muss die Datenbank geeignete Testdaten enthalten.
Hat der Entwickler einen Editor kompiliert, so legt er anschließend die erzeugte DLL-Datei in das Programmverzeichnis des PS4-Clients, damit dieser den Editor findet und einbinden kann. Zum Testen wird der Client gestartet. Sobald dieser mit dem Abarbeiten von gewissen Caching-Vorgängen fertig ist, kann ein Objekt gesucht, ausgewählt und zum Editieren mit dem gewünschten (falls die Konfiguration hier eine Auswahl erlaubt) Editor geöffnet werden.

[1] PS4 steht für Publishing Studio, Version 4.x

1.3.2. Problemanalyse

Die obige Beschreibung der Voraussetzungen macht bereits das Hauptproblem deutlich, nämlich der erhebliche Aufwand, der zum Testen nötig ist. Die Nachteile sind im einzelnen:

- Hoher Installationsaufwand
- Bei verteiltem System Netzwerkinfrastruktur und Serverhardware erforderlich
- Hoher Konfigurationsaufwand
- Bei eigenhändiger Konfiguration detaillierte Kenntnisse des PS4-Systems notwendig
- Geeignete Testdaten müssen im System eingepflegt sein
- Bei jedem Testlauf viele Handgriffe notwendig
- Bei jedem Testlauf hohe Wartezeiten

Doch es zeigen sich überdies auch schwerwiegende Probleme durch mangelnde Flexibilität:

- Testen beliebiger XML-Daten nicht möglich
  Nur die Objekte, die im System gespeichert sind, können verwendet werden. So kann das Verhalten des Editors insbesondere bei XML-Daten, die fehlerhaft sind oder bestimmte, vom System nicht erlaubte Eigenschaften haben, nicht getestet werden.
- Keine Möglichkeit der Kontrolle über das editierte XML
  Es kann nicht geprüft werden, was der Editor an den XML-Daten geändert hat.
- Kein beliebiger Zugang zu allen Editorfunktionen und Propertys jederzeit
- Keine Kontrolle über Events, die der Editor auslöst

1.4. Soll-Konzept: Unabhängige Testumgebung

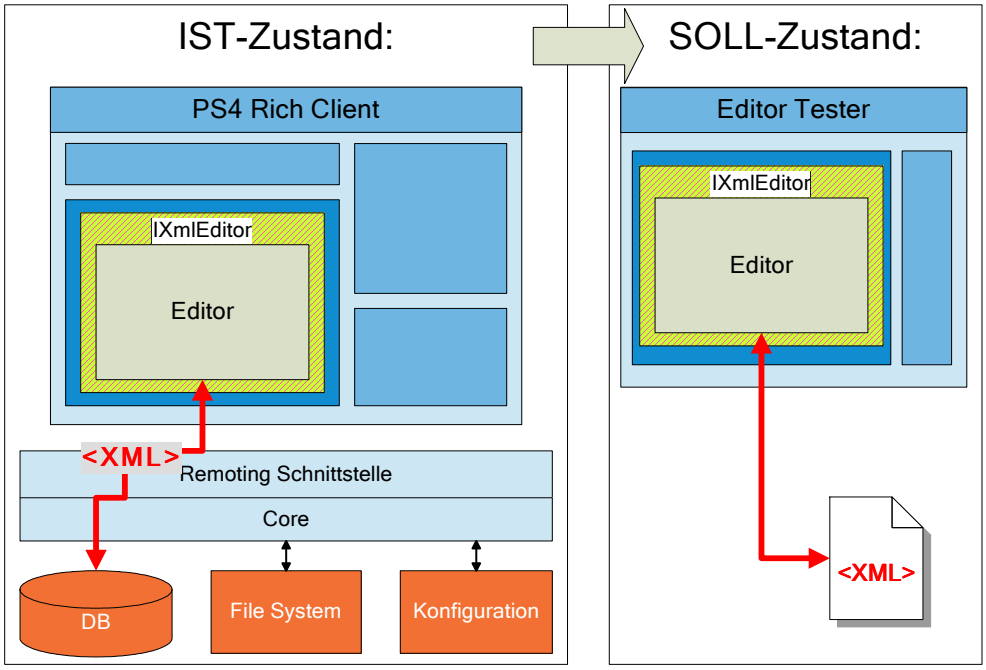


Abb. 3 – Lösungsansatz: unabhängige Testumgebung

1.4.1. Lösungsansatz

Gefordert wird eine schlanke, vom PS4 unabhängige Testumgebung als Insellösung, mit der Editoren, analog zur Funktionsweise im PS4-Client, per Late Binding eingebunden, und über das Editor-Interface angesprochen werden können. XML-Daten sollen direkt aus XML-Dateien entnommen und wieder hineingespeichert werden können, wie in Abb. 3 dargestellt. Mit Hilfe des Testprogramms soll man in der Lage sein, schnell und effizient Editoren systematisch durchzutesten, ohne die oben aufgeführten Probleme zu haben.

1.4.2. Projektziele

1.4.2.1. Ablauf

Ein Editor wird (wie im PS4) zur Laufzeit des Testprogramms eingebunden, indem der Benutzer die entsprechende Assembly-Datei auswählt. Findet das Programm darin einen Editor, so erzeugt es daraus ein Control-Objekt und zeigt dieses an. Befinden sich mehrere Editoren in der Assembly, so wird der Benutzer zur Auswahl des gewünschten aufgefordert. Auch das Laden der XML-Daten erfolgt durch direkte Benutzerinteraktion: Nach Auswahl einer XML-Datei wird diese vom Programm ausgelesen und der Inhalt über das Interface an den Editor übergeben. Will der Benutzer speichern, so wird der `XmlContent` vom Editor entgegengenommen und in die Datei zurück geschrieben. Auch alle übrigen Funktionalitäten, die das Interface anbietet, sollen dem Benutzer unmittelbar über Buttons und ähnliche Steuerelemente zur Verfügung stehen. Events, die im Editor ausgelöst werden, sollen sofort gemeldet werden.

1.4.2.2. Ergonomie

Folgende Anforderungen an die Benutzerfreundlichkeit und Ergonomie werden gestellt:

- Übersichtlichkeit: Alle wichtigen Statusinformationen sollen auf einen Blick verfügbar sein; ebenso die wichtigsten Funktionen
- In möglichst kurzer Zeit soll die Umgebung gestartet und ein Editor zum Laufen gebracht werden.
- möglichst wenige Mausklicks/Tastatureingaben erforderlich

1.4.2.3. Qualität

Die folgenden Qualitätsanforderungen werden gestellt:

- Das Programm soll möglichst stabil laufen, damit sich Entwickler ganz auf die Editoren konzentrieren können.
- Exceptions, die ein Editor werfen könnte, sollen ohne Absturz abgefangen und aussagekräftige Fehlermeldungen ausgegeben werden

1.4.3. Zusatzfeatures

Neben den geforderten Problemlösungen sind die folgenden Features wünschenswert:

- Eine Funktion, die das dem Editor übergebene XML mit dem XML vergleicht, das er zurückliefert: Damit kann z. B. sichergestellt werden, dass der Editor ein XML-Dokument nicht antastet, wenn der Benutzer nicht editiert.
- Durch Mehrfenstertechnik sollen mehrere Editor-Instanzen (auch verschiedene) gleichzeitig in einzelnen Dokumentenfenstern offen gehalten werden können. Jedes dieser Dokumentenfenster hat dann seine eigene Sammlung an Informationen über den getesteten Editor, die Konfiguration des Editors, den Speicherort der Assembly, usw. Diese Informationen soll man, um eine Testsitzung schnellstmöglich wieder herzustellen, in eine Datei speichern und wieder laden können.

1.4.4. Systemvoraussetzungen

Das Testprogramm wird entsprechend der Technologie der Editoren als Windows-Forms-Anwendung in Microsoft .NET 1.1 entwickelt. Zum Betrieb der Anwendung wird daher ein PC mit Microsoft Windows 2000 oder XP, sowie installierter Microsoft .NET 1.1 Laufzeitumgebung benötigt. Die Anforderungen an Prozessorleistung und Arbeitsspeicher richten sich nach den zu testenden Editoren.

2. Projektplanung

2.1. Inhaltliche Strukturierung

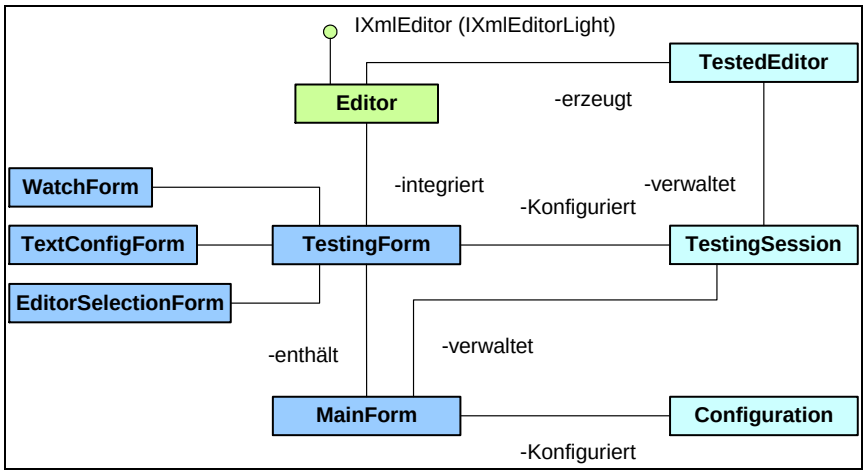


Abb. 4 – Grobkonzept des Klassenmodells (Siehe auch Anhang ▶ A2)

2.1.1. Klassendesign

Es wurden 8 Klassen entworfen, davon sind 5 von System.Windows.Forms.Form abgeleitet, in Abb. 4 blau dargestellt. Eines dieser Forms, `TestingForm`, ist die eigentliche Testplattform. Es bindet den zu testenden Editor, der durch die Klasse `TestedEditor` aus der Assembly entnommen und umhüllt wird, ein, und bietet dem Benutzer Zugriff auf alle wesentlichen Abläufe und die Testfunktionen. Im Hauptformular, `MainForm`, können beliebig viele Instanzen von `TestingForm` als „Dokumentenfenster“ angezeigt und verwaltet werden. In jedem dieser offenen Dokumentenfenster findet eine „Testsitzung“ statt, welche durch die Klasse `TestingSession` vertreten wird. In einem `TestingSession`-Objekt werden alle relevanten Informationen zu einer Testsitzung aufbewahrt. Mit seiner Hilfe kann die Sitzung in eine Sessiondatei abgespeichert und wieder geladen werden. Die `Configuration`-Klasse, dient dazu, benutzerspezifische Einstellungen des `MainForm` zu speichern und wieder herzustellen.
Ein detaillierteres Diagramm der Klassenbeziehungen mit Kurzbeschreibung der einzelnen Klassen ist in Anhang ▶ A2 zu finden.

2.1.2. Benutzeroberflächendesign

2.1.2.1. TestingForm

Das `TestingForm` wird wie folgt aufgeteilt: Ein Hauptmenü stellt die Hauptfunktionen wie Editor öffnen und schließen, XML-Datei öffnen, schließen und speichern, und den Zugriff auf die zuletzt verwendeten Editoren und XML-Dateien zur Verfügung. Bietet der Editor ein MainMenu an seiner Schnittstelle an, so wird dieses hier angehängt.
Im linken Bereich des Fensters, der den Großteil seiner Fläche ausmacht, wird der Editor eingebunden. Am rechten Rand befindet sich eine Sidebar, mit der die Editorfunktionen gesteuert werden. Die einzelnen Steuerelemente hierfür sind auf mehrere Seiten eines TabControls verteilt. Im unteren Bereich werden ständig Informationen über den Editor und die aktuell verwendete XML-Datei angezeigt.
Unterstützt wird das `TestingForm` durch die 3 zusätzlichen Forms `EditorSelection-Form`, `TextConfigForm` und `WatchForm` (Funktionsbeschreibungen siehe Anhang ▶ A2):

- **`EditorSelectionForm`** ist ein modaler Dialog, in dem die in einer geöffneten Assembly gefundenen Editoren mit Klassenname und Interfacename in einer ListView zur Auswahl angeboten werden.
- **`TextConfigForm`** ist ein modaler Dialog, der eine mehrzeilige Textbox zum Bearbeiten eines Strings anzeigt.

- **`WatchForm`** ist ein skalierbares Toolwindow, das den XML-Text in einer schreibgeschützten, mehrzeiligen Textbox anzeigt. Über Buttons und Checkboxen am oberen Rand können Funktionen zum Aktualisieren des Inhalts, zum Umschalten zwischen Normal- und Zeilenumbruchansicht und weitere Funktionen bedient werden.

2.1.2.2. MainForm

Um die Mehrfenstertechnik zu realisieren, wird das `MainForm` als MDI-Container benutzt. Die `TestingForm`-Fenster stellen die MDI-Children dar. Im Menü „Datei“ des Hauptmenüs stehen die Funktionen zum Neuanlegen, Öffnen und Schließen von TestingSessions, sowie der Aufruf der zuletzt benutzen Sessiondateien zur Verfügung. Ein Fenstermenü steuert die MDI-Children. Das Menü des aktiven `TestingForm` wird automatisch an das MainMenu angehängt.
Im unteren Bereich des Fensters werden Editor-Events gemeldet. Hier befindet sich eine mehrzeilige Textbox, in der sämtliche Editor-Events aller offenen Testsitzungen untereinander mit Benennung der jeweiligen Session und der Eventdaten eingetragen werden. In der Statusleiste werden die Eigenschaften `ReadOnly` und `IsDirty` des Editors der aktiven `TestingForm` angezeigt.

2.1.3. Datenablage

An 2 Stellen im Programm werden Daten als XML-Daten generiert und in Dateien abgelegt:

- Die Klasse `TestingSession` schreibt beim Aufruf der Methode `SaveAs (string filepath)` ihre Informationen in ein XML-Dokument und speichert dieses als Textdatei unter einem vom Benutzer frei gewählten Namen an einem frei gewählten Ort. Als Dateierweiterung wird **`"*.tsess"`** für „TestingSession“ festgelegt.
- Die Klasse `Configuration` schreibt ihre Informationen in ein XML-Dokument und speichert dieses als Textdatei unter dem Namen **`userSettings.config`** in einem Ordner unter „Anwendungsdaten“ des aktuellen Benutzerprofils. Den Pfad zu diesem Ordner gibt das Betriebssystem über den Aufruf `System.Environment.SpecialFolder.ApplicationData` vor, dem noch der Name der Testanwendung angehängt wird.

Die genaue Spezifikation der abgelegten XML-Dateien inkl. Ihrer Strukturen und der Bedeutung der einzelnen Elemente ist in Anhang ▶ A3 zu finden.

2.2. Zeitliche Strukturierung

Das Projekt wird stufenweise umgesetzt. Im Folgenden sind die einzelnen geplanten Arbeitspakete mit den jeweiligen Zwischenzielen und dem geplanten Zeitaufwand aufgeführt:

Arbeitsschritt	Std.
1. Implementierung von `TestedEditor`; Aufsetzen von `TestingForm` als Hauptklasse und Designen der Oberfläche; Implementieren der Funktionen zum Zugriff auf Assembly- und XML-Dateien und deren Verarbeitung *Ziel*: Laden und Anzeigen eines Editors im `TestingForm`; Laden von XML-Inhalt und Anzeigen desselben im Editor	4
2. Implementieren des `EditorSelectionForm` *Ziel*: Gezieltes Instanziieren einer gewünschten Editorklasse bei Vorhandensein mehrerer Editoren in einer Assembly	1
3. Ausprogrammieren aller Testfunktionen in `TestingForm`; Implementieren des `TextConfigForm` *Ziel*: Durchtesten aller Editorfunktionen (außer Events)	6
4. Implementieren des `WatchForm` *Ziel*: Beobachten des XML-Inhalts	1
5. Aufsetzen der `MainForm` als Hauptklasse und MDI-Container *Ziel*: Erzeugen neuer `TestingForms` und Anzeigen derselben im `MainForm`	1
6. Implementieren von `TestingSession`; Verlagerung der bisher im `TestingForm` gehaltenen Sitzungsdaten in die `TestingSession` *Ziel*: Herstellen der bisherigen Funktionalität unter Verwendung der `TestingSession`	4
7. Ausprogrammieren des MainForm inkl. aller Zugriffe auf `TestingSession` und der Dateizugriffe auf die TestingSession-Dateien. *Ziel*: Speichern und wieder Öffnen von TestingSessions; Anzeige der Event-Meldungen und der Flags `ReadOnly` und `IsDirty`	3
8. Implementieren von `Configuration`; Ergänzung von `MainForm` um die Zugriffe auf `Configuration` *Ziel*: Abspeichern und Laden der benutzerspezifischen Konfiguration	2

Tabelle 1 – Projektablaufplanung

Der geplante Gesamtaufwand beträgt 22 Stunden.

3. Projektrealisierung

Das Projekt wurde in C# unter Microsoft Visual Studio.NET 2003 entwickelt. Als Entwicklungsrechner diente ein PC mit AMD Duron Prozessor, 900 MHz, 640 MB Arbeitsspeicher und 40 GB Festplattenspeicher.
Für die Testanwendung wurde im Namespace `Noxum.Editing`, das auch die Editoren enthält, ein eigenes VisualStudio-Projekt mit dem Namen `Noxum.Editing.Test-Environment` angelegt. Dieses Projekt enthält alle Klassen der Testanwendung, die auch den gleichnamigen Namespace verwenden.

Das Projekt verwendet die folgenden Referenzen:

- **`System`, `System.Windows.Forms`, `System.Drawing`, `System.XML`**
  Grundlegende .NET Framework API-Komponenten; benötigt für Formulare und XML-Funktionalität
- **`Noxum.Editing.Interfaces`**
  Projekt, welches die Editor-Interfaces und die speziell von `IXmlEditor` verwendeten Klassen, Delegaten, und Enumerationen enthält. Wird im Projekt von `TestingForm` und `TestedEditor` verwendet.
- **`Noxum.Editing.Utilities`**
  Dieses Projekt beinhaltet die Klasse `XpathUtility`, mit deren Hilfe sich ein `IXPathNavigable` in ein `XmlDocument` umwandeln lässt. Sie wird im Projekt von der Klasse `TestingForm` in der Methode `FetchXml` verwendet, mit der der `XmlContent` vom Editor geholt wird.
- **`Microsoft.XmlDiffPatch`**
  Von Microsoft kostenlos verfügbare .NET-Komponente. Sie beinhaltet die Klasse `XmlDiff`, mit der einige mächtige XML-Vergleichsfunktionen möglich sind. Sie wird im Projekt von `TestingForm` in der Methode `CompareXmls` verwendet.

3.1. Realisierung der Benutzeroberfläche

Die Graphische Benutzeroberfläche der einzelnen Forms wurde mit Hilfe des Windows Form Designers von Visual Studio 2003 gestaltet. Die Ergebnisse sind anhand der Screenshots in Anhang ▶ A4 zu sehen.

3.2. Implementierung der Klassen

Es wurden gemäß der Projektplanung 8 Klassen in C# programmiert. Entstanden sind 8 C#-Dateien mit einem Umfang von insgesamt ca. 4.380 Codezeilen (inkl. Leerzeilen). Davon sind ca. 1.340 Zeilen vom Windows Form Designer generiert worden. In den 8 Klassen wurden insgesamt 163 Methoden implementiert.

Klasse	Methoden	Zeilen (Circa-Angabe)	davon selbst programmiert
`MainForm.cs`	28	660	440
`TestingForm.cs`	69	2070	1270
`EditorSelectionForm.cs`	5	210	90
`WatchForm.cs`	10	280	140
`TextConfigForm.cs`	4	140	80
`TestedEditor.cs`	15	250	250
`TestingSession.cs`	16	370	370
`Configuration.cs`	16	400	400
Summe	**163**	**4380**	**3040**

Tabelle 2 – Codeumfang

In Anhang ▶ A5 sind die vollständigen UML-Diagramme der einzelnen Klassen in der Reihenfolge der Projektablaufplanung abgebildet.

4. Projektergebnis

4.1. Soll-Ist-Abgleich

Vergleicht man die Eigenschaften und Fähigkeiten der erstellten Anwendung mit dem anfangs erarbeiteten Soll-Konzept, so zeigt sich, dass die dort aufgestellten Projektziele allesamt weitgehend umgesetzt wurden. Die erreichten Ziele sind im Einzelnen:

- Die Anwendung läuft unabhängig von PS4, Datenbanken und jedweden Netzwerkressourcen. (Selbstverständlich kann jedoch auch eine HTTP-URI zur Konfiguration des Editors übergeben werden.)

Ablaufziele:

- Benutzer kann Assembly-Datei frei wählen
- Der in der Assembly gefundene Editor wird (zur Laufzeit) automatisch eingebunden.
- Bei mehreren Editoren kann der Benutzer den Editor wählen.
- Benutzer kann XML-Datei frei wählen
- Editiertes XML kann wieder in die Datei zurück geschrieben werden (auch in eine neue Datei).
- Alle Funktionen, die das Interface vorsieht, können unmittelbar angesteuert werden. Ausnahmen: `InsertMarkup`, `GetAttributeFromCurrentElementOrAncestor` und `StringTable`
- Alle Editor-Events werden direkt gemeldet.

Ergonomieziele:

- Statusinformationen über den Editor und die XML-Datei sind während einer Testsitzung ständig sichtbar.
- Die Anwendung startet schnell.
- Sind erst eigene TestingSessions gespeichert worden, so genügt nach dem Start ein Mausklick auf eine der zuletzt verwendeten TestingSessions im Menü Datei, um diese zu starten und den Editor sofort einzubinden.
- Die in der Session zuletzt verwendeten XML-Dateien sind ebenfalls mit einem Mausklick zu erreichen. Dies ist von Vorteil, da i. d. R. zum Testen eines Editors sehr oft die selben XML-Dateien wieder verwendet werden

Qualitätsziele:

- Das Programm wurde mit größter Sorgfalt geplant und schrittweise, unter Kontrolle der aufgestellten Zwischenziele, erstellt. Dadurch wurde schwerwiegenden Stabilitätsmängeln und Programmierfehlern weitgehend vorgebeugt. Auftretende Fehler rühren daher meistens vom Editor her.
- Eine sorgfältige Fehlerbehandlung meldet eigene Exceptions, sowie die des Editors sauber dem Benutzer, so dass man die Quelle leicht erkennen kann.

Zusatzfeatures:

- Die Testumgebung verfügt über eine XML-Vergleichsfunktion
- Die Mehrfenstertechnik ist realisiert worden. Mehrere Editoren können somit gleichzeitig in einzelnen Sitzungen getestet werden. Jede Sitzung kann abgespeichert und wieder hergestellt werden.

4.2. Abschlussanalyse

Das Projekt ist insgesamt sehr erfolgreich verlaufen. Trotz der im vorigen Punkt genannten Ausnahmen bei den umgesetzten Editor-Testfunktionen wurden die Ziele für die derzeitigen Anforderungen im Betrieb mehr als hinreichend umgesetzt. Die drei Funktionen, die noch nicht testbar sind, werden bis jetzt auch in keinem Editor verwendet. Sollten sie einmal zum Einsatz kommen, können die entsprechenden Testfunktionen mit sehr geringem Aufwand in das `Testform` eingebaut werden.
Es wurde bewusst darauf verzichtet, alle nur denkbaren Features mit einzuplanen, um das Projekt mit Blick auf seine Wirtschaftlichkeit nicht unverhältnismäßig aufzublähen. Die Anwendung bietet jedoch an vielen Stellen noch Raum für Erweiterungen. Ein Bereich der besonders ausbaufähig ist, ist das `WatchForm` in Verbindung mit der XML-Vergleichsfunktion. Hier könnten die weit reichenden Möglichkeiten von `Microsoft.XmlDiffPatch` z. B. für eine Echtzeit-Änderungsverfolgung mit farblicher Markierung des geänderten XML-Inhalts genutzt werden.
Somit erhebt die Editor-Testumgebung keinerlei Anspruch auf Endgültigkeit. Im Zuge der Weiterentwicklung der Editoren kann und soll sie in Zukunft je nach Bedarf erweitert werden.

Anhang

A1. Schnittstellenspezifikation: IXmlEditorLight und IXmlEditor

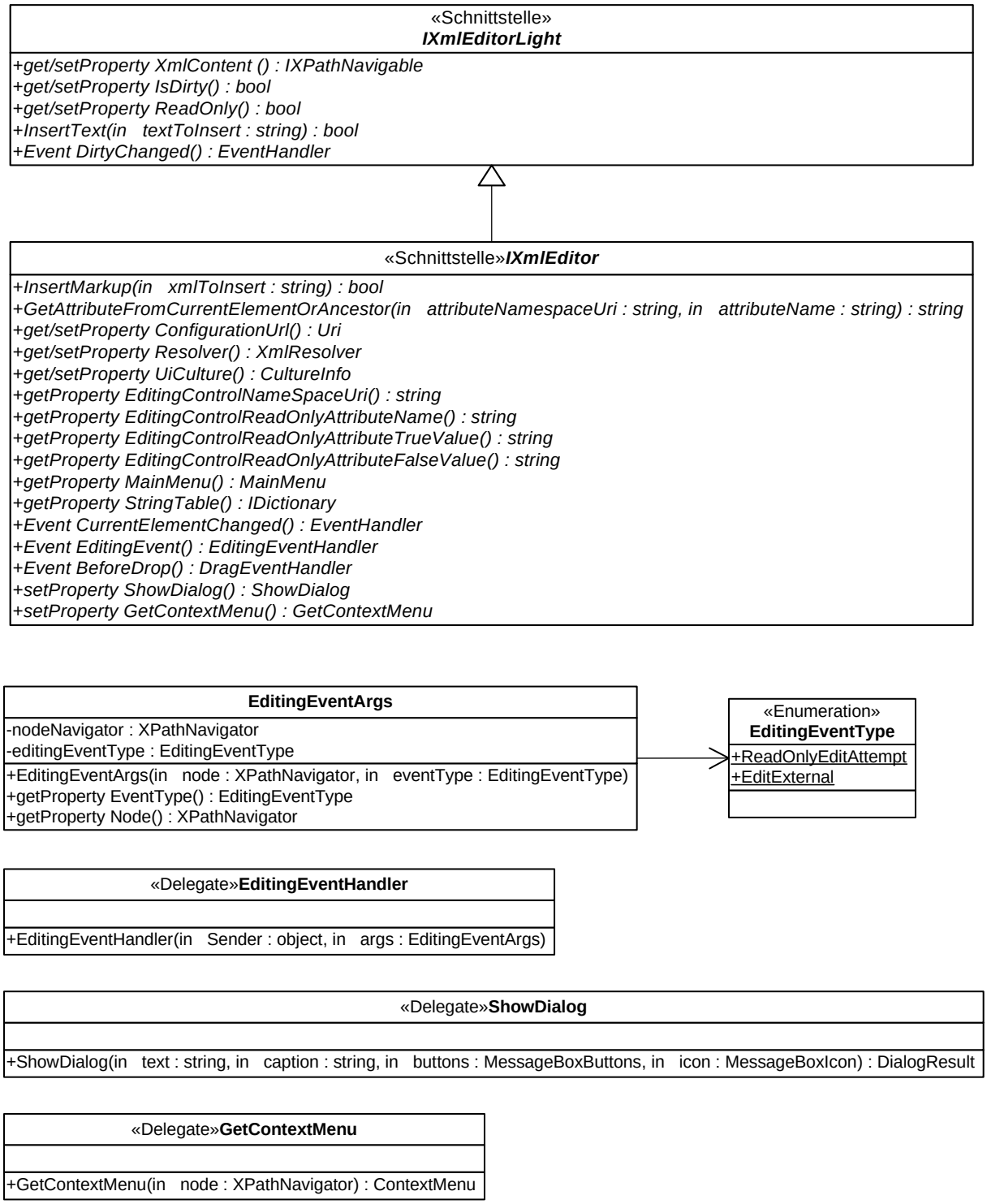


Abb. 5/6 – Die Editorschnittstellen und die von ihnen verwendeten Typen

Es folgen die Erläuterungen der einzelnen Member der beiden Interfaces. Da diese in C# entwickelt wurden, wird auch hier die C#-Notation angewendet.

Erläuterungen zu IXmlEditorLight	
Member	**Beschreibung**
`IXPathNavigable` **`XmlContent`** `{get; set;}`	Zuweisen bzw. abholen des zu editierenden bzw. editierten XML-Inhalts (z. B. als XmlDocument oder XPathNavigator). Es steht dem Editor frei, sich eine innere Kopie der Daten anzulegen, d. H. er editiert nicht die Ihm zugewiesene Instanz (z.B. eines XmlDocuments)
`bool` **`IsDirty`** `{get; set;}`	der Editor gibt eine Information darüber, ob sich der XML-Content seit dem letzten Zuweisen geändert hat. Wenn der besitzende Container den momentanen Stand speichert, sollte er IsDirty auf false setzen.
`bool` **`ReadOnly`** `{get; set;}`	Flag für einen (globalen) ReadOnly-Modus
`bool` **`InsertText`** `(string textToInsert);`	Fordert den Editor auf, an der aktuellen Position (Cursorposition bzw. Selektion) den angegebenen Text einzufügen. Kann z.B. für Einfügen von Sonderzeichen aus externem Dialog/Tool verwendet werden Rückgabe True, wenn Einfügen erfolgreich
`event EventHandler` **`DirtyChanged`**`;`	Dieses Event soll der Editor jedes Mal auslösen, wenn sich `IsDirty` ändert.

Erläuterungen zu IXmlEditor	
Member	**Beschreibung**
`bool` **`InsertMarkup`** `(string xmlToInsert);`	Fordert den Editor auf, an der aktuellen Position (Cursorposition bzw. Selektion) das angegebene XML-Markup einzufügen. Kann z. B. in Reaktion auf BeforeDrop verwendet werden.
`string` **`GetAttributeFromCurrentElementOrAncestor`** `(string attributeNamespaceUri, string attributeName);`	Liefert zu dem zur aktuellen Cursorposition gehörenden Element den Wert des angegebenen Attributs zurück
`Uri` **`ConfigurationUrl`** `{get; set;}`	Hier wird dem Editor eine URI (URL) übergeben, wo er seine Konfiguration beziehen kann (z.B. DTD, CSS, Editor-eigene Konfigurationsdateien)
`XmlResolver` **`Resolver`** `{get; set;}`	Möglichkeit, dem Editor einen (customized) Resolver zu übergeben, über den dieser benötigte Ressourcen (Konfiguration, Bilder , ...) ziehen kann.
`CultureInfo` **`UiCulture`** `{get; set;}`	CultureInfo des Editors (z. B. für Spracheinstellung der Buttonbeschriftungen)
`string` **`EditingControlNameSpaceUri`** `{get;}`	Die Namespace-Uri von XML-Nodes, aus denen der Editor gewisse Meta-Informationen über das editierte Dokument (z. B. Read-Only-Abschnitte) herausliest.
`string` **`EditingControlReadOnlyAttributeName`** `{get;}`	Name des Attributs, durch das der Editor den Read-Only-Status einzelner XML-Elemente erkennen kann.
`string` **`EditingControlReadOnlyAttributeTrueValue`** `{get;}`	Schreibweise des True-Wertes für das ReadOnly-Attribut
`string` **`EditingControlReadOnlyAttributeFalseValue`** `{get;}`	Schreibweise des False-Wertes für das ReadOnly-Attribut

Erläuterungen zu IXmlEditor (Fortsetzung)	
Member	**Beschreibung**
`MainMenu` **`MainMenu`** `{get;}`	Der Editor kann ein MainMenu anbieten, das die Umgebung in Ihr Menü integrieren kann, wenn der Editor den Fokus bekommt
`IDictionary` **`StringTable`** `{get;}`	Der Editor kann sein StringTable für Meldungen, etc. zur Verfügung stellen, so dass die Umgebung diese Strings ggf. bearbeiten/modifizieren kann. Kann auch dazu verwendet werden, herauszufinden, welche Art von Meldungen der Editor überhaupt kennt.
`event System.EventHandler` **`CurrentElementChanged`**`;`	Event, das ausgelöst werden soll, wenn der Cursor das aktuelle Element verlässt
`event EditingEventHandler` **`EditingEvent`**`;` **`EditingEventArgs.editingEventType`** `==`	Steht für verschiedene Ereignisse. Um was es sich im Einzelfall beim Auslösen dieses Events handelt, wird in den **`EditingEventArgs`** im **`EditingEventHandler`** näher spezifiziert. Sie enthalten einen XPathNavigator und die Enumeration **`EditingEventType`**. Letztere kennt derzeit zwei Werte:
`EditingEventType.ReadOnlyEditAttempt`	Es wurde versucht, einen schreibgeschützten Bereich zu editieren oder das `ReadOnly`-Flag war gesetzt.
`EditingEventType.EditExternal`	Ein Teil des Dokuments, der z. B. eine externe Referenz oder ein Link sein kann, soll in einem eigenen Editor geöffnet werden.

`event`
`System.Windows.Forms.`
`DragEventHandler`
`BeforeDrop``;`
	Wenn der Editor ein Drop-Ereignis empfängt, soll er seinerseits, bevor er den gedroppten Inhalt einfügt, dieses Event auslösen und ihm die ursprünglichen DragEventArgs mitgeben. Durch abbonieren dieses Events in der Umgebung kann das Drop-Verhalten beeinflusst werden.
	Falls die entsprechende Einfügeaktion dann außerhalb des Editors durchgeführt wird (per `InsertText` oder `InsertMarkup`), sollte danach DragEventArgs.Data gelöscht werden, damit der Editor das Event nicht mehr selbst handelt.

`ShowDialog` **`ShowDialog`** `{set;}`	Das Anzeigen und Auswerten von Meldungsdialogen des Editors kann von der Umgebung des Editors übernommen werden. Dazu muss die Umgebung den **Delegat `ShowDialog`** auf eine eigene Methode setzen, die die Dialogparameter empfängt und ein DialogResult zurückliefert.
`GetContextMenu` **`GetContextMenu`** `{set;}`	Kontextmenüs, die der Editor im Dokument erscheinen lässt, können von der Umgebung, abhängig vom angeklickten Dokumentknoten, vorgegeben werden. Dazu setzt die Umgebung den **Delegat `GetContextMenu`** auf eine eigene Methode, die einen XPathNavigator empfängt und ein ContextMenu zurückliefert.

A2. Klassenmodellentwurf mit Klassenbeziehungen

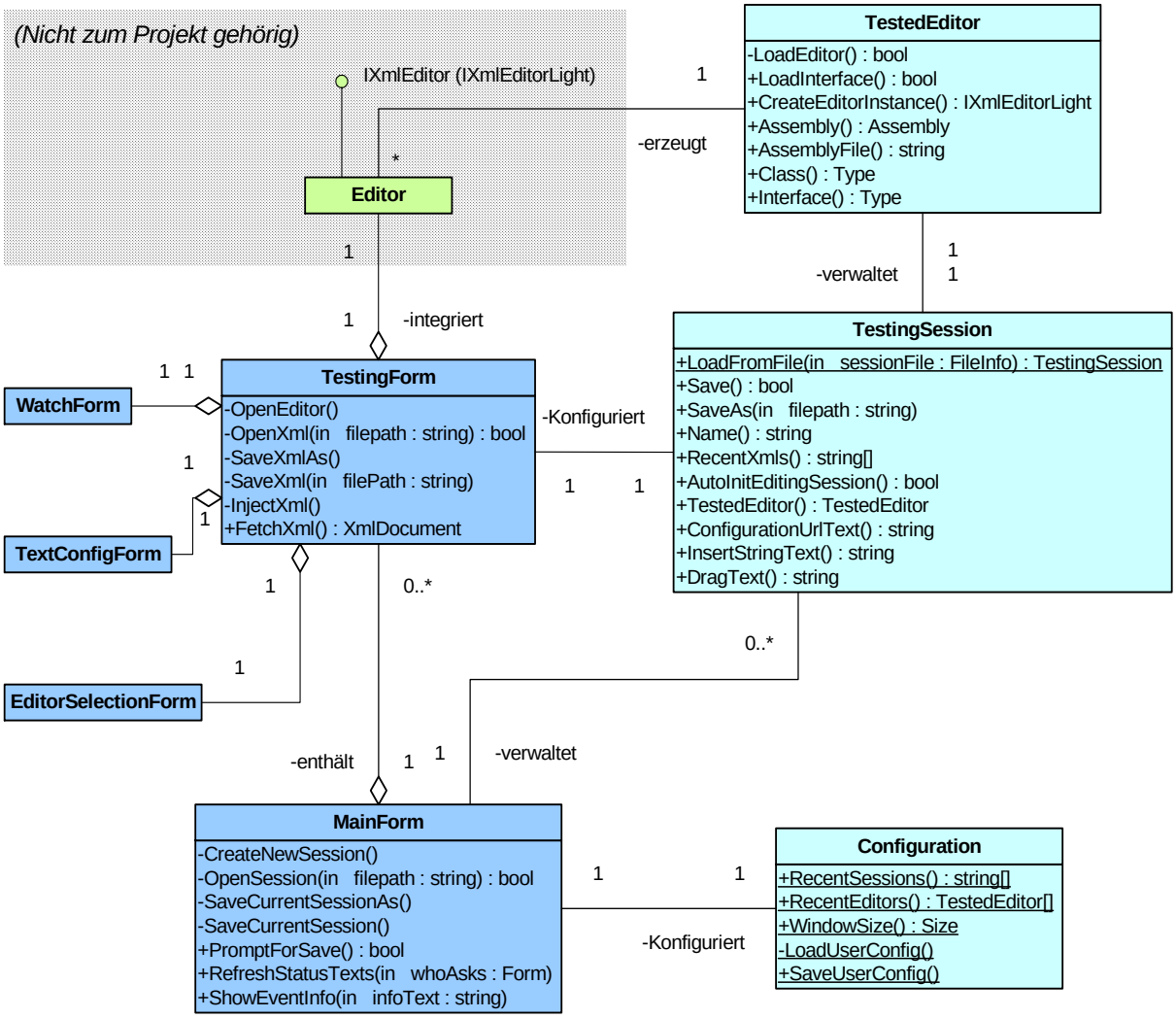


Abb. 7 – Die Klassen des Projekts und ihre Beziehungen. Dargestellt sind der Übersicht halber nur die wichtigsten Methoden

`MainForm`

Haupt- und Startklasse der Windows Forms-Anwendung. Sie fungiert als MDI-Container und kann beliebig viele TestingForms gleichzeitig als MDI-Children beinhalten. Sie verwaltet auch die TestingSessions, die sie als Dateien speichern, laden und neu anlegen kann.

`Configuration`

Konfiguriert das `MainForm`. Benutzerspezifische Informationen über die zuletzt geöffneten Dateien und die Fenstergröße werden in statischen Propertys gehalten. Beim Beenden der Anwendung speichert sie automatisch eine XML-Konfigurationsdatei im Ordner „Anwendungsdaten“ im Benutzerprofil des aktuellen Windows-Benutzers. Beim Starten lädt sie die Datei automatisch wieder, wenn vorhanden.

TestingForm

Fenster, in der eine Test-Sitzung stattfindet. Es wird als MDI-Child im `MainForm` eingebunden. Es besitzt die drei weiteren Forms `WatchForm`, `TextConfigForm` und `EditorSelectionForm`. Im `TestingForm` wird ein Editor aufgerufen, integriert und angesteuert. XML-Dateien werden geladen und gespeichert, deren Inhalt dem Editor übergeben und von ihm abgerufen.

TestingSession

Konfiguriert das `TestingForm`. Es hält verschiedene Einstellungen für eine Test-Sitzung, die Pfade der zuletzt geöffneten XML-Dateien und stellt einen `TestedEditor` zur Verfügung. Die `TestingSession` kann über Load- und Save-Methoden von sich selbst eine Datei im XML-Format erstellen aus der sie sich später wieder herstellen kann.

TestedEditor

Wird von einer `TestingSession` verwaltet und hält Informationen über einen Editor, wie Assembly, Assemblypfad, Editor-Klasse und das Interface, das vom Editor implementiert wird. Sind alle nötigen Informationen vorhanden, kann `TestedEditor` eine Instanz des Editors vom Typ `IXmlEditorLight` oder `IXmlEditor`, je nach Implementierung, erzeugen.

Editor

Ein XML-Editor, der das Interface `IXmlEditorLight` oder `IXmlEditor` implementiert. Diese Editoren wurden und werden außerhalb dieses Projektes entwickelt und sind i.d.R. von System.Windows.Forms.Control abgeleitet. Das `TestingForm` integriert ihn als Control auf seiner Oberfläche.

EditorSelectionForm

Wird vom `TestingForm` angezeigt. Falls eine Assembly mehrere Klassen aufweist, die das Editor-Interface implementieren, soll der Benutzer hier die gewünschte Klasse auswählen.

WatchForm

Wird vom `TestingForm` angezeigt. Dient zum Beobachten des XML-Inhalts eines Editors während des Editierens. Hierzu kann das `WatchForm` beliebig oft den `XmlContent` vom Editor abrufen und ihn anzeigen. Auch ein automatisches Aktualisieren in regelmäßigen Zeitabständen mit Hilfe eines Timers ist möglich.

TextConfigForm

Wird vom `TestingForm` angezeigt. Dient zum Bearbeiten eines mehrzeiligen Textes, wie er für die Drag&Drop-Operation benötigt wird.

A3. Datenablage

A3.1 TestingSession-Datei

Inhalt einer TestingSession-Datei am Beispiel `mySession1.tsess`:

```
<?xml version="1.0" encoding="utf-8"?>
<testingSession name="mySession1">
  <recentXmls>
    <xmlfile path="C:\xml\section1.xml" />
    <xmlfile path="C:\xml\book42.xml" />
    <xmlfile path="C:\xml\chapter3.xml" />
    <xmlfile path="C:\xml\chapter4.xml" />
  </recentXmls>
  <autoInitEditingSession value="False" />
  <testedEditor assemblyfile="c:\dlls\XmetalEditor.dll" type="XmlEditor" />
  <configurationUrl text="file:///D:/Noxum/Config/Editors/Xmetal/" />
  <insertString text="insert me" />
  <dragDrop>
    <text>
      &lt;bookstructure&gt;
      &lt;bookref&gt;
            &lt;chapterstructureref id="AWtestbbbCS2" /&gt;
      &lt;/bookref&gt;
      &lt;/bookstructure&gt;
    </text>
  </dragDrop>
</testingSession>
```

Erläuterungen:

Element	Beschreibung
`<testingSession name=" " </testingSession>`	TestingSession **name**: Name der Session ohne Dateierweiterung
`<recentXmls> </recentXmls>`	Zuletzt verwendete XML-Dateien
`<xmlfile path=" " />`	XML-Datei: **path**: Pfad zur XML-Datei
`<autoInitEditingSession value=" " />`	Einstellung des Funktion zum Automatischen Initialisieren des Editors: **value**: True oder False
`<testedEditor assemblyfile=" " type=" " />`	Informationen zur Erzeugung eines `TestedEditors`: **assemblyfile**: Pfad zur Assemblydatei **type**: Klassenname des Editors
`<configurationUrl text=" " />`	`ConfigurationUrl` für den Editor **text**: URI-String
`<insertString text=" " />`	Text für die `InsertString`-Funktion **text**: String zum Einfügen
`<dragDrop> <text> </text> </dragDrop>`	Text für die Drag+Drop-Testfunktion

A3.2 UserSettings.config-Datei

Inhalt einer `userSettings.Config`-Datei:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <userAppSettings>
    <recentSessions>
      <session path="c:\sessions\mySession1.tsess" />
      <session path="c:\sessions\mySession2.tsess" />
      <session path="c:\sessions\mySession3.tsess" />
      <session path="c:\sessions\mySession4.tsess" />
    </recentSessions>
    <recentEditors>
      <editor assemblyfile="c:\dlls\XmetalEditor.dll" type="XmlEditor" />
      <editor assemblyfile="c:\dlls\SimpleEditor.dll" type="EditControl" />
      <editor assemblyfile="c:\dlls\ImageMapEditor.dll" type="Editor" />
    </recentEditors>
    <windowSize width="786" height="596" />
  </userAppSettings>
</configuration>
```

Erläuterungen:

Element	Beschreibung
<**recentSessions**> </recentSessions>	Zuletzt verwendete TestingSession-Dateien
<**session** path=" " />	TestingSession-Datei: **path**: Pfad zur Datei
<**recentEditors**> </recentEditors>	Zuletzt verwendete Editoren
<**editor** assemblyfile=" " type=" " />	Informationen zur Erzeugung eines `TestedEditors`: **assemblyfile**: Pfad zur Assemblydatei **type**: Klassenname des Editors
<**windowSize** width=" " height=" " />	Zuletzt eingestellte Fenstergröße der MainForm **width**: Breite **height**: Höhe

A4. Screenshots

Abb. 8 – MainForm direkt nach dem Start

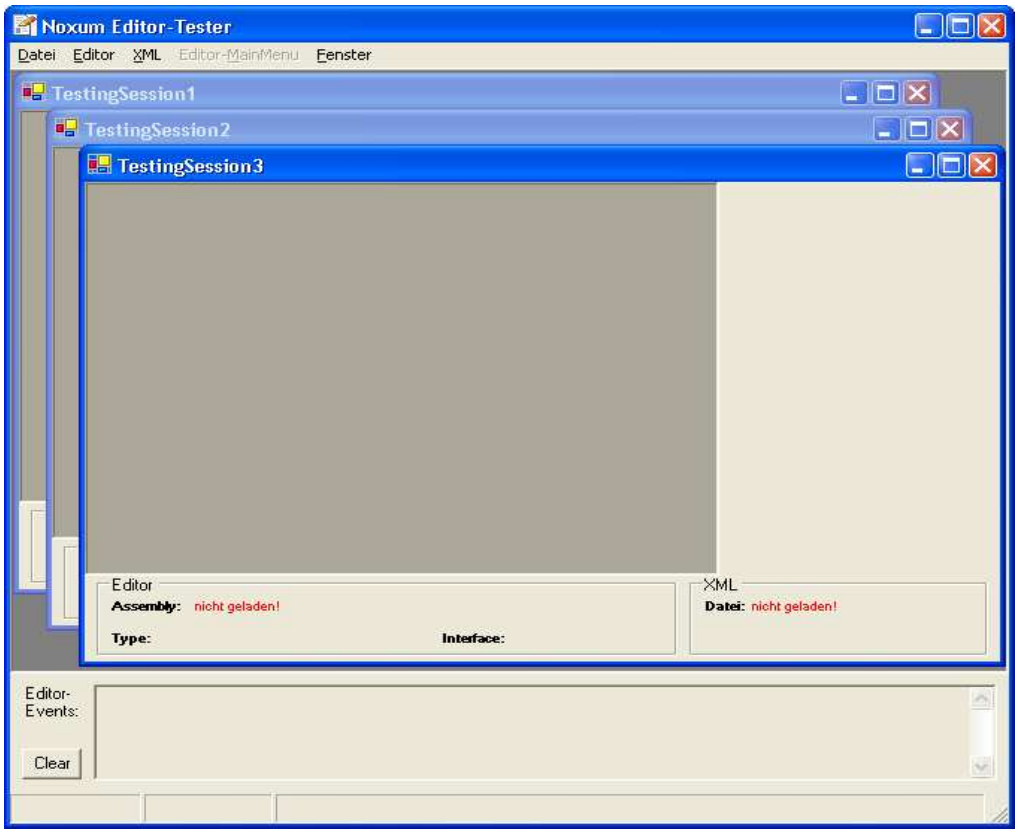


Abb. 9 – MainForm mit TestingForms als MDI-Children

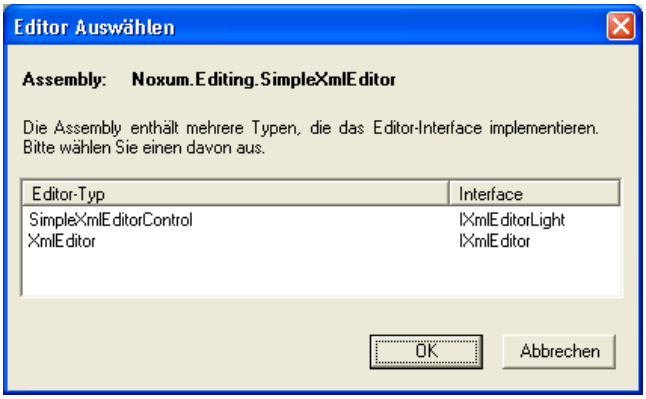


Abb. 10 – EditorSelectionForm

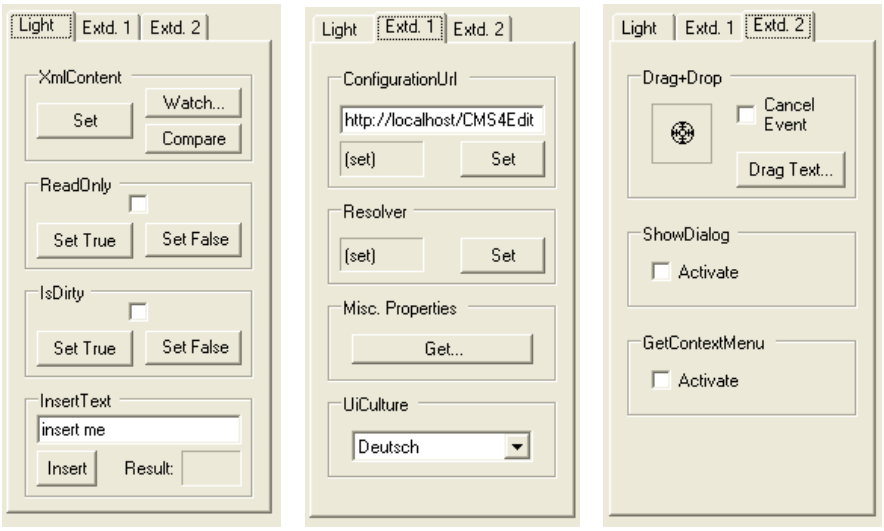


Abb. 11 – Die Editor-Testfunktionen im TestForm. Links die Funktionen von IXmlEditorLight, in der Mitte und rechts die Funktionen des (erweiterten) Interfaces IXmlEditor.

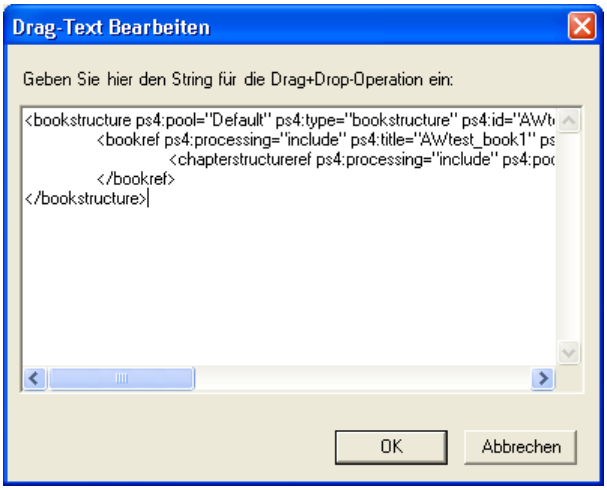


Abb. 12 – TextConfigForm

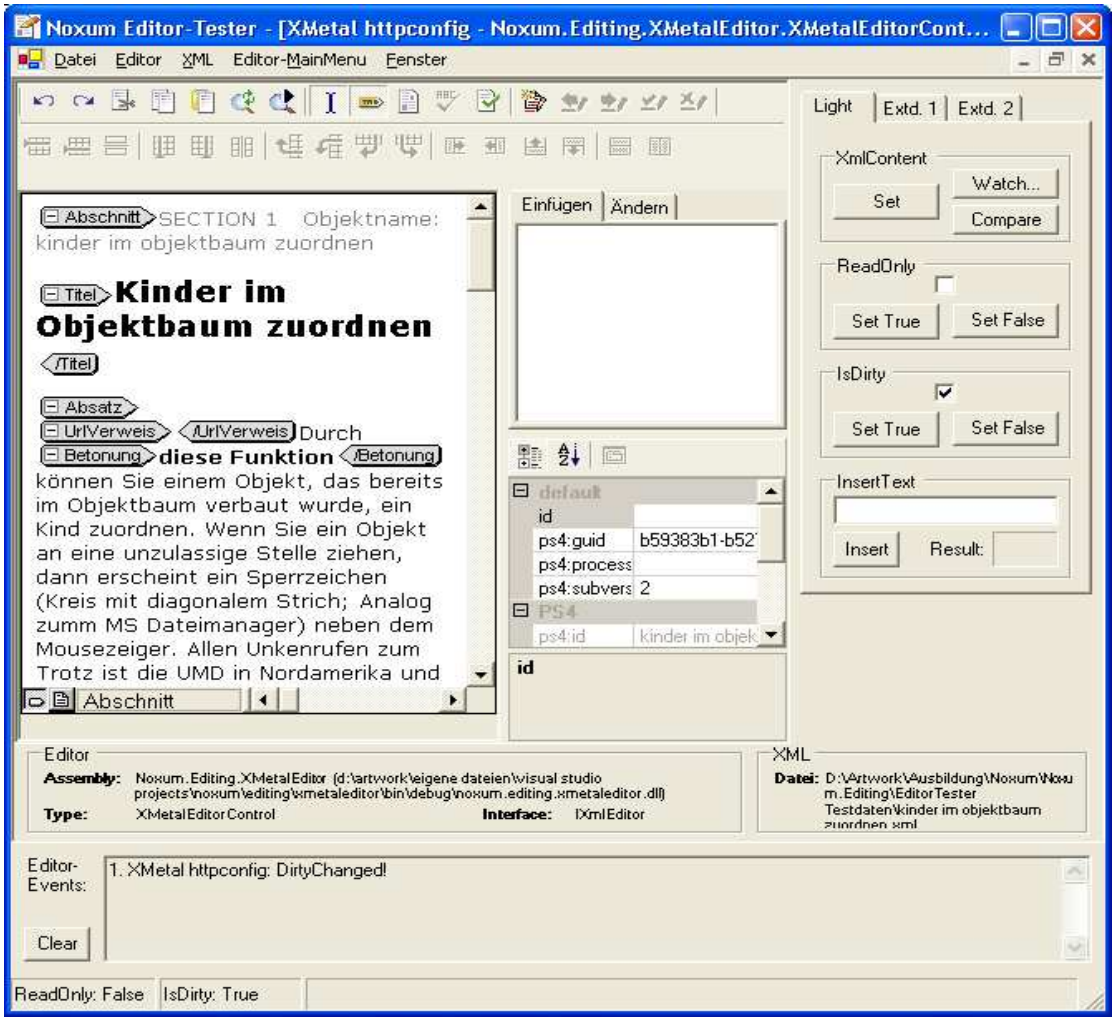


Abb. 13 – MainForm mit komplett eingerichteter Testsitzung und eingebundenem Editor

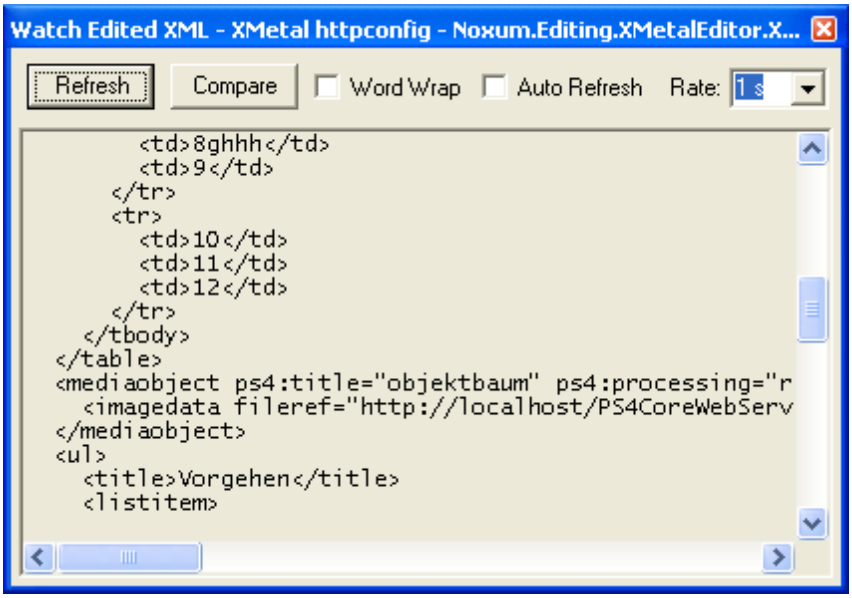


Abb. 14 – WatchForm

A5. Klassendiagramme

TestingForm
-mainForm : MainForm -session : TestingSession -currentEditor : IXmlEditorLight = null -currentXml : XmlDocument -currentXmlFile : string -currentContextNode : XPathNavigator -eventLock : bool = false -statusReadOnly : string = "" -statusIsDirty : string = "" -pnlEditing : Panel -pnlEditingInfo : Panel -gpbEditor : GroupBox -label2 : Label -label3 : Label -label4 : Label -lblEditorAssembly : Label -lblEditorType : Label -lblEditorInterface : Label -gpbXml : GroupBox -label5 : Label -lblXmlFile : Label -pnlEditingControls : Panel -tabPage1 : TabPage -gpbReadOnly : GroupBox -chkReadOnly : CheckBox -gpbXmlContent : GroupBox -butSetXml : Button -butCompare : Button -gpbIsDirty : GroupBox -butIsDirtySetFalse : Button -butIsDirtySetTrue : Button -gpbInsertText : GroupBox -lblInsertTextReturnvalue : Label -lblInsertText1 : Label -butInsertText : Button -txtInsertText : TextBox -tabPage2 : TabPage -gpbReadOnlyProps : GroupBox -butShowReadOnlyProps : Button -gpbResolver : GroupBox -lblResolverSetStatus : Label -butSetResolver : Button -gpbConfigurationUrl : GroupBox -lblConfUrlSetStatus : Label -butSetConfUrl : Button -txtConfUrl : TextBox -gpbCulture : GroupBox -cmbCulture : ComboBox -tabPage3 : TabPage -gpbGetContextMenu : GroupBox -chkGetContextMenu : CheckBox -gpbDragDrop : GroupBox -chkCancelDropEvent : CheckBox -butDragSource : Button -gpbShowDialog : GroupBox -chkShowDialog : CheckBox -mainMenu1 : MainMenu -mnuEditor : MenuItem -openFileDialogAssembly : OpenFileDialog -openFileDialogXml : OpenFileDialog -toolTipMain : ToolTip -saveFileDialogXml : SaveFileDialog -tbcEditingControls : TabControl -chkIsDirty : CheckBox -butReadOnlySetFalse : Button -butReadOnlySetTrue : Button -mnuEditorManagement : MenuItem -mnuXml : MenuItem -mnuCloseEditor : MenuItem -mnuInitEditingSession : MenuItem -mnuAutoInit : MenuItem -mnuOpenXml : MenuItem -mnuSaveXml : MenuItem -mnuSaveXmlAs : MenuItem -mnuCloseXml : MenuItem -mnuOpenEditor : MenuItem -butEditDragText : Button -butWatchXml : Button -components : IContainer

Abb. 16 – Klasse TestingForm (nur Felder)

Um die Klasse hier vollständig abbilden zu können, wurde das Diagramm zweigeteilt.
Die nächste Abbildung zeigt die Methoden und Propertys.

TestEnvironment::**TestedEditor**
-assemblyFile : string -className : string -loaded : bool = false -editorAssembly : Assembly -editorClass : Type -editorInterface : Type
+TestedEditor(in editorAssembly : Assembly, in editorClass : Type, in loadAttempt : bool) +TestedEditor(in editorAssembly : Assembly, in editorClass : Type, in editorInterface : Type) +TestedEditor(in assemblyFile : string, in className : string, in loadAttempt : bool) -LoadEditor() : bool +LoadInterface() : bool +CreateEditorInstance() : IXmlEditorLight +getProperty Assembly() : Assembly +getProperty AssemblyFile() : string +getProperty Class() : Type +getProperty ClassName() : string +getProperty Interface() : Type +getProperty Loaded() : bool +ToString() : string +Equals(in obj : object) : bool +GetHashCode() : int

Abb. 15 – Klasse TestedEditor

TestingForm
+TestingForm(in session : TestingSession) #Dispose(in disposing : bool) -InitializeComponent() +get/setProperty CurrentEditor() : IXmlEditorLight +getProperty Session() : TestingSession +getProperty StatusReadOnly() : string +getProperty StatusIsDirty() : string -RefreshRecentList() -SetCurrentEditor(in editor : IXmlEditorLight) : bool *-SetCurrentEditor(in testedEditor : TestedEditor) : bool* -RefreshEditorPropertyInfo() +RefreshInfo() -OpenEditor() -OpenXml() *-OpenXml(in filepath : string) : bool* -SaveXmlAs() -SaveXml() *-SaveXml(in filePath : string)* -InjectXml() +FetchXml() : XmlDocument -CallInitEditingSession() +CompareXmls(in showDetails : bool) : bool +Xml2Text(in document : XmlDocument) : string -StringOrNull(in myString : string) : string +MessageError(in messages : params string[]) +MessageInfo(in messages : params string[]) *+MessageInfo(in owner : IWin32Window, in messages : params string[])* -ShowEventInfo(in infoText : string) -editor_DirtyChanged(in sender : object, in e : EventArgs) -editorEx_CurrentElementChanged(in sender : object, in e : EventArgs) -editorEx_EditingEvent(in sender : object, in e : EditingEventArgs) -editorEx_BeforeDrop(in sender : object, in e : DragEventArgs) -Editor_ShowDialog(in message : string, in caption : string, in buttons : MessageBoxButtons, in icon : MessageBoxIcon) : DialogResult -Editor_GetContextMenu(in node : XPathNavigator) : ContextMenu -Editor_contextMenu_Click(in sender : object, in e : EventArgs) -TestingForm_Load(in sender : object, in e : EventArgs) -TestingForm_Closing(in sender : object, in e : CancelEventArgs) -mnuOpenEditor_Click(in sender : object, in e : EventArgs) -mnuCloseEditor_Click(in sender : object, in e : EventArgs) -mnuOpenXml_Click(in sender : object, in e : EventArgs) -mnuSaveXml_Click(in sender : object, in e : EventArgs) -mnuSaveXmlAs_Click(in sender : object, in e : EventArgs) -mnuCloseXml_Click(in sender : object, in e : EventArgs) -recentEditorMenuItem_Click(in sender : object, in e : EventArgs) -recentXmlMenuItem_Click(in sender : object, in e : EventArgs) -mnuInitEditingSession_Click(in sender : object, in e : EventArgs) -mnuAutoInit_Click(in sender : object, in e : EventArgs) -butSetXml_Click(in sender : object, in e : EventArgs) -butWatchXml_Click(in sender : object, in e : EventArgs) -butCompare_Click(in sender : object, in e : EventArgs) -chkReadOnly_CheckedChanged(in sender : object, in e : EventArgs) -butReadOnlySetTrue_Click(in sender : object, in e : EventArgs) -butReadOnlySetFalse_Click(in sender : object, in e : EventArgs) -chkIsDirty_CheckedChanged(in sender : object, in e : EventArgs) -butIsDirtySetTrue_Click(in sender : object, in e : EventArgs) -butIsDirtySetFalse_Click(in sender : object, in e : EventArgs) -txtInsertText_TextChanged(in sender : object, in e : EventArgs) -butInsertText_Click(in sender : object, in e : EventArgs) -txtConfUrl_TextChanged(in sender : object, in e : EventArgs) -txtConfUrl_KeyPress(in sender : object, in e : KeyPressEventArgs) -butSetConfUrl_Click(in sender : object, in e : EventArgs) -butSetResolver_Click(in sender : object, in e : EventArgs) -butShowReadOnlyProps_Click(in sender : object, in e : EventArgs) -cmbCulture_SelectedIndexChanged(in sender : object, in e : EventArgs) -butDragSource_MouseDown(in sender : object, in e : MouseEventArgs) -butEditDragText_Click(in sender : object, in e : EventArgs) -chkShowDialog_CheckedChanged(in sender : object, in e : EventArgs) -chkGetContextMenu_CheckedChanged(in sender : object, in e : EventArgs)

Abb. 17 – Klasse TestingForm (nur Methoden und Propertys)

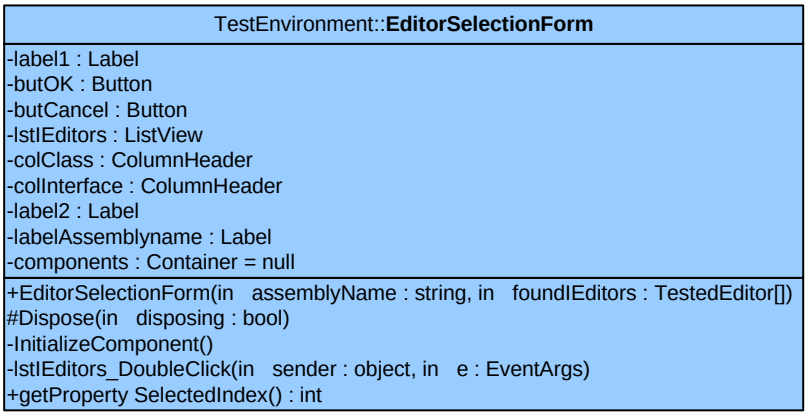


Abb. 18 – Klasse EditorSelectionForm

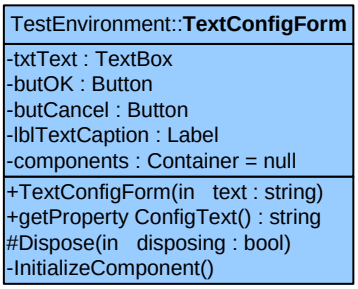


Abb. 19 – Klasse TextConfigForm

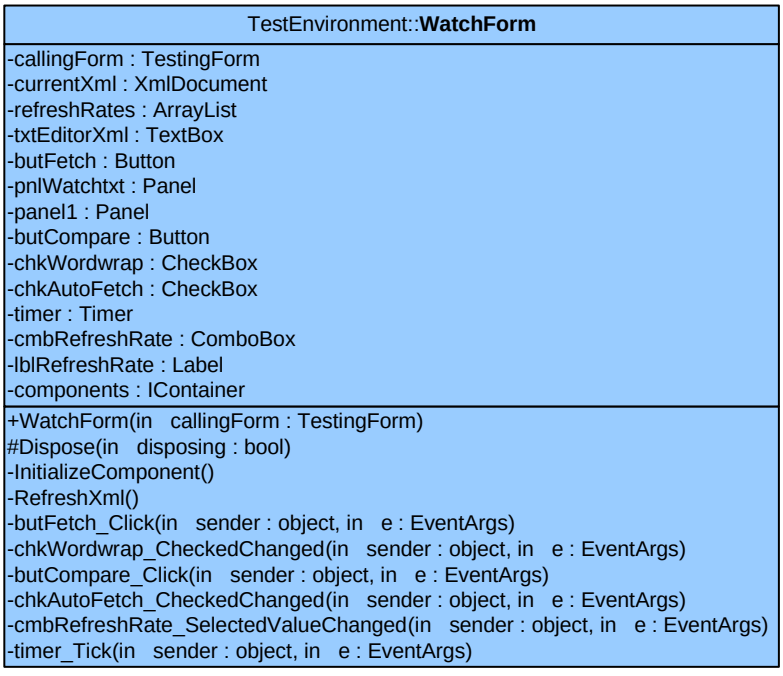


Abb. 20 – Klasse WatchForm

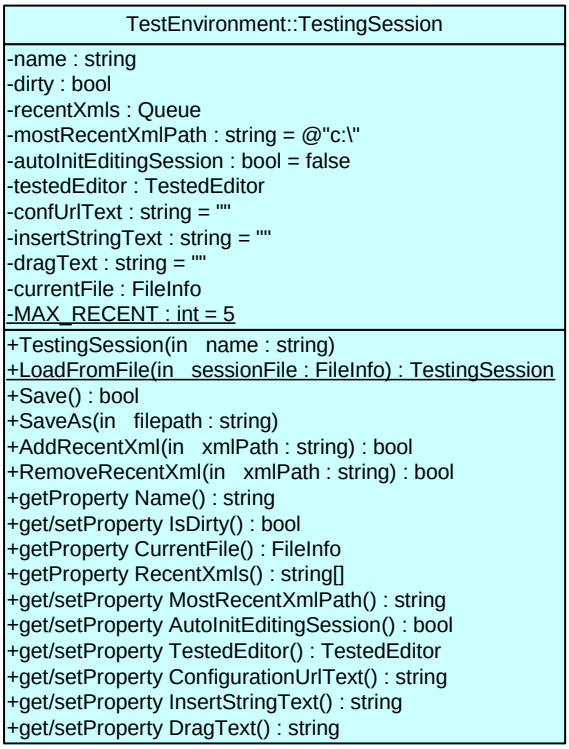


Abb. 21 – Klasse TestingSession

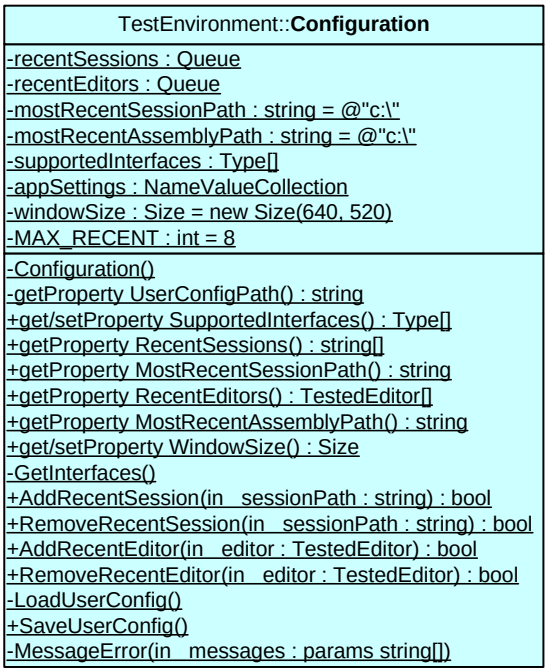


Abb. 22 – Klasse Configuration

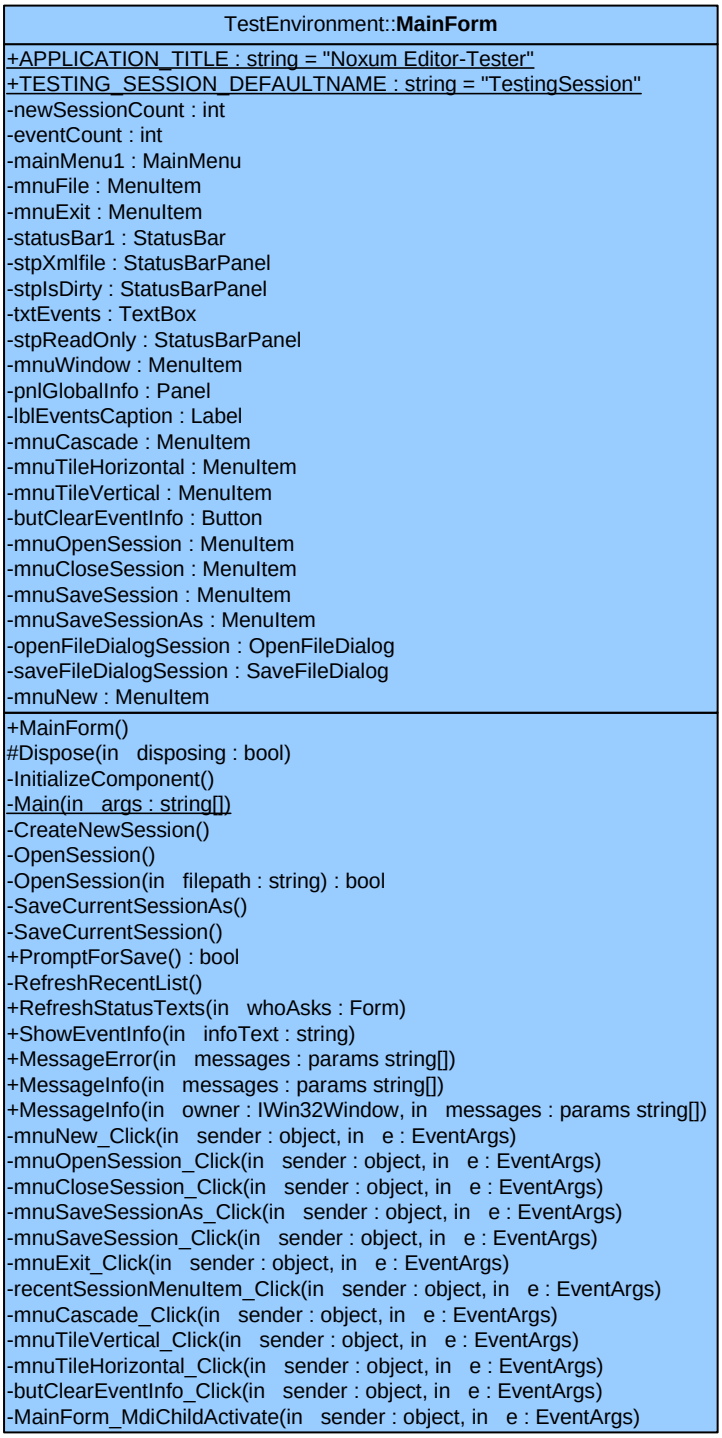


Abb. 23 – Klasse MainForm