UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Bachelorarbeit

Entwicklung einer benutzerorientierten Startumgebung für CrypTool 2.0

Simone Sauer
Matrikelnummer: 2233605

UNIVERSITÄT DUISBURG ESSEN

Abteilung Informatik und angewandte Kognitionswissenschaft
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

15. April 2011

Betreuer und Prüfer:
Dr. Arno Wacker (Lehrstuhl Prof. Weis)

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Bachelorarbeit nimmt sich dem Problem an, die Einsteigerfreundlichkeit des Lernprogramms „CrypTool 2.0“ zu steigern. Zu diesem Zweck wurde eine Startumgebung für dieses Programm implementiert, die die Verwendung für Benutzer vereinfachen soll und so den Einstieg auf diese Weise erleichtert.

1.1. Motivation

CrypTool 2.0 zeichnet sich vor allem durch sein Konzept der visuellen Programmierung aus. Hierdurch können auch sehr komplexe kryptographische Zusammenhänge dargestellt werden. Für Einsteiger erwies sich der Umgang und selbst das Lösen einfacher Aufgaben mit dem Programm jedoch als schwierig [Mey09].

Um diesem Nachteil zu begegnen, wurde die Implementierung eines Startcenters vorgeschlagen. Dieses soll dem Benutzer helfen, Aufgaben zu lösen, die für ihn relevant sind.

1.2. Aufgabenstellung

Da bis zu dieser Arbeit kein Startcenter existierte, sollte die ein solches implementiert werden. Wichtig hierbei war, dass der Benutzer Schritt für Schritt von einer Problemstellung zu einer Lösung geführt wird. Eine Startumgebung sollte nach Programmstart erscheinen und dem Benutzer Aufgaben präsentieren, die zur Auswahl stehen. Entscheidet der Benutzer sich für eine Aufgabe, so kann er mit Hilfe von Erklärungen und Festlegung weiterer Details die Aufgabe weiter spezifizieren. Nach Beendigung der Spezifikation soll sich je nach Festlegung in der Konfiguration entweder ein grafisches CrypTool-2.0-Programm öffnen, das in der Lage ist, diese Aufgabe zu erfüllen oder das Startcenter selbst diese Aufgabe ausführen.

Ein besonderer Aspekt der Anforderungen ist auch, dass die Benutzerabfragen nicht fest kodiert sein dürfen. Sie sollen aus einer Konfigurationsdatei geladen werden und es somit ermöglichen, den Umfang stetig zu erweitern. Es versteht sich hierbei von selbst, dass die Konfiguration von Entwicklerseite einfach gehalten werden soll. Auch ein hohes Maß an Qualität wird vorausgesetzt.

Da die Internationalisierung von CrypTool 2.0 ebenso ein wichtiges Anliegen darstellt, wurde darüber hinaus angestrebt, dass dieses Startcenter Mehrsprachigkeit unterstützen sollte.

1.3. Aufbau der Arbeit

Zunächst sollen alle zum Verständnis der Problemlösung notwendigen konzeptionellen und technischen Voraussetzungen erläutert werden (Kapitel 2, Grundlagen).

Anschließend folgt in Kapitel 3 (Konzept und Design) eine Erläuterung des hier vorgestellten Lösungsansatzes.

In Kapitel 4 (Implementierung) wird auf die Umsetzung des zuvor diskutierten Lösungsansatzes eingegangen.

Darauf folgend wird in Kapitel 5 (Evaluation) unter anderem Anhand von Benutzerbefragungen untersucht, ob das Ergebnis den Anforderungen gerecht wird.

Abschließend erfolgt in Kapitel 6 eine Zusammenfassung mit einem Ausblick über mögliche Erweiterungen und Nutzungsmöglichkeiten.

2. Grundlagen

Da für die Durchführung dieser Arbeit einige Konzepte und technische Grundlagen aufgegriffen wurden, beschreibt dieses Kapitel alle Voraussetzungen, die für die Umsetzung von Bedeutung waren. Dies betrifft selbstverständlich CrypTool 2.0 selbst, ebenso wie die Technologien, auf denen diese Arbeit basiert. Hierunter fallen die *Windows Presentation Foundation* (kurz WPF), welche bei der Entwicklung der Oberfläche genutzt wurde, die Auszeichnungssprache *Extensible Markup Language* (kurz XML), die für die Konfiguration verwendet wurde und das .NET-Framework, welches verwendet wurde, um mit XML zu arbeiten und CrypTool 2.0 zu implementieren. Abschließend wird auf die Richtlinien der Entwicklung grafischer Benutzeroberflächen eingegangen, insbesondere auf die *Microsoft Inductive User Interface Guidelines*.

2.1. CrypTool 2.0

CrypTool 2.0 (im Folgenden CT2) ist eine auf WPF (welche in Abschnitt 2.4.2 beschrieben wird) basierende Lernplattform für kryptografische und kryptoanalytische Verfahren. CT2 wurde in der Programmiersprache C# entwickelt und bedient sich fast ausschließlich .NET-Technologien. Das grundlegende Konzept von CT2 ist die visuelle Programmierung, mit der die bereits implementierten Verfahren beliebig erweitert und daher auch sehr komplex werden können. CT2 versteht sich als Nachfolger von *CrypTool 1*, welches die Möglichkeit der visuellen Programmierung nicht bietet. Dagegen ist der Benutzer von CT2 in der Lage, per *Drag & Drop* eigene Szenarien zu erschaffen, die er wiederum mit anderen Benutzern teilen kann. Im Folgenden soll ein Einblick in Funktionalität und Aufbau von CT2 gegeben werden.

2.1.1. Die Oberfläche

Die Oberfläche lässt sich zunächst grob in fünf Bereiche unterteilen. Wie in Abbildung 2.1 gezeigt, ist die Arbeitsfläche (engl. *Workspace*) in der Mitte angeordnet, während die unterstützenden Elemente, die für das Arbeiten mit CT2 von Bedeutung sind, um den Workspace herum angesiedelt sind. Dies sind links die zu Verfügung stehenden Komponenten, die per Drag & Drop in den Workspace gezogen und die Beispielvorlagen, die geöffnet werden können. Oben befindet sich die Werkzeugleiste (engl. *Toolbar*). Mit ihr können zusammengeschaltete Komponenten ausgeführt und gestoppt werden. Ebenso können auch allgemeine Funktionen dort aufgerufen werden, wie beispielsweise das Laden und Speichern von Dateien. Auf der rechten Seite befinden sich die Einstellungen zu dem derzeitig selektierten Komponente. Unten werden relevante Statusmeldungen angezeigt, die nach Typen unterschieden werden und nach Bedarf auch ausgeblendet werden können. Zu erwähnen ist hier, dass diese Elemente nicht fest sind und man sie je nach Belieben anders anordnen kann.

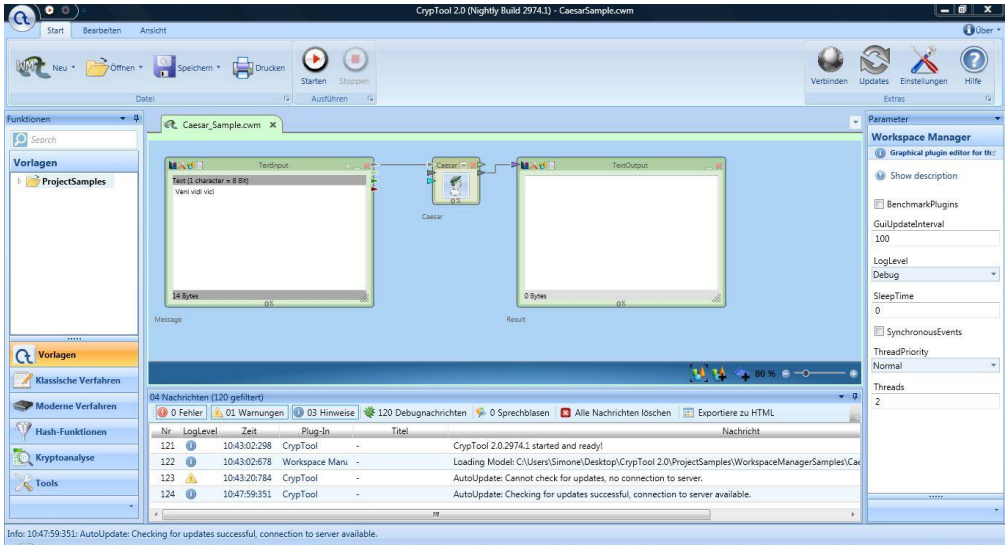


Abbildung 2.1.: Die CT2-Oberfläche mit geöffneter Vorlage

2.1.2. Komponenten

Komponenten bilden die Kernfunktionalität von CT2, da diese die eigentlichen Funktionalitäten von CT2 bereitstellen. Dies können sowohl kryptografische oder kryptoanalytische Verfahren sein, als auch Hilfsfunktionen wie beispielsweise die Konvertierung in unterschiedliche Datentypen. Komponenten können in den Workspace hineingezogen und miteinander verbunden werden, sodass diese eine Kette von Funktionen darstellen, die ausgeführt werden kann, wobei auch Zyklen möglich sind. Komponenten verfügen hierzu über verschiedene Ein- und Ausgänge, von denen einige optional sind, andere hingegen nicht. Diese unterscheiden sich zudem in den jeweiligen Datentypen, die sie akzeptieren.

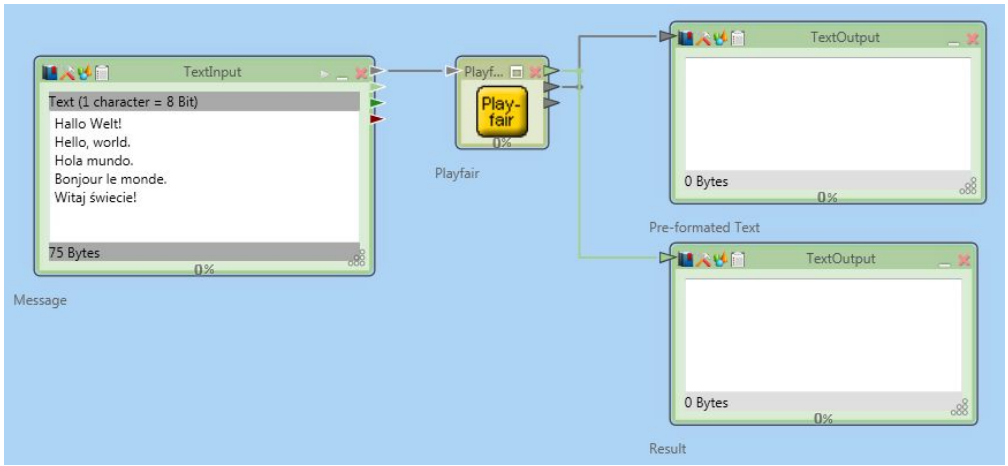


Abbildung 2.2.: Komponenten mit verschiedenen Ein- und Ausgängen

Wie in Abbildung 2.2 erkennbar, haben nicht alle Komponenten sowohl Ein- als auch Ausgänge. Dies hat zur Folge, dass einige Komponenten nur am Anfang einer Kette

stehen können und wiederum andere nur am Ende.

Komponenten können zudem über eine eigene grafische Repräsentation verfügen. Diese kann beispielsweise dazu dienen, Teilergebnisse für den Benutzer sichtbar zu machen oder das Verfahren zu visualisieren. Diese Repräsentation basiert ebenfalls auf WPF.

2.1.3. Datenfluss

Wie zuvor schon angedeutet, können Komponenten miteinander verbunden werden. Auf grafischer Ebene werden diese Verbindungen mit Linien ausgedrückt, die von jeweils einem Ausgang zu einem Eingang führen. Abbildung 2.2 zeigt insgesamt drei Komponenten, die miteinander verbunden sind, (von links nach rechts) einer Text-Eingabe-Komponente, einer Playfair-Komponente (welches einen kryptografischen Algorithmus darstellt) und einer Text-Ausgabe-Komponente.

Die Verbindungen dieser Komponenten repräsentieren den Datenfluss, der zwischen ihnen geschieht. Wird in diesem Fall eine neue Eingabe auf den Ausgang der Text-Eingabe-Komponente gelegt, so kann die Playfair-Komponente diesen verarbeiten und die Ausgabe an den eigenen Ausgang legen. Dieser wiederum hält die Daten für die Text-Ausgabe-Komponente bereit, welches das Ergebnis am Ende anzeigt. Da keine weitere Komponente mehr der Kette angehört, ist die Ausführung damit beendet.

Komponenten verfügen über eine Fortschrittsanzeige. Je weiter der Prozess fortgeschritten ist, desto weiter färbt sich der Rand der entsprechenden Komponente dunkelgrün. Somit ist es für den Benutzer abschätzbar, wann die Ausführungskette beendet sein wird.

2.1.4. Editoren

Die Arbeitsfläche, beziehungsweise der Workspace ist wie zuvor schon erwähnt in der Mitte angeordnet. Dieser bietet die Fläche für die Editoren, mit denen ein kryptographisches oder kryptoanalytisches Szenario bearbeitet werden kann. Zu den Bearbeitungsmöglichkeiten gehören das zuvor erwähnte Drag & Drop, mit dem Komponenten hinzugefügt werden können. Auch die Verbindungen zwischen den Ein- und Ausgängen der Komponenten müssen erstellt werden können.

Neben den Bearbeitungsfunktionen die ein Editor leisten können muss, ist es ebenso nötig, den zuvor beschriebenen Datenfluss zwischen den Komponenten zu unterstützen. Er ist verantwortlich dafür, dass die Ausführung gestartet und gestoppt werden kann. Ebenso dazu gehört auch das Laden und Speichern des erstellten Szenarios.

Da CT2 vor allem auf das Konzept der visuellen Programmierung setzt, ist es essentiell, dass der Editor über eine angemessene grafische Repräsentation verfügt, die es unter anderem auch erlaubt, die Repräsentationen der einzelnen Komponenten anzuzeigen. Auch für die Bearbeitung ist die grafische Repräsentation des Editors von hoher Wichtigkeit.

2.1.5. Der WorkspaceManager

Der *WorkspaceManager* [Kop10] ist ein für CT2 entwickelter Editor. Wie kein anderer Editor zuvor ist er in der Lage, durch sein *WorkspaceModel*, das durch seine Model-View-Controller-Architektur von der grafischen Oberfläche losgelöst ist, Zugriff auf alle Elemente des Modells zu ermöglichen (siehe hierzu Abbildung 2.3).

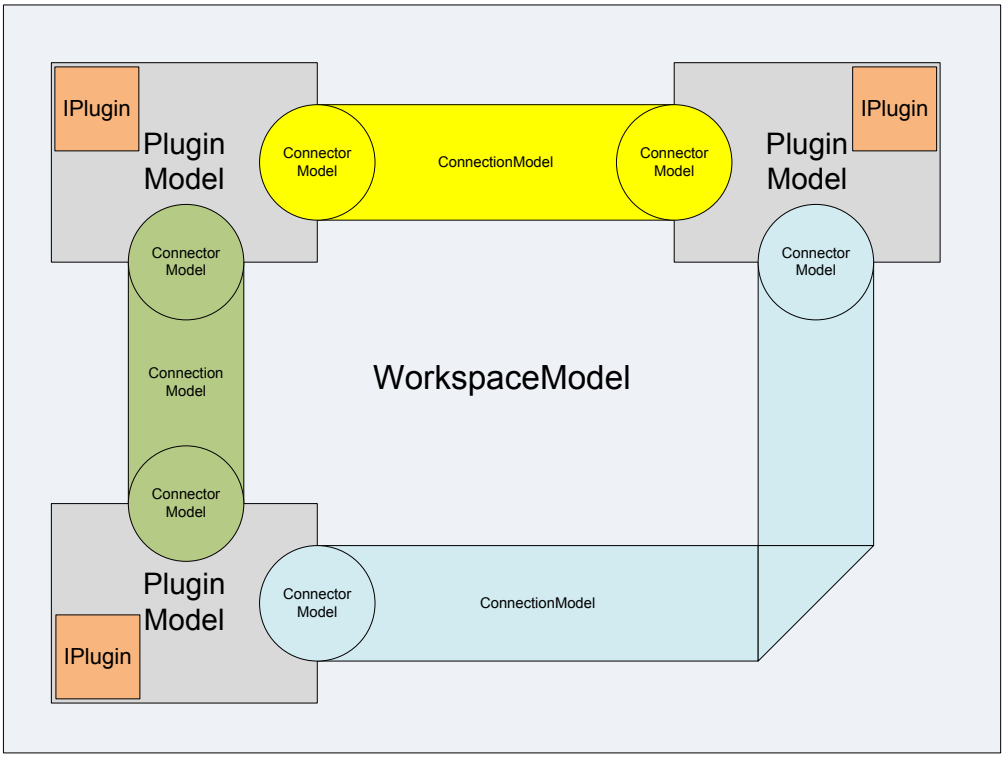


Abbildung 2.3.: Das WorkspaceModel-Konzept [Kop10]

Was hierbei interessant für das Startcenter ist, sind vor allem die Komponenten, die im Modell als Datenstruktur repräsentiert werden. Dadurch eröffnet sich die Möglichkeit, eine Vorlage vor dem Öffnen so weit anzupassen, dass sie für die Erledigung einer vom Benutzer näher spezifizierten Aufgabe geeignet ist. Aus diesem Grund stellt dieses Modell eine wichtige Grundlage dar.

2.2. Windows Presentation Foundation

Die Windows Presentation Foundation (kurz WPF) [Hub10] stellt seit der Einführung von .NET 3.0 Ende 2006 ein modernes Programmiermodell für die Entwicklung von Windowsanwendungen dar. Als Teil des .NET Frameworks ist es Microsofts Plattform für die Entwicklung von Benutzeroberflächen. Da die WPF mit ihren Konzepten für CT2, wie ebenso auch für diese Arbeit verwendet wurde, sollen im Folgenden besondere Eigenschaften der WPF erläutert werden, bevor auf einige der wichtigsten Konzepte der WPF eingegangen wird.

2.2.1. Besondere Eigenschaften

WPF bietet anders als zuvor mit Windows Forms ein flexibles Inhaltsmodell an. Während beispielsweise Buttons zuvor also nur ein Bild oder Text enthalten konnten, kann dies nun ein beliebiges Objekt sein. Dieses Objekt könnte wiederum mehrere Objekte enthalten. Dies bedeutet einen hohen Freiheitsgrad in der visuellen Gestaltung, der zuvor nicht möglich war.

Sogenannte *Custom Controls* (dt. benutzerdefinierte Steuerelemente) [Hub10] trennen ihre visuelle Erscheinung von der eigentlichen Logik und ihrem Verhalten. Im Gegensatz zu sogenannten *User Controls* definiert ein solches Control kein spezielles Aussehen, es muss erst zugewiesen werden. Der Vorteil hierbei ist, dass auf Subklassen verzichtet werden kann, die zuvor nur deshalb erstellt wurden, um das Aussehen auf die eigenen Bedürfnisse anzupassen. Die meisten Controls in der WPF sind als Custom Controls implementiert, was zu einem hohen Wiederverwendungsgrad der Komponenten führt.

Auch einen integrierten Mechanismus für Animationen stellt die WPF bereit. Zweidimensionale oder dreidimensionale Grafiken können in der WPF auf die gleiche Weise gezeichnet und animiert werden. Bisher waren für dreidimensionale Grafiken erweiterte Kenntnisse von beispielsweise *DirectX* erforderlich.

2.2.2. Konzepte

Die *Extensible Application Markup Language* (kurz XAML) [Hub10] ist eine mit .NET 3.0 eingeführte Auszeichnungssprache, die auf XML (in Abschnitt 2.3 erläutert) basiert. In XAML deklarierte XML-Elemente werden zur Laufzeit in .NET-Objekte überführt. XAML kann also zur Beschreibung von Benutzeroberflächen in Windows verwendet werden. Die eigentliche Logik wird dagegen in der CS-Datei (einer in der Programmiersprache C# geschriebenen Datei) abgebildet, auch bezeichnet als *Codebehind-Datei*. Man könnte zwar die gesamte Benutzeroberfläche auch in C# definieren, jedoch bietet XAML einige Vorteile, wie das übersichtliche Trennen der Darstellung von der eigentlichen Geschäftslogik. XAML selbst stellt sich wesentlich kompakter und übersichtlicher dar als äquivalenter Programmcode. Zudem kann XAML separat betrachtet werden, was Designern ermöglicht, unabhängig vom Code zu arbeiten.

Dependency Properties (dt. Abhängigkeitseigenschaften) [Hub10] stellen ein weiteres wichtiges Konzept der WPF dar. Innerhalb der WPF gibt es verschiedene Möglichkeiten, diese Eigenschaften zu setzen. Wie der Name schon verrät, sind sie von verschiedenen Quellen der Anwendung und des Systems abhängig. Dependency Properties haben einen integrierten Benachrichtigungsmechanismus für Änderungen, wodurch die WPF in der Lage ist, diese wahrzunehmen. Diese Eigenschaft macht sie prädestiniert für sogenannte Data Bindings (dt. Datenbindungen). Bei einer Änderung kann so auf diese in einer vordefinierten Weise reagiert werden. Ein Beispiel hierfür wäre eine Bindung von visuellen Elementen an die derzeitige Größe des Elements, in dem sie sich befinden, sodass sie sich an diese anpassen.

Ein ebenfalls durchgängig verwendetes Konzept sind *Routed Events* (dt. weitergeleitete Ereignisse) [Hub10]. Das Prinzip der Entwicklung von Benutzeroberflächen bei WPF besteht darin, eine Hierarchie von Elementen zu erzeugen. Die Elemente innerhalb dieser

Hierarchie stehen in einer Eltern-Kind-Beziehung. Da Elemente wie zuvor angedeutet beliebige Objekte enthalten können, ist es nicht mehr ausreichend, dass nur das im Vordergrund stehende Element ein Ereignis, wie beispielsweise eines Mausklicks, empfängt. In der WPF wäre dies die *Direct*-Strategie. Neu hinzugekommen sind die Strategien *Tunneling* (dt. Tunneln) und *Bubbling* (dt. Blubbern). Bei Tunneling wird ein Event nach unten durchgereicht (von den Eltern zu den Kindern), beim Bubbling nach oben (von den Kindern zu den Eltern). Diese Strategien treten oft in Paaren auf, sodass erst Tunneling und anschließend Bubbling erfolgt. So ist es möglich, beim Tunneling eine Art Vorschau auf das Event zu erhalten, während beim Bubbling eine Reaktion auf das Event erfolgen kann.

Commands (dt. Kommandos) [Hub10] erlauben im Gegensatz zu Events eine bessere Trennung der Oberfläche von der Anwendungslogik. Anstelle eines Eventhandlers wird in XAML ein Command auf einem Element angegeben. Im entsprechenden Interface sind lediglich zwei Methoden und ein Event definiert. Die Execute-Methode löst das Command aus, die CanExecute-Methode gibt an, ob das Command ausgeführt werden kann und das CanExecuteChanged-Event wird dann ausgelöst, wenn sich die nächste Rückgabe der CanExecute-Methode ändern würde. Eine Besonderheit der WPF liegt darin, dass viele Controls bereits vordefinierte Commands besitzen, auf die zugegriffen werden kann.

2.3. Extensible Markup Language

Die Extensible Markup Language (kurz XML) [Bra00] ist heutzutage eine der wichtigsten und verbreitetsten Auszeichnungssprachen. Sie bietet weitaus mehr Möglichkeiten der Verwendung als nur die Gestaltung von Webseiten. Gerade die flexible Datenstruktur ermöglicht es, eine bereits vorhandene beliebig zu erweitern und macht es für eine Anwendung als Datenaustauschformat interessant. Es ist möglich, Dokumente nach eigener Definition zu validieren, worauf später noch eingegangen wird. Diese Eigenschaften ebenso wie die hierarchische Baumstruktur und die menschliche Lesbarkeit von XML waren die Entscheidungsgrundlage, die dazu geführt haben, XML zur Konfiguration des Startcenters zu nutzen. Im Folgenden sollen die Techniken beschrieben werden, die in dieser Arbeit verwendet wurden, um mit XML zu arbeiten.

2.3.1. Document Object Model

In der Regel bieten Programmiersprachen Klassen beziehungsweise Schnittstellen an, mit deren Hilfe man einen vorwärtsgerichteten Lesezugriff oder einen vorwärtsgerichteten Schreibzugriff auf XML-Dokumente erhält. Was hierbei jedoch fehlt, sind Methoden, die das Bearbeiten der dort enthaltenen Elemente erlaubt, wie beispielsweise das Hinzufügen oder Löschen von Elementen. Das Document Object Model (kurz DOM) [Kü10] ist eine vom W3C-Konsortium spezifizierte Schnittstelle, die freien Zugriff auf einzelne Elemente in einem Dokument erlaubt.

Das gesamte XML-Dokument wird beim Arbeiten auf der Basis von DOM in den Speicher geladen. Beim Einlesen des DOM werden alle enthaltenen Teile als Knoten in einem Baum umgesetzt, die Metadaten über sich enthalten, wie beispielsweise den Namen und

die Werte. Welche Aktionen ausgeführt oder Eigenschaften festgelegt oder abgerufen werden, wird bestimmt durch das Objekt des Knotentyps.

2.3.2. XPath

Im Gegensatz zu einem rein vorwärtsgerichteten Lesezugriff wie bei *SAX* (Simple API for XML, eine Standard-API zum Parsen von XML-Daten), erlaubt *XPath* [Kü10], eine ebenfalls vom W3C-Konsortium entwickelte Abfragesprache, eine Navigation zu den gewünschten Elementen, beispielsweise auch rückwärts. Hierbei können Suchmuster angegeben werden, sogenannte *XPath-Ausdrücke*. Wie zuvor bei DOM muss hierfür das gesamte XML-Dokument in den Speicher geladen werden. Die .NET-Implementierung von XPath stellt auch einige *Move*-Methoden zu Verfügung, die ebenfalls erlauben zu navigieren, ohne einen solchen XPath-Ausdruck formulieren zu müssen, was aber den Nachteil birgt, dass gerade bei komplizierten Anweisungen der Quellcode leicht unübersichtlich wird.

2.3.3. LINQ to XML

Die Technik LINQ to XML [Kü10] wurde mit *Microsofts .NET 3.5* eingeführt. Es vereinigt die Fähigkeiten von DOM mit den Möglichkeiten von XPath, innerhalb der Dokumente zu navigieren und soll dem Entwickler die Arbeit mit XML erleichtern. In der Tat stellte sich diese Technik während dieser Arbeit als komfortabel heraus und wurde daher durchgehend verwendet. Hierbei wird jedes Element innerhalb des Dokuments als eine Instanz der Klasse *XObject* beziehungsweise auf Spezialisierungen dieser Klasse abgebildet.

LINQ (Language Integrated Query) ist im Allgemeinen eine Sprachergänzung von .NET. Der eigentliche Vorteil von LINQ sind die zahlreichen Möglichkeiten, über ein neues Abstraktionsmodell Daten abzufragen, wie in diesem Fall die Daten eines XML-Dokuments im Speicher.

```
1 //Anfrage als From-Klausel
2 Customer[] customers = service.GetCustomers();
3 var cust = from customer in customers
4 where customer.Name.Length < 6
5 select new { customer.Name, customer.City };
6 //Anfrage mit Operatoren
7 var cust = customers
8 .Where(customer => customer.Name.Length < 6)
9 .Select(c => new { c.Name, c.City });
```

Listing 2.1: Beispiel für Anfragen mit Hilfe von LINQ to Object [Kü10]

Dieses Beispiel zeigt zwei verschiedene Möglichkeiten der Abfrage. Die verwendete Syntax ähnelt der anderer Abfragesprachen, wie beispielsweise SQL (Structured Query Language) für Datenbanken.

LINQ to XML wurde in dieser Arbeit verwendet, um Elemente innerhalb des in der Konfiguration definierten Baumes mit Hilfe anhand bestimmter Kriterien zu finden. Weitergehende Details werden in Kapitel 4 behandelt.

2.3.4. Document Type Definition

In einer Document Type Definition (kurz DTD) [PW99] werden die Grundregeln für die Struktur eines Dokuments festgelegt. Es wird unter anderem festgelegt, welche Elemente in welcher Reihenfolge im Dokument enthalten sein dürfen und welche enthalten sein müssen. Das Festlegen einer solchen DTD kann je nach Anwendung von großer Wichtigkeit sein. Da neben der Wohlgeformtheit auch die Validität eines XML-Dokuments von Bedeutung ist, müssen DTDs verwendet werden, um sie auf diese hin zu überprüfen.

Viele Entwicklungsprogramme unterstützen heutzutage den Entwickler dabei, sowohl auf Wohlgeformtheit als auch auf Validität der Dokumente zu achten. Dies kann man sich zu Nutze machen, um zu gewährleisten, dass sich jeder, der am Entwicklungsprozess beteiligt ist, daran hält. Zudem macht die Überprüfung den Einstieg eines neuen Entwicklers leichter und die Dokumente somit weniger fehleranfällig. Dies sind die Gründe, die dazu geführt haben, für die Konfiguration des Startcenters eine solche DTD zu definieren. In Abschnitt 3.4 wird näher auf die grundlegende Struktur der Konfigurationsdatei eingegangen.

2.4. Grafische Benutzeroberflächen

Das Design von Benutzeroberflächen (im Folgenden *User Interfaces*, kurz UI) [SP10] geht heutzutage weit mehr darüber hinaus, als nur Felder bereitzustellen, in denen Text eingegeben werden kann. Jedes UI sollte in einem hohen Maße Benutzbarkeit, Universalität und Zweckmäßigkeit bieten. Das primäre Ziel des UI-Designs ist es, Oberflächen auf die Bedürfnisse des Benutzers zuzuschneiden.

Ein hohes Maß an Benutzbarkeit stellt sicher, dass die Benutzer alle Funktionen verwenden können. Das mag zunächst einmal trivial klingen, ist es jedoch nicht. Ein Beispiel wäre hier, dass sichergestellt werden muss, dass der Benutzer in möglichst kurzer Zeit in der Lage sein muss, mit einem Programm umgehen zu können. Ebenso ist es beispielsweise wichtig, dass er bei der Benutzung des Programms eine geringe Fehlerrate hat [SP10].

Der Fokus bei der Universalität liegt darauf, dass das Programm von möglichst vielen unterschiedlichen Benutzern verwendet werden kann [SP10]. Dies hängt nicht zuletzt von der Zielgruppe ab, für die es bestimmt ist. Da es jedoch selbst innerhalb einer Zielgruppe viele Unterschiede geben kann, beispielsweise in der Sprache und Kultur, ist dies ebenso kein zu vernachlässigender Faktor.

Zweckmäßigkeit ist für alle Programme essentiell, selbst für solche, die nicht hauptsächlich über ein UI gesteuert werden. Das Programm muss einfach die Funktionalitäten bieten, für die es bestimmt ist. Für Programme mit UI gibt es allerdings die Einschränkung, dass der Benutzer bei zu viel Funktionalität eventuell den Überblick verliert. Hier gilt es,

das UI trotz vielfältiger Funktionen nicht zu überladen, was wiederum der Benutzbarkeit dient [SP10].

In den folgenden Abschnitten soll auf Richtlinien des UI-Designs eingegangen werden wie ebenso auf die bestehenden Möglichkeiten, diese zu evaluieren.

2.4.1. Die acht goldenen Regeln

Die acht goldenen Regeln des Interface Designs [SP10], entstanden aus drei Jahrzehnten der Erfahrung und können bei den meisten interaktiven Systemen verwendet werden. Nichtsdestotrotz müssen diese immer wieder überprüft und auf die jeweiligen Anwendungsgebiete angepasst werden:

1. Nach Konsistenz streben
2. Für universale Einsetzbarkeit sorgen
3. Informatives Feedback zu Verfügung stellen
4. In sich geschlossene Dialoge entwerfen
5. Fehler vermeiden
6. Leichte Umkehr von Aktionen erlauben
7. Das innere Kontrollbedürfnis unterstützen
8. Die Belastung für das Kurzzeitgedächtnis reduzieren

Im Folgenden soll zu jedem Punkt der acht goldenen Regeln eine kurze Erläuterung gegeben werden.

Konsistente Anweisungssequenzen sollten für alle einander ähnlichen Aktionen eingesetzt werden. Dies betrifft beispielsweise Farben, Schriften, Schriftstile und Terminologie [SP10]. Dies hilft dem Benutzer sich zurechtzufinden, während abweichende Stile zu Verwirrung führen können, die ihn von der eigentlichen Aufgabe ablenken.

Universale Einsetzbarkeit (siehe oben) ist ebenfalls ein Bestandteil der goldenen Regeln. Um diese zu gewährleisten, müssen die individuellen Bedürfnisse der Benutzer und damit der Zielgruppe erfasst werden.

Der Benutzer soll zudem zu jedem Schritt informatives Feedback erhalten. Dieses kann bei kleineren Aktionen kleiner ausfallen, bei wichtigeren sollte es jedoch umso umfangreicher sein. Erwähnenswert ist hier die Möglichkeit, eine direkte Manipulation zu unterstützen [SP10].

In sich geschlossene Dialoge sind für den Benutzer für die Orientierung von großer Bedeutung. Aktionen sollten in Gruppen eingeteilt sein, die einen Anfang, eine Mitte und ein Ende besitzen. Ein gutes Beispiel hierfür sind Internetshops, die den Benutzer durch jeden Schritt der Bestellung führen [SP10].

Fehler zu vermeiden mag einfach klingen, jedoch bedeutet dies nicht nur, das Programm auf eine stabile Weise zu entwickeln. Es gilt auch zu vermeiden, dass der Benutzer selbst mit seiner Eingabe überhaupt kritische Fehler verursachen kann [SP10]. Letztendlich

erspart es dem Benutzer die Frustration, einen Vorgang eventuell erneut durchlaufen zu müssen, um zu einem Ergebnis zu gelangen.

Die Möglichkeit jederzeit von einer Aktion umzukehren befreit den Benutzer davon, befürchten zu müssen, dass seine Fehler vom Programm „nicht verziehen“ werden. Dies ermutigt ihn dazu, mit dem Programm zu experimentieren und neue Möglichkeiten kennenzulernen [SP10].

Erfahrenere Benutzer möchten das Gefühl haben, das UI vollkommen zu beherrschen. Hierzu ist es wichtig, dass das UI auf jede seiner Aktionen reagiert, und zwar immer auf dieselbe Weise mit dem gewünschten Ergebnis. Dies bedeutet das innere Kontrollbedürfnis zu unterstützen [SP10].

Eine Faustregel besagt, dass Menschen in der Regel sieben plus oder minus zwei Informationseinheiten im Kurzzeitgedächtnis verarbeiten können. Was beim UI-Design also vermieden werden sollte, ist dieses zu strapazieren [SP10]. Der Benutzer soll sich von einer Eingabeseite zur nächsten also nichts merken müssen, denn kann er es nicht, muss er zwangsläufig zurückgehen oder im schlimmsten Fall von vorn anfangen, was wiederum zu Frustration führt.

2.4.2. Microsoft Inductive User Interface Guidelines

Das Konzept des *Inductive User Interface* (kurz IUI) [Mic01] wurde entwickelt, um einem allgemeinen Problem des UI-Designs zu begegnen: Software, deren Verwendung sich als schwierig für den Benutzer erweist. Diese Software wird als *deduktiv* (engl. deductive) bezeichnet, da der Benutzer lange Zeit dafür aufwenden muss, von dem UI auf dessen Verwendung und Zweck zu schließen. Eine mögliche Lösung dieses Problems ist die Verwendung eines IUIs, das dagegen selbsterklärend sein soll. Die Gestaltung des UI soll hierbei folgenden Prämissen unterliegen:

1. Den Fokus eines Bildschirms auf eine Aufgabe richten
2. Die Aufgabe benennen
3. Den Bildschirm auf die Aufgabe anpassen
4. Verweise auf weiterführende Aufgaben anbieten

Der erste Schritt klingt simpel, allerdings verlangt er vom Entwickler des UI, die Gesamtaufgabe in ihre Teile aufzuteilen. Wann eine ausreichende Aufteilung gefunden wurde, ist schwer messbar, aber es gibt folgende Faustregel: Ein Bildschirm hat den Fokus auf einer einzelnen Aufgabe, wenn der Entwickler den Zweck mit einer präzisen, aussagekräftigen und natürlich klingenden Überschrift ausdrücken kann [Mic01].

Die Aufgabe zu benennen bedeutet, diese Aufgabe mit einer Anweisung zu belegen. Dies kann sowohl durch eine direkte Anweisung erfolgen („Bitte wählen Sie eine Aufgabe.“) oder durch eine Fragestellung („Welche Aufgabe möchten Sie wählen?“). Da im Schritt zuvor schon im besten Fall eine präzise Überschrift gewählt wurde, sollte man sich auf diese beziehen [Mic01].

Der dritte Schritt ist selbsterklärend. Der Benutzer soll in der Lage sein, die geforderte Aufgabe, die angegeben wird, zu lösen [Mic01]. Welche Anordnung der Elemente sinnvoll erscheint, muss von Fall zu Fall entschieden werden.

Im vierten Schritt sollen Verweise auf weiterführende Aufgaben eingefügt werden. Was diese weiterführenden Aufgaben sind, kann je nach Anwendung unterschiedlich ausfallen. Denkbar wären Möglichkeiten, den Benutzer zurückzunavigieren, falls er den Überblick verloren hat oder Nebenaufgaben, die die Hauptaufgabe unterstützen. Auch Möglichkeiten, die Elemente innerhalb des Fensters neu anzuordnen, sind vorstellbar [Mic01].

Da ein Vorgehen nach dieser Richtlinie in der Lage zu sein scheint, ein UI hervorzubringen, das der Anforderung genügt, selbsterklärend zu sein, wurde entschieden, diese Richtlinie mit in diese Arbeit einfließen zu lassen.

2.4.3. Benutzeroberflächen-Evaluation

Es gibt sehr viele verschiedene Arten von Evaluationsmöglichkeiten [SP10], von Expertenkritik, Versuchsaufbauten in Laboren bis hin zu Befragungen der Benutzer. Auch ist es möglich, ein UI anhand der Einhaltung bestimmter Richtlinien zu validieren. Ziel ist zumeist die Verbesserung des UIs, indem beispielsweise etwas über die Bedürfnisse der Benutzer herausgefunden wird.

In diesem Fall schien es sinnvoll, sowohl eine Befragung mit Hilfe von Fragebögen als auch eine Validierung anhand der Richtlinien vorzunehmen, die bei dieser Arbeit eingeflossen sind. Die Evaluation dieser Arbeit wird in Kapitel 5 behandelt.

3. Konzept und Design

Nach der Abhandlung der Grundlagen, die für die Durchführung der Arbeit Voraussetzung waren, soll nun auf das Konzept wie ebenso auf das Design des neuen Startcenters eingegangen werden. Hierzu wird mit einem Einblick in das Konzept eines sogenannten *Wizards* begonnen, bevor dazu übergegangen wird, auf die Integration in die Oberfläche, die grundlegende Funktionalität und die Konzepte der Konfiguration einzugehen. Abschließend erfolgt die Erläuterung des eigenen Ansatzes der Umsetzung eines IUI.

3.1. Das Wizard-Konzept

Ein *Wizard* (dt. Zauberer) [Mic11] bezeichnet einen Assistenten, der eine Hilfestellung für eine meist mehrschrittige Dateneingabe in einem Programm bietet. Da der Benutzer, wie in Abschnitt 1.2 erläutert, Schritt für Schritt von einer Problemstellung zu einer Lösung geführt werden sollte, bot es sich an, dieses Konzept aufzugreifen.

Wizards nehmen hierbei nach Abfrage von Details einen Teil der Arbeit für den Benutzer ab, der sonst von ihm selbst übernommen werden müsste. Dieses Verhalten kann dazu genutzt werden, um eine Arbeitsfläche in einem gewissen Maß vorzubereiten, bevor der Benutzer in dieser selbst aktiv wird. Hierdurch wird ein Großteil der darunterliegenden Komplexität verborgen [Mic11]. Dies erleichtert vor allem unerfahrenen Benutzern einen Einstieg ins Programm, was das Hauptanliegen dieser Arbeit darstellt. Aufgrund der Umsetzung dieses Konzepts wurde das Startcenter als CT2-Wizard (im Folgenden *Wizard*) benannt.

Wizards sind sowohl für vollkommen lineare Abläufe, als auch für Abläufe, die mehrere Pfade umfassen, geeignet. Wichtig vor allem bei Abläufen, die mehrere Verzweigungen beinhalten, ist sicherzustellen, dass der Benutzer genug Möglichkeiten erhält, sich zu orientieren.

Durch die Komplexität und Vielfalt der möglichen kryptographischen und kryptoanalytischen Szenarien schien es sinnvoll, diese innerhalb des Wizards durch Verzweigungen abzubilden. Ein Pfad in den Verzweigungen behandelt immer ein spezifisches Szenario, wobei die Details zu diesem Szenario auf den jeweiligen *Pages* (dt. Seiten) Schritt für Schritt, durch in sich geschlossene Dialoge, abgefragt werden.

Am Ende eines Pfades erwartet den Benutzer schließlich ein vorbereitetes Szenario, welches sich durch seine Komplexität unterscheidet. Es bietet sich an, ein einfacheres Szenario als Spezialseite des Wizards abzubilden, und komplexere wie zuvor üblich in einem separaten Tab (dt. Reiter) von CT2 zu öffnen, in dem sich der WorkspaceManager befindet. Weitergehende Details werden in den folgenden Abschnitten erläutert.

3.2. Integration in die Oberfläche

Der Wizard kann nach dem Start von CT2 über das Menü oben links geöffnet werden. Nach dem Öffnen befindet er sich in einem Tab in der Mitte, wo der Workspace angesiedelt ist.

Der Benutzer kann sich dafür oder dagegen entscheiden, ihn zu verwenden. Möchte er es nicht, kann er ihn einfach ignorieren und andere Tabs öffnen oder den Wizard-Tab wieder schließen. Er kann über das Menü jederzeit neue und beliebig viele Instanzen des Wizards öffnen, wodurch es ihm ermöglicht wird, in jeder Instanz unterschiedliche Szenarien zu konfigurieren. Dies kann nur dadurch erreicht werden, indem der Wizard als Editor implementiert wird, weshalb entschiedenen wurde, ihn in dieser Weise umzusetzen.

Der Wizard-Tab hebt sich von anderen durch seine einheitliche Farbe und Bezeichnung ab, wie ebenso durch sein Icon, das einen Zauberstab darstellt. Durch seine semantische Nähe stellt das Icon einen hohen Wiedererkennungswert für den Wizard dar. Der dunkelblaue Hintergrund wirkt durch den verwendeten Farbverlauf lebendig, ohne vom eigentlichen Geschehen abzulenken.

Der helle, einfarbige Vordergrund mit deutlicher Umrandung dagegen hebt sich erkennbar vom Hintergrund ab und stellt den eigentlichen Mittelpunkt der Aufmerksamkeit dar. Hier findet der Benutzer auch alle wichtigen Informationen vor, die ihm helfen, sich für einen nächsten Schritt zu entscheiden oder Daten einzugeben.

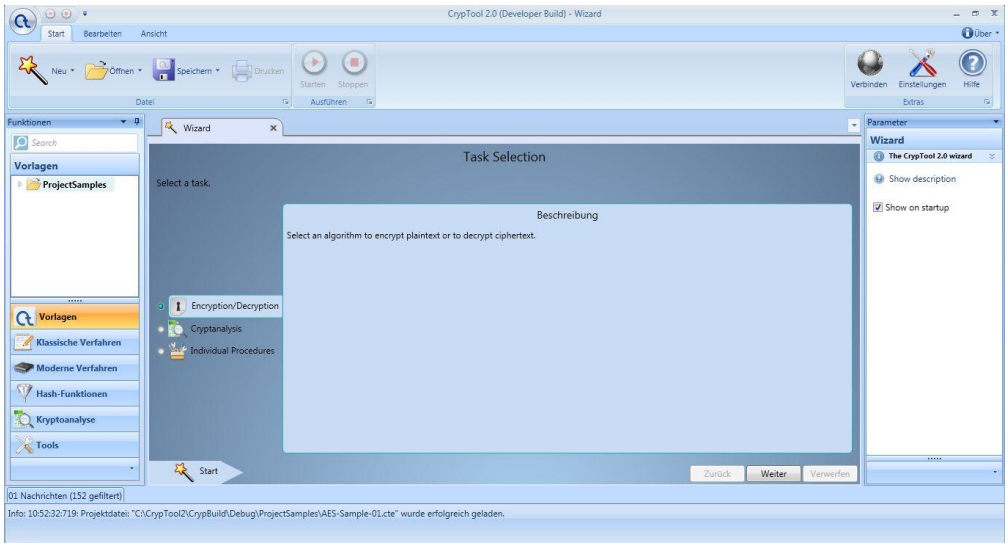


Abbildung 3.1.: Die CT2-Oberfläche mit geöffnetem Wizard

Wie in Abbildung 3.1 erkennbar, befindet sich unten zudem eine Verlaufsanzeige, die optisch durch ihr informatives Feedback unterstützend wirkt. Mit ihrem pfeilartigen Aussehen unterstreicht sie die Semantik des Fortschreitens. Zu jedem Schritt ist ein Icon erkennbar, das symbolisch für den Schritt steht. Rechts neben der Verlaufsanzeige sind die Kontrollschaltflächen platziert, wie man es auch von anderen Wizards gewohnt ist, wodurch ein schnelleres Zurechtfinden gewährleistet werden soll.

Die Gesamtoptik des Wizards versucht also, durch Farben und deren Eingrenzungen die Aufmerksamkeit zu lenken, während Bilder und Formen die Semantik unterstützen sollen. Beides wird jedoch genutzt, um ein gewisses Maß an Wiedererkennungswert zu erreichen. Durch das immer gleiche Auftreten der Seiten soll eine Konsistenz gewahrt bleiben.

3.3. Grundlegende Funktionalität

In den folgenden Abschnitten soll auf die grundlegenden Funktionen eingegangen werden, die der Wizard zu Verfügung stellt. Zunächst soll auf die Frage eingegangen werden, welche Teile des Wizards frei konfiguriert werden können, bevor als nächstes die Verwendungsmöglichkeit mehrerer Sprachen erläutert wird. Anschließend wird beschrieben, wie der Benutzer mit dem UI interagieren kann, wonach als letzter Punkt auf die Vorlagenvorbereitung des Wizards eingegangen wird.

3.3.1. Konfigurierbarkeit

Wie in Abschnitt 1.2 bereits erwähnt, bestand eine der Anforderungen des Wizards darin, dass die Abfrageseiten nicht fest kodiert sein dürfen. Da trotz der Vielfalt der möglichen Szenarien gewährleistet werden sollte, dass die Konfiguration dieser Vielfalt gerecht wird, war es ein Anliegen, die Pages des Wizards so konfigurierbar wie möglich zu gestalten.

Tatsächlich wurde ein hohes Maß an Konfigurierbarkeit erreicht. Bis auf die grundlegende Anordnung der Elemente in Kopfzeile, Aktionsmitte und Fußzeile (vgl. Abbildung 3.4), ebenso wie die Schaltflächen unten rechts ist der Inhalt jedoch frei konfigurierbar. Natürlich gibt es hierbei gewisse Einschränkungen, diese sind jedoch auf die verwendete Technik sowie die CT2-Umgebung zurückzuführen.

Konkret sind der textuelle Inhalt der Überschrift sowie der Aufgabenstellung darunter, die Auswahl- oder Eingabemöglichkeiten mit Beschreibung in der Mitte, und die Beschriftung des Verlaufs konfigurierbar. Dies betrifft ebenso alle verwendeten Icons, die in der Konfigurationsdatei festgelegt werden können. Eingabe- beziehungsweise Ausgabefelder haben zudem die Eigenschaft, dass sie per Konfiguration in Verbindung mit einer Komponente stehen. Ebenso kann konfiguriert werden, wie ein Pfad beendet werden soll. Wie zuvor angedeutet, stehen hierzu zwei Möglichkeiten zur Auswahl. Weitergehende Details hierzu und zur Konfiguration werden in Abschnitt 3.4 behandelt.

3.3.2. Mehrsprachigkeit

Da es auch ein Anliegen darstellte, dass der Wizard Mehrsprachigkeit unterstützt, wurde diese sowohl in den konfigurierbaren als auch statischen Elementen des Interfaces umgesetzt und bedeutet einen Schritt hin zur universellen Einsetzbarkeit. Während sich die statischen Elemente (darunter die Schaltflächen) nach der derzeitigen Sprache und deren Entsprechung in der in der XAML-Datei angegebenen Ressourcendatei richten, werden

die Entsprechungen der konfigurierbaren Elemente in der Konfigurationsdatei selbst gesucht(Genaueres hierzu wird in Kapitel 4 beschrieben). Für die statischen Elemente wurde darauf verzichtet, die Mehrsprachigkeit über die Konfiguration zu unterstützen, da dies nur für Inhalte sinnvoll erschien, die konfigurierbar sind.

3.3.3. Oberflächen-Interaktion

Benutzer haben mehrere Möglichkeiten, mit der Oberfläche zu interagieren. Dazu gehören Eingaben, Auswahl- und Navigationsmöglichkeiten. Die rot umrandeten Elemente in Abbildung 3.2 sind solche, mit denen der Benutzer interagieren kann. Die Hauptaktionen wie Auswahl oder Eingabe sind in der Mitte (1) angeordnet, die Navigation erfolgt über die Leiste unten.

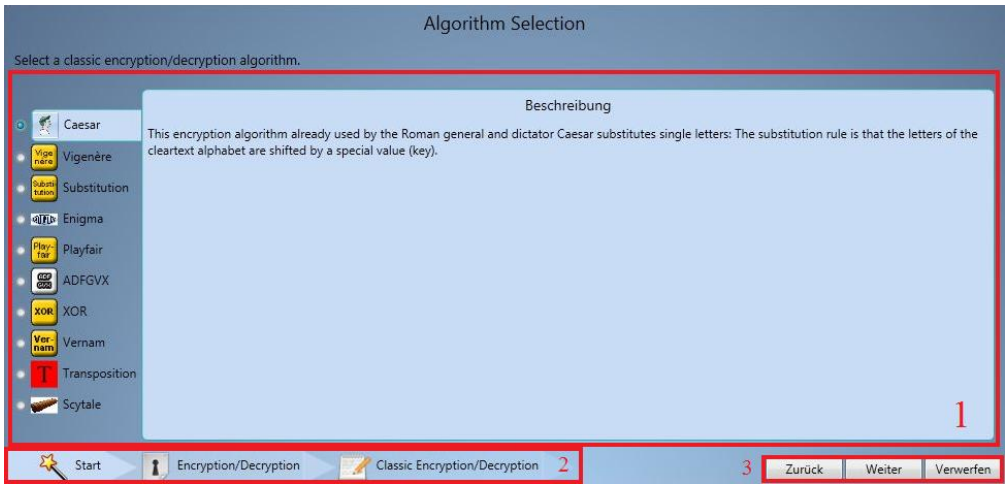


Abbildung 3.2.: Der Wizard

Das mit 2 bezeichnete Element stellt die zuvor schon erwähnte Verlaufsanzeige dar. Mit dieser kann der Benutzer ebenfalls interagieren. Er kann sie verschieben, um die vorherigen Schritte einzusehen, sollte der Anzeigebereich zu klein sein. Damit dient sie zwar hauptsächlich der Orientierung, bietet jedoch auch die Möglichkeit, per Doppelklick zum angeklickten Schritt zurückzukehren. Das Besondere an dem Verhalten ist, dass sich der Wizard alle Eingaben, die der Benutzer bis dahin gemacht hat, „merkt“. Beendet der Benutzer den Wizard nicht und kehrt später zum selben Pfad zurück, findet er alle Eingaben so vor, wie er sie zuletzt belassen hat. Dies ist auch der Fall, wenn er durch Vor- oder Zurücknavigieren (3) die Seite verlässt. So ist eine leichte Umkehr von Aktionen möglich und der Benutzer muss sich von einer Seite zur nächsten nichts merken. Nur wenn der Benutzer den Wizard schließt oder seine Auswahl und Eingaben verwirft, findet er die zuvor in der Konfiguration definierten Standardeingaben vor. Der Benutzer kann auch per Pfeiltasten vor- und zurücknavigieren oder eine Auswahl treffen (in 1). Es gibt auch Situationen, in denen ein Wechsel in den nächsten Schritt nicht erlaubt wird, und zwar dann, wenn die Eingabe in einem Textfeld bezüglich eines zuvor in der Konfiguration festgelegten, regulären Ausdrucks nicht gültig ist, wodurch Fehler durch den Benutzer vermieden werden.

Trifft der Benutzer eine Entscheidung, so reagiert das Interface mit einer Animation. Dies soll das Gefühl einer direkten Beeinflussung hervorrufen. Da das Navigieren hierdurch aber künstlich verlangsamt wird, kann er diese auch per Einstellung ausschalten. Dies könnte für erfahrenere und ungeduldigere Benutzer relevant sein.

3.3.4. Modellmanipulation

Durch die Möglichkeit der Verwendung des zuvor in Kapitel 2 erwähnten Workspace-Models ist der Wizard in der Lage, die Eingaben eines zu einer bestimmten Vorlage gehörenden Pfades an die in dem Modell repräsentierten Komponenten weiterzureichen. Das Modell wird also bereits manipuliert, bevor es zu einer Ausführung kommt, was als Vorbereitung verstanden werden kann. Auch während der Ausführung ist das Modell manipulierbar. Näheres hierzu wird in Kapitel 4 erläutert. Erst durch diese Modellmanipulation entstand die Möglichkeit, einen Wizard, wie in Abschnitt 3.1 beschrieben, umzusetzen.

3.4. Konfiguration

Die Konfiguration erfolgt wie zuvor schon angedeutet mit Hilfe von XML. In diesem Abschnitt soll nun das Konzept der Konfiguration erläutert werden, indem zunächst auf den grundlegende Aufbau eingegangen wird, bevor spezielle Inhalte der Konfiguration behandelt werden. Abschließend werden die Wartbarkeit der Konfiguration, die Robustheit und die Auslieferung behandelt.

3.4.1. Grundlegender Aufbau

Wie in Abschnitt 3.1 bereits beschrieben, leistet ein Wizard meist über mehrere Abfrageseiten verteilt Hilfestellung und sammelt Informationen, die benötigt werden, um eine bestimmte Aufgabe auszuführen. Zu diesen speziellen Seiten wurden jeweils bestimmte XML-Elemente definiert, die eine bestimmte Semantik besitzen.

Es gibt drei verschiedene Arten von Seiten: Auswahlseiten, Eingabeseiten und Ergebnisseiten. Hierbei sind Ergebnisseiten eine spezielle Form von Eingabeseiten, mit dem Unterschied, dass Ergebnisseiten das Ende eines Pfades markieren und auch Resultate anzeigen, die aus einer im Hintergrund ausgeführten Vorlage stammen (mehr hierzu in Kapitel 4). Mit der Identifizierung dieser Seitentypen wurde bereits ein Teil der Komplexität der Aufgaben pro Seite für den Benutzer reduziert, da eine strikte Trennung zwischen den zwei Hauptaktivitäten des Benutzers, Auswählen und Eingeben, während der Verwendung des Wizards durchgesetzt wird.

Eine Auswahlseite wird in der XML-Datei als *Category* (dt. Kategorie) bezeichnet, da man dort in der Regel zwischen verschiedenen Kategorien wählt. Eine Eingabeseite wird dagegen als *Input* (dt. Eingabe) und eine Ergebnisseite als *Sampleviewer* bezeichnet. Der Name „Sampleviewer“ bedeutet so viel wie „Vorlagenbetrachter“ und steht für eine eigene Ansicht des Wizards auf eine Vorlage. Es gibt noch ein spezielles XML-Element, welches ebenso das Ende eines Pfades markiert, jedoch im Gegensatz zum Sampleviewer

keine eigene Seite repräsentiert. Stattdessen zeigt es an, dass ein separater Tab des WorkspaceManagers mit der vorbereiteten Vorlage geöffnet werden soll. Dieses XML-Element heißt *Loadsample*. Diese Bezeichnung ist davon abgeleitet, dass eine Vorlage wie zuvor erläutert mit Hilfe des WorkspaceManagers geladen (engl. load) und angezeigt wird. Bei der Bedeutungsbelegung dieser und anderer XML-Elemente wurde Wert darauf gelegt, dass diese selbsterklärende Namen erhalten, mit denen ein mit CT2 vertrauter Entwickler nach kurzer Zeit umgehen kann.

Das XML-Element, welches eine nächste Seite des Ablaufs definiert, ist hierarchisch immer dem XML-Element untergeordnet, welches die vorherige Seite des Ablaufs definiert. Untergeordnet sind auch alle XML-Elemente, die den Inhalt dieser Seite festlegen (mehr hierzu etwas später in diesem Abschnitt). Ein XML-Element, welches eine Seite definiert (im Folgenden *Seitenelement*) darf jedoch nicht beliebig oft einem anderen Seitenelement untergeordnet sein, was von diesem übergeordneten Seitenelement abhängig ist.

In einem Category-Seitenelement dürfen beispielsweise beliebig viele der anderen erwähnten Seitenelemente vorhanden sein, zwischen denen gewählt werden kann, in einem Input-Seitenelement hingegen darf nur eins der übrigen definiert werden, da es die nächste Seite ohne Wahlmöglichkeit definiert. Da Sampleviewer und Loadsample jeweils das Ende eines Pfades markieren, versteht es sich von selbst, dass innerhalb dieser XML-Elemente keine weiteren definiert werden, die eine neue Seite beschreiben. Diese Anordnung der Seitenelemente wurde gewählt, da sie durch die natürliche Hierarchie des XML-Baumes die möglichen Verlaufspfade am besten wiederzugeben schien.

Das folgende Beispiel (Listing 3.1) soll diese Struktur verdeutlichen und stellt keine vollständige Konfigurationsdatei dar. Eine vollständige Dokumentation zur Konfiguration wird dem CT2-Projektteam übergeben. Abbildung 3.3 stellt zunächst die möglichen Verlaufspfade hierzu dar.

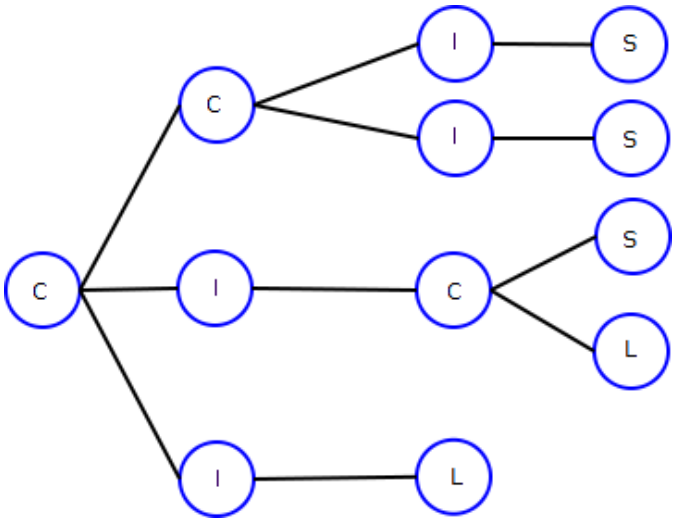


Abbildung 3.3.: Der Verlaufspfad

```
<?xml version="1.0" encoding="utf-8"?>
<category>
  <category>
    <input>
      <sampleViewer/>
    </input>
    <input>
      <sampleViewer/>
    </input>
  </category>
  <input>
    <input>
      <category>
        <sampleViewer/>
        <loadSample/>
      </category>
    </input>
  </input>
  <input>
    <loadSample/>
  </input>
</category>
```

Listing 3.1: Ein Beispiel für eine grobe Seitenelement-Struktur

Wie oben erkennbar, sind in Category-Seitenelementen mehrere weiterfolgende Seiten definiert, während Input-Seitenelemente nur ein weiteres Seitenelement enthalten. Sampleviewer-Seitenelemente und das besondere XML-Element Loadsample enthalten keine weiteren Seitenelemente. Die Struktur für die einzelnen Seiten festzulegen ist natürlich nicht ausreichend, es fehlt unter anderem auch der (sichtbare) *Inhalt*. Auf die verschiedenen möglichen Inhalte soll nun im nächsten Abschnitt eingegangen werden.

3.4.2. Inhalte

Die möglichen Inhalte der einzelnen Seiten variieren, je nachdem, um was für eine Art Seite es sich handelt. Was alle Seiten jedoch gemeinsam haben ist, dass wie bereits in Abschnitt 3.3.1 angedeutet auch der Inhalt der Überschrift und der Aufgabenbeschreibung konfiguriert werden kann. Auch der Name und die Beschreibung einer Seite kann festgelegt werden. Die hierzu verwendeten XML-Elemente sind (in der Reihenfolge) *Headline*, *Task*, *Name* und *Description*. Einem Seitenelement kann auch per Attribut ein Icon zugewiesen werden. Der Name und das Icon stehen in der Verlaufsanzeige repräsentativ für diese Seite beziehungsweise für diesen Schritt.

Der Inhalt einer Auswahlseite wird aus den möglichen Folgeseiten, aus denen der Benutzer wählen kann, abgeleitet. Konkret bedeutet dies für die Konfiguration, dass die Seitenelemente, die dem Seitenelement untergeordnet sind, welches diese Auswahlseite repräsentiert, bereits die benötigten Informationen bieten, um sie in diese Auswahlseite zu überführen. Der Name und das Icon werden hierbei zum Inhalt eines Optionsfelds

und die zugehörige Beschreibung erscheint bei der Auswahl dieses Optionsfelds rechts daneben.

Eingabe- beziehungsweise Ergebnisseiten enthalten dagegen andere Steuerelemente als Optionsfelder. In die Konfiguration aufgenommene und als sinnvoll betrachtete Steuerelemente für diesen Zweck sind konkret Textfelder, Kombinationsfelder und Auswahlkästchen. Bei Textfeldern wird zudem zwischen Eingabe- und Ausgabetextfeldern unterschieden. Alle Steuerelemente repräsentieren in diesem Kontext auch verschiedene Datentypen, worauf später in dieser Arbeit eingegangen wird. Diese Steuerelemente stellen das „Handwerkszeug“ dar, das ausreicht, um Informationen für die Modellmanipulation zu sammeln. Das Modell müsste sonst vom Benutzer über die grafische Repräsentation dieses Modells soweit angepasst werden, dass es die von ihm angedachte Aufgabe erfüllen kann. Diese grafische Repräsentation erwies sich wie bereits erläutert jedoch besonders für Einsteiger als zu kompliziert. Ein Wizard muss daher in der Lage sein, dieses auf äußerlich einfache und bedienbare Steuerelemente zu reduzieren, um die Komplexität vor dem Benutzer zu verbergen.

In der Konfiguration werden diesen Seitenelementen, die Eingabe- beziehungsweise Ergebnisseiten repräsentieren, bestimmte XML-Elemente untergeordnet. Die Bezeichnungen sind wie zuvor bei den Seitenelementen selbsterklärend und weisen somit auf die Art von Steuerelement hin, die sie repräsentieren. Ein Beispiel hierfür wären die Bezeichnungen *Inputbox* und *Outputbox* für Eingabe- und Ausgabetextfelder. Das Besondere an Eingabetextfeldern ist zudem, dass bei der Konfiguration der Inputbox-Elemente ein regulärer Ausdruck angegeben werden kann. Dieser wird während der Laufzeit verwendet, um zu überprüfen, ob die Eingabe des Benutzers eine, in Hinsicht auf diesen Ausdruck, gültige ist. Dies dient, wie bereits erwähnt, der Vermeidung von Fehlern durch den Benutzer. Details hierzu werden ebenfalls später in dieser Arbeit behandelt.

Eine besondere Möglichkeit ist auch, für Ergebnisseiten komponenteneigene Repräsentationen zu konfigurieren. Da diese jedoch von Komponente zu Komponente variieren, ist es von dieser abhängig, ob sich darin Steuerelemente befinden oder nicht. Es könnte auch sein, dass sie rein zur Visualisierung des Verfahrens dienen und daher keine Interaktionsmöglichkeiten bieten. Aus diesem Grund repräsentieren sie auch keinen speziellen Datentyp. Komponenteneigene Repräsentationen können wichtig sein, um ein Verfahren zu verstehen und machen es anschaulich, weshalb diese Möglichkeit angeboten wird.

Das Besondere an textuellen Inhalten ist, dass sie innerhalb der Konfiguration per Attribut eine Sprache in der Form $< Sprachcode > - < Länder/Regionscode >$ oder $< Sprachcode >$ als neutrale Form zugewiesen bekommen können. Ein Beispiel wäre „de-DE“ für „deutsch-Deutschland“. Hierbei ist der Sprachcode von ISO 639-1 und der Länder- beziehungsweise Regionscode von ISO 3166 abgeleitet. Auf diese Weise kann Text als natürlichsprachlicher Inhalt deklariert werden, sodass eine Auswertung anhand der derzeit eingestellten Sprache erfolgen kann (hierzu mehr in Kapitel 4). Auf diese Weise unterstützt die Konfiguration Mehrsprachigkeit. Diese Form der Angabe wird auch durch .NET unterstützt und bietet für Entwickler daher die Möglichkeit, sich über die Dokumentation zu informieren, welche Sprachen angegeben werden können.

All diesen Inhalten gemeinsam ist, dass dem Entwickler viele Möglichkeiten offen stehen, diese durch die Konfiguration nach seinen Wünschen anzuordnen oder die Größe zu bestimmen. Dies ist vor allem wichtig, um eine gewisse Verhältnismäßigkeit zu wahren.

Diese kann indirekt eine Hilfestellung für den Benutzer sein, wenn beispielsweise semantisch zusammengehörige Inhalte nebeneinander angeordnet werden oder die Größe eines Eingabetextfelds vermuten lässt, wie groß die Eingabe in der Regel ist. Die innerhalb der Konfiguration für diese Angaben verwendeten XML-Attribute sind bewusst an XAML angelehnt, da diese hierdurch Wiedererkennungswert erhalten und es für mit XAML vertrauten Entwicklern erleichtert, damit zu arbeiten. Da CT2 auf diesem Konzept der WPF basiert, sollte dies der Fall sein.

3.4.3. Wartbarkeit

Da die Konfiguration unter Umständen aufgrund der Komplexität und Anzahl möglicher Szenarien sehr viele Pfade und Seiten pro Pfad umfassen kann, stellt es ein Anliegen dar, sie dennoch wartbar zu halten. Wartbarkeit bedeutet in diesem Fall unter anderem, dass die Konfiguration so übersichtlich wie möglich gestaltet wird, damit die Pflege einfach ist. Hierzu wurde die Möglichkeit geschaffen, den Konfigurationsbaum auf verschiedene XML-Dateien aufzuteilen. Dies geschieht über XML-Elemente, die eine „Platzhalter-Funktion“ besitzen und einen Dateipfad angeben. Der Inhalt, also das Wurzelelement inklusive aller Nachfolger dieser XML-Dateien wird dann als Geschwisterelement unter dieses Platzhalter-Element im Baum „gehängt“ (Details hierzu werden in Kapitel 4 behandelt).

Wartbarkeit der Konfiguration bedeutet ebenfalls, wie auch bei Software im Allgemeinen, dass Erweiterungen ohne Probleme möglich sind. Das XML-Format gestattet Verzweigungen beliebiger Anzahl und Tiefe, jedoch sind es auch fehlende Hilfen, die einer Erweiterung im Wege stehen könnten. Zur Konfiguration wurde wie bereits erwähnt eine DTD definiert. Da *Visual Studio 2010*, die für CT2 genutzte Entwicklungsumgebung, diese unterstützt, hilft sie den Entwicklern so, die Konfiguration zu bearbeiten. Dies geschieht durch Autovervollständigung, Syntaxhighlighting und Fehlermarkierungen. Ebenso die menschliche Lesbarkeit des XML-Formats macht die Pflege der Konfiguration für Entwickler einfach.

3.4.4. Robustheit

Nicht zu vernachlässigen ist zudem die Behandlung von falsch konfigurierten Inhalten. Welche Elemente und Attribute wo platziert werden dürfen, wird durch die DTD festgelegt, gesonderte Betrachtung muss jedoch den Werten gelten, die innerhalb eines Elements oder Attributs angegeben werden. Das konzeptionelle Grundverhalten, vor allem bei allen Elementen, ist den Wert zunächst zu überprüfen. Stimmt er nicht mit einem der erwarteten Werte überein, so wird zurückgegriffen auf einen zuvor definierten Standardwert, wodurch die Grundfunktionalität einer Seite gewährleistet bleibt, auch wenn sie durch die leicht fehlerhafte Konfiguration eine andere Erscheinung haben könnte. Ein Beispiel wäre hier, dass angegeben werden soll, welcher Eintrag in einem Kombinationsfeld standardmäßig ausgewählt sein soll. Erwartet wird hier eine Ganzzahl, also wird der Wert daraufhin überprüft. Ist die Erwartung nicht erfüllt, so ist der erste Eintrag automatisch der angewählte.

3.4.5. Auslieferung

Während der Durchführung dieser Arbeit kam die Frage auf, wie die Konfiguration ausgeliefert wird und infolgedessen für wen diese verfügbar sein sollte. Die eine Möglichkeit besteht darin, die Konfiguration außerhalb der Binärdateien auszuliefern, sodass jeder, der CT2 benutzt, seinen „eigenen Wizard“ konfigurieren könnte. Die andere Möglichkeit, die ebenfalls diskutiert wurde, besteht darin die Konfiguration innerhalb, also in den Binärdateien auszuliefern. Diese Variante würde nur Entwicklern erlauben, den Wizard zu konfigurieren.

Für die erste Variante spricht, dass sie allen Benutzern einen höheren Freiheitsgrad bietet, da jeder die Möglichkeit hätte, auch jemand, der nicht aktiv am Entwicklungsprozess teilnimmt, zu konfigurieren. Der Nachteil ist jedoch, dass sollte ein Entwickler aus irgendeinem Grund eine wichtige Änderung in der Konfiguration vornehmen wollen, beispielsweise eine Änderung in der Konfigurationsweise, so wird dies Probleme mit der Durchsetzung hervorrufen. Ein anderer Benutzer wird sich immer wieder gezwungen sehen, die eigenen Änderungen an das neue Format anzupassen.

Auch stellt sich die Frage, welche Intention ein Benutzer haben könnte, einen Wizard nach den eigenen Bedürfnissen zu konfigurieren. Vorstellbar wäre, einen Pfad für eine eigene Komponente zu konfigurieren. Allerdings hat er für diese Komponente bereits den Quellcode heruntergeladen und ist somit Entwickler. Um einen Pfad wirklich komponentenbezogen zu konfigurieren, ist Wissen über den Quellcode dieser Komponente nötig.

Abgesehen von diesen Überlegungen ist die Intention eines Wizards, eine Hilfestellung zu bieten. Jemand, der einen Pfad konfiguriert, hat also höchstwahrscheinlich die Absicht, diesen mit anderen Benutzern zu teilen. Als Entwickler kann die Versionsverwaltung genutzt werden, ein anderer Benutzer kann dies nicht. Diese und die vorherigen Überlegungen führten zu dem Schluss, dass die Konfiguration des Wizards nur für Entwickler sinnvoll ist, was auch in dieser Weise umgesetzt wurde.

3.5. IUI-Umsetzung

Wie zuvor in Abschnitt 2.4.2 beschrieben, wurde angestrebt, die *Microsoft Inductive User Interface Guidelines* in diese Arbeit einfließen zu lassen. Da jedoch, wie im letzten Abschnitt erläutert, die meisten Inhalte frei konfigurierbar sind, musste dieses Konzept auf eine solche Umgebung angepasst werden. Anstelle einer konkreten Ausprägung einer Anwendung dieses Konzepts, war in der Implementierung das hauptsächliche Anliegen, Rahmenbedingungen zu schaffen, die es ermöglichen, dieses Konzept umzusetzen und diese Umsetzung zu fördern.

Der Ansatz, der hierzu gewählt wurde, war den grundlegenden Aufbau der grafischen Oberfläche des Wizards und die Konfiguration dieser so zu gestalten, dass der Entwickler, der die Konfiguration vornimmt, sich stets mit den zuvor erläuterten Prämissen der Richtlinie auseinandersetzen muss. Wie im letzten Abschnitt bereits angedeutet, ist die grafische Oberfläche des Wizards in drei Bereiche unterteilt: Kopfzeile, Aktionsmitte und Fußzeile. Die Abbildung 3.4 stellt diese schematisch dar.

Abbildung 3.4.: Schematischer Aufbau der Wizard-Oberfläche

Die Kopfzeile beinhaltet die Seitenüberschrift oben und die Aufgabenbeschreibung darunter, und behandelt damit die ersten beiden Prämissen. Durch die mittig angeordnete Seitenüberschrift, die zudem eine deutlich größere Schriftgröße als die der Beschreibung besitzt, sollen durch einen Entwickler definierte, überlange Überschriften negativ auffallen. Eine fehlende Aufgabenbeschreibung soll ebenso negativ auffallen, da der Abstand so gewählt wurde, dass sonst eine große Lücke zwischen der Überschrift und der Aktionsmitte entsteht. Unpassende oder überlange Beschreibungen sollten Entwickler schon aus zweckgebundenen Gründen vermeiden.

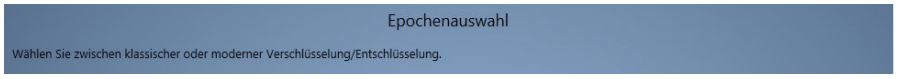


Abbildung 3.5.: Kopfzeile der Wizard-Oberfläche

Bei der Konfiguration der Aktionsmitte muss sich ein Entwickler ebenso zwangsläufig mit der ersten und auch der dritten Prämisse auseinandersetzen. Da die Aufgabe, die auf einer einzelnen Seite erledigt werden kann, im besten Fall bereits mit einer Überschrift belegt wurde, ist die Gestaltung des Inhalts bereits eingeschränkt. Sollte diese Gestaltung vorgezogen und muss die Überschrift noch gewählt werden, so wird dies nicht so leicht gelingen, wenn die Funktionalität dieser Seite bereits „überladen“ ist. Die Aktionsmitte lässt übergroße Elemente zu, jedoch fällt auch dies negativ auf, da diese nicht gänzlich sichtbar sein werden, sodass ein Benutzer den Inhalt nach oben oder unten verschieben muss, um Eingaben vornehmen zu können. Dies kann unter Umständen toleriert werden, jedoch hält diese Eigenschaft, wie auch vor allem das Definieren einer Überschrift mit Aufgabenbeschreibung dazu an, umzudenken und den Inhalt auf mehrere Seiten zu verteilen, zumal dies unbegrenzt möglich ist.

Abbildung 3.6 zeigt eine Eingabeseite des Wizards. Erkennbar sind die kurze, aber aussagekräftige Überschrift und die passende Aufgabenbeschreibung. Darunter befinden sich die passenden Elemente, die zur Eingabe genutzt werden können. Der Inhalt wirkt schlicht, aber zweckmäßig.

Bisher noch nicht behandelt wurde die vierte Prämisse, weiterführende Verweise anzu-

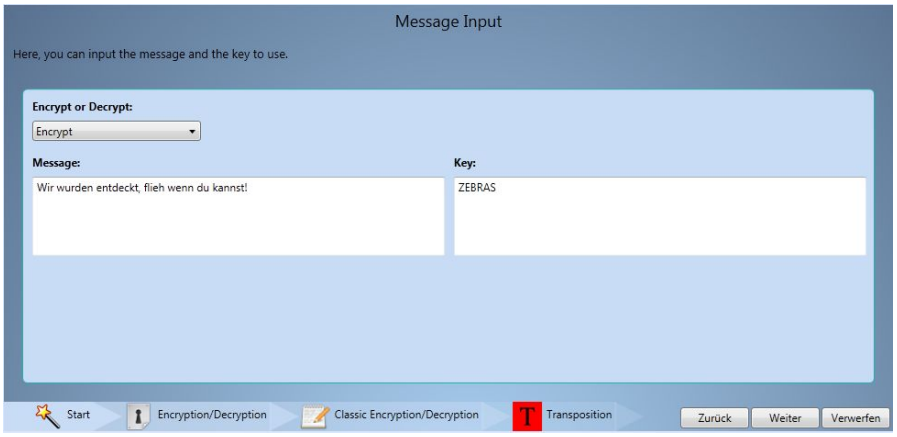


Abbildung 3.6.: Eine Eingabeseite des Wizards

bieten. Diese werden mit der Fußleiste (vgl. Abbildung 3.4) abgedeckt, wo sowohl die Schaltflächen zur Navigation als auch die Verlaufsanzeige angeordnet sind. Bis auf die Beschriftung und die Icons der einzelnen Schritte in der Verlaufsanzeige, ist wie bereits angedeutet, die Fußleiste ausgenommen hiervon nicht konfigurierbar, weshalb sie als eine „echte“ Ausprägung der Richtlinie angesehen werden kann. Als weiterführender Verweis kann, wie zuvor erläutert, auch die Navigation zu zurückliegenden oder zu nächsten Schritten angesehen werden, die über die Fußleiste möglich ist.

4. Implementierung

Nachdem alle für die Arbeit notwendigen Grundlagen und auch die Konzepte und das Design behandelt wurden, beschäftigt sich dieses Kapitel mit der eigentlichen Implementierung des Wizards. Zunächst soll hierzu ein Einblick in die Klassenstruktur und hiermit auch die Einbettung des Wizards in CT2 gegeben werden. Besonders soll hierbei das *WizardControl* betrachtet werden, welches den Hauptanteil der Funktionalität des Wizards beinhaltet. Anschließend wird der interne Ablauf des Wizards von der Instanziierung, über den Seitenaufbau und die Datenübergabe, bis schließlich zum Laden einer Vorlage beschrieben. Zum Ablauf gehört auch die Navigation innerhalb des XML-Baums mit Hilfe von LINQ to XML (wie in Abschnitt 2.3.3 erläutert).

4.1. Klassenstruktur

In diesem Abschnitt soll auf die Klassenstruktur des Wizards eingegangen werden. Wie in Abbildung 4.1 zu erkennen ist, ist die Funktionalität des Wizards in zwei Klassen unterteilt, *Wizard* und *WizardControl* (einige Elemente wurden zu Gunsten der Übersichtlichkeit ausgeblendet):

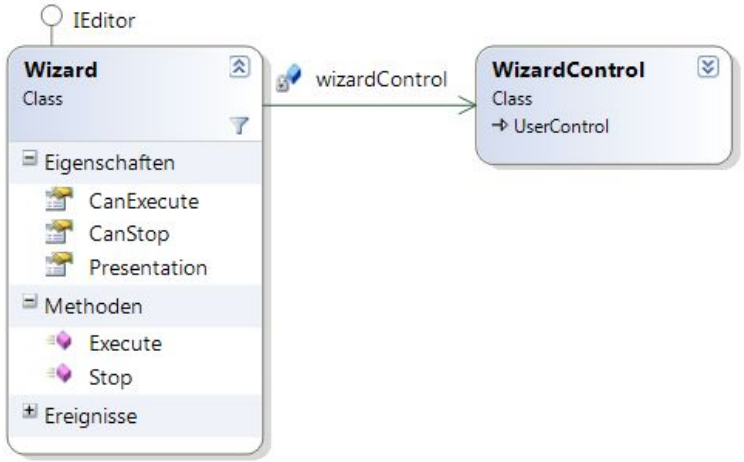


Abbildung 4.1.: Klassendiagramm der Wizard-Implementierung

Das private Feld *wizardControl* erhält bei der Initialisierung eine Referenz auf eine Instanz der Klasse WizardControl. Diese ist auch der Rückgabewert der Eigenschaft *Presentation*. Wie zu erkennen ist, erbt die Klasse WizardControl von der Klasse *UserControl*. In den folgenden Abschnitten soll Näher auf die genaue Einbettung des Wizards in CT2 und auch auf das WizardControl eingegangen werden, welches die Hauptfunktionalität des Wizards bereitstellt.

4.1.1. Einbettung in CT2

Der Wizard wurde als Editor in CT2 eingebettet. Als solcher implementiert er das CT2-Interface *IEditor*. Dieses erweitert das CT2-Interface *IPlugin*. Das IEditor-Interface beinhaltet noch weitaus mehr Methoden und Eigenschaften, die allerdings nicht alle vom Wizard verwendet werden und von denen die wichtigsten bereits in der Abbildung 4.1 zu sehen sind. Der Grund hierfür ist, dass der Wizard zwar als Editor implementiert wurde, aber anstatt die in Abschnitt 2.1.4 erläuterten Funktionen selbst anzubieten, greift er auf den neuesten, für CT2 implementierten Editor, den WorkspaceManager (aus in Abschnitt 2.1.5 erläuterten Gründen) zurück. Für die Implementierung des Wizards interessant sind die Funktionen des WorkspaceManagers zum Laden, Ausführen und Anhalten von Vorlagen und auch die Möglichkeit, das Modell einer Vorlage zu manipulieren, worauf später in diesem Kapitel eingegangen wird.

Die Eigenschaften *CanStop* und *CanExecute* wie ebenso die Methoden *Stop* und *Execute* werden durch die CT2-Oberfläche genutzt, um die dort implementierten Commands umzusetzen. Konkret wird durch CanStop und CanExecute bestimmt, ob der derzeitig offene Editor ausgeführt oder gestoppt werden kann, wobei die Methoden dazu dienen, genau dies zu tun. Der Zustand dieser Eigenschaften bestimmt, ob oben in der CT2-Oberfläche auf Start oder Stopp geklickt werden kann. Bei einem Klick wird dann die entsprechende Methode ausgeführt.

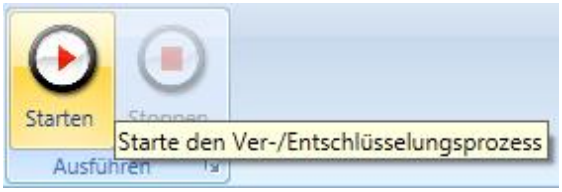


Abbildung 4.2.: Start- und Stoppschaltflächen der CT2-Oberfläche

Da der Wizard diese Funktionalität nicht selbst bereitstellt und auf den WorkspaceManager zurückgreift, wird diese nach dem Laden einer Vorlage auf die Rückgaben der entsprechenden Instanz des WorkspaceManagers zurückgeführt, die diese Vorlage geladen hat.

4.1.2. Das WizardControl

Wie bereits in Abschnitt 2.2 erläutert, ist eine Stärke der WPF die Möglichkeit, die Definition der Oberfläche von der eigentlichen Logik zu trennen, indem die Oberfläche in einer XAML-Datei und die Logik in einer zugehörigen CS-Datei (der sogenannten *Codebehinddatei*) definiert werden. Für das WizardControl wurde dieses Konzept der sogenannten *partiellen Klasse* aufgegriffen. Das WizardControl ist also auf zwei solche Dateien *verteilt* (daher partiell).

Allerdings wurde entschieden, dieses Konzept nicht strikt umzusetzen. Dies bedeutet, dass Teile der Oberfläche nicht von der XAML-Datei definiert, sondern vom CS-Code generiert werden. Da, wie bereits in den Abschnitten 3.3.1 und 3.4 erläutert, ein hohes Maß an Konfigurierbarkeit durchgesetzt wurde, wurde diese Entscheidung getroffen.

Eine Umsetzung durch eine in der XAML-Datei definierte Datenbindung hätte einen deutlich höheren Aufwand bedeutet. Die Informationen, um die jeweils nächste Seite des Wizards zu generieren, werden zur Laufzeit aus dem bereits im Speicher vorhandenen XML-Baum generiert (hierzu später mehr in diesem Kapitel). Es wurden daher nur die grundlegenden Steuerelemente, die das Layout vorgeben, aber selbst keinen Inhalt (das heißt keine „Kindelemente“) besitzen, in der XAML-Datei definiert, da dieser Inhalt erst zur Laufzeit bestimmt wird. Außerdem wurde die Möglichkeit genutzt, in dieser Datei einige *Styles* für bestimmte Steuerelemente und die für Animationen verwendeten *Storyboards* vorzudefinieren, um sie während des Programmablaufs zu nutzen.

Die Codebehinddatei des WizardControls enthält dagegen alle Hauptfunktionalitäten des Wizards, da dort alle Inhalte generiert werden und dort auf die Benutzerinteraktionen mit der Oberfläche reagiert wird.

4.2. Ablauf

Im folgenden Abschnitt soll der interne Ablauf des Wizards von der Instanziierung, über den Seitenaufbau und die Datenübergabe, bis schließlich zum Laden einer Vorlage beschrieben werden. Besonders soll hierbei auch auf die Behandlung der Daten eingegangen werden, die aus den Konfigurationsdateien stammen.

4.2.1. Laden der XML-Konfigurationsdateien

Bei der Initialisierung des WizardControls wird zunächst der komplette XML-Baum auf der Basis von DOM aufgebaut und in den Speicher geladen. Im WizardControl ist der Pfad zur „Eintrittsdatei“ fest kodiert. Diese XML-Datei wird als erstes geladen und stellt den Teil der Baumes dar, der hierarchisch am höchsten liegt. Jeder Knoten eines Baumes wird hierbei als Instanz der Klasse *XElement* repräsentiert. Das XElement, das beim Laden dieser XML-Datei instantiiert wird, stellt also das Wurzelelement des in der XML-Datei dargestellten Baumes dar und stellt Methoden bereit, um innerhalb dieses Baumes nach anderen durch die Klasse XElement repräsentierten XML-Elementen und deren Attributen und Werten zu suchen.

In Abschnitt 3.4 wurde bereits angedeutet, wie die Implementierung des Wizards die Verteilung der Konfiguration auf mehrere XML-Dateien behandelt. Gesucht wird zunächst auf der hierarchisch untergeordneten Ebene des Baumes mit Hilfe der Methoden nach dem XML-Element, welches innerhalb der Konfiguration eine „Platzhalter-Funktion“ (im Folgenden „Platzhalter-Element“) besitzt. Der dort innerhalb dieses Platzhalter-Elements angegebene Dateipfad wird benutzt, um die an dieser Stelle liegende XML-Datei, genau wie bei der Eintrittsdatei, zu laden. Nun wird das XElement, das diese XML-Datei repräsentiert, neben dieses Platzhalter-Element als Geschwisterknoten „gehängt“. Werden in dieser Ebene keine solchen Platzhalter-Elemente mehr gefunden, so wird in die nächst tiefere Ebene gewechselt. Was folgt, ist der gleiche Vorgang wie zuvor. Dieser wiederholt sich, bis keine weiteren Platzhalter-Elemente mehr gefunden werden. Dieser Vorgang wurde mit Hilfe eines rekursiven Methodenaufrufs realisiert. Die zugrunde liegende Annahme ist hier das Vorhandensein einer Baumstruktur, welche

keine Zyklen enthält, da dies keinem gewünschten Szenario entspricht. Das Gesamtergebnis ist eine XElement-Instanz, welche den vollständigen XML-Konfigurationsbaum repräsentiert. Zu erwähnen ist hier, dass jede XML-Datei, die geladen wird, mit der DTD verknüpft wird, die für die Konfiguration erstellt wurde.

4.2.2. Seitenaufbau

Nach dem Laden der XML-Konfigurationsdateien wird der Inhalt für die erste Seite aus dem obersten Element des in den Speicher geladenen XML-Baums generiert. Wie in Abschnitt 3.4 bereits angedeutet, wird der Seitentyp, also Auswahl-, Eingabe- oder Ergebnisseite durch den Namen des XML-Elements bestimmt. Bei einem XElement wäre dies der *XName*.

Zur Generierung des Inhalts der ersten Seite wird das höchste Element des Baumes, also das Wurzelelement, verwendet. Für hierarchisch untergeordnete Elemente unterscheidet sich der Vorgang des Seitenaufbaus nicht. Je nach Name dieses Elements werden andere hierarchisch untergeordnete Elemente vorausgesetzt. Zu erwähnen ist hier, dass die Klasse XElement Methoden und Eigenschaften bereitstellt, mit deren Hilfe zum Vaterknoten und auch zu allen Kindknoten navigiert werden kann. Zunächst soll auf die Vorgänge eingegangen werden, die für alle Seitentypen gleich sind.

Anhand des Vorhandenseins eines Vaterknotens wird zunächst entschieden, ob die Schaltflächen zum Verwerfen der Auswahl und der Eingaben und zum Zurückgehen zum vorherigen Schritt überhaupt aktiviert sind. Die Abwesenheit eines Vaterknotens zeigt an, dass es sich um das Wurzelelement handelt, weshalb diese dann nicht aktiviert sind.

Auch allen Seitentypen gemeinsam sind die Überschrift, die Aufgabenbeschreibung darunter und der Name der Seite, *Headline*, *Task*,*Name* und *Description*. Wie alle den Seiteninhalt bestimmende XML-Elemente sind sie den XML-Elementen, die die Seite repräsentieren (Seitenelemente) untergeordnet. Wie zuvor erläutert, können für alle natürlichsprachlichen, textuellen Inhalte in der Konfiguration Sprachcodes angegeben werden, so auch bei diesen. Die Vorgehensweise, um die passenden XML-Elemente zu der derzeitig eingestellten Sprache herauszufiltern ist hierbei immer gleich. Die derzeitig verwendete Sprache leitet sich von der in .NET in einem Objekt der Klasse *CultureInfo* eingestellten Kultur ab. Der Sprachcode eines XML-Elements wird durch den Wert eines angegebenen Attributs bestimmt. Bei Headline beispielsweise ist die konkrete Vorgehensweise in der Implementierung folgende: Das Seitenelement (also das XElement) wird zusammen mit dem gesuchten Elementnamen (XName), in diesem Fall „headline“ an eine Methode übergeben, die diejenigen hierarchisch untergeordneten XElemente in einer listenartigen Struktur zurückgibt, deren Attributwert mit der derzeitig eingestellten Sprache übereinstimmt. Hierzu werden zunächst alle untergeordneten XElemente herausgefiltert, die den gesuchten Namen besitzen. Enthält die Ergebnismenge Elemente, so wird als nächstes mit Hilfe eines LINQ-Ausdrucks nach denjenigen Elementen gesucht, die im Attributwert den auf den Regions- beziehungsweise Ländercode näher spezifizierten Sprachcode, beispielsweise „de-DE“ enthalten. Sind in der Ergebnismenge Elemente enthalten, so werden diese zurückgegeben. Ist dies nicht der Fall, so wird dagegen nach Elementen gesucht, die die neutrale Form der derzeitig eingestellten Sprache als Attributwert besitzen, beispielsweise „de“. Sollte dies auch keine Elemente als Ergebnis liefern, so wird nach den Elementen gesucht, die als Attributwert die in einer Konstante

angegebenen Standardsprache enthalten, die derzeitig Englisch (also „en“) ist. Sollte es der Fall sein, dass trotzdem keine passenden Elemente gefunden werden, so werden alle Elemente, die nach der Suche nach dem entsprechenden Namen herausgefiltert wurden (in diesem Fall „headline“) zurückgegeben. Sollten jedoch selbst in dieser Liste schon keine Elemente vorhanden gewesen sein, so wird, da es für die Weiterverwendung vorausgesetzt wird, eine neue Liste mit einem Element zurückgegeben, welches im weiteren Verlauf aber keine Bedeutung hat, da dieses XElement weder Werte noch Attribute enthält. Da bei jeder Verwendung von Werten und Attributen eines XElements zuvor geprüft sind, ob diese auch existent sind, stellt dies kein Problem dar. Diese Methode gibt immer eine Liste zurück, da innerhalb der Konfiguration auch Steuerelemente definiert werden können, die mehrere Einträge mit natürlichsprachlichem Inhalt enthalten können, wie beispielsweise Kombinationsfelder. Die Überschrift und die meisten anderen Inhalte dagegen enthalten nur einen Eintrag, weshalb bei der Weiterverwendung der Rückgabe nur das erste Element der Liste betrachtet wird.

Alle wichtigen Informationen zu einer Seite werden mit Hilfe einer separaten Datenstruktur, der *PageInfo*-Klasse festgehalten, die den Namen, die Beschreibung und das Icon, die aus dem entsprechenden XElement generiert wurden, und auch das XElement selbst, das die Seite repräsentiert, enthält. Bei jedem Seitenaufbau wird eine neue Instanz erzeugt, die einer Liste hinzugefügt wird. Bei Veränderung dieser Liste wird ein Event ausgelöst, welches die Generierung der Verlaufsanzeige aus denen in der Liste enthaltenen Informationen auslöst. Mit Hilfe verschiedener Klassen, die in der WPF eine Container- beziehungsweise Anordnungsfunktion für visuelle Objekte bieten, wie *ContentControl* und *StackPanel*, werden die Icons mit den Namen der Seiten zusammengehörig angeordnet, mit der Beschreibung (Description) als sogenanntem *ToolTip* und mit dem zugehörigen XElement als *Tag* versehen (der Grund hierfür wird später in diesem Abschnitt erläutert). StackPanel können horizontal oder vertikal angeordnet sein und mehrere Kindelemente (*Children*) enthalten, die in einer listenartigen Struktur festgehalten werden. Zusätzlich wird ein Event bei jeder Struktur registriert, die den Namen und die Beschreibung einer Seite enthält, um dem Benutzer zu ermöglichen, per Doppelklick zu dieser Seite zurückzukehren. Anhand der Anzahl der in der Liste mit den Seiteninformationen wird die Transparenz der Hintergrundfarbe berechnet.

Der sonstige Seitenaufbau unterscheidet sich nun vom Seitentyp, wobei zunächst die Auswahlseite behandelt werden soll. Das XML-Element, welches in der Konfiguration für eine solche Seite steht heißt „category“. Anhand dieses Namens wird das weitere Vorgehen entschieden. Wie bereits in Abschnitt 3.4 erwähnt, sind alle Inhalte, die eine solche Seite bestimmen in der XML-Datei diesem XML-Element hierarchisch untergeordnet. Dies bedeutet in der Implementierung nun, dass alle XElemente, die weitere Seiten, die zur Auswahl stehen, repräsentieren, herausgefiltert werden müssen. Dies geschieht mit Hilfe eines LINQ-Ausdrucks (siehe Listing 4.1). Hierbei ist „element“ das gerade betrachtete XElement, welches die derzeitige Auswahlseite repräsentiert.

```
1 var options = from el in element.Elements()
2 where el.Name == "category" || el.Name == "input"
3 || el.Name == "loadSample" || el.Name == "sampleViewer"
4 select el;
```

Listing 4.1: Verwendeter LINQ-Ausdruck

Für jedes XElement in diesem Ergebnis werden, wie bereits bei der Generierung des Inhalts der Verlaufsanzeige, das Icon und der Name in einem StackPanel angeordnet. Dieses wird dann einem neu erstellen Optionsfeld zugewiesen. Dieses Optionsfeld erhält wiederum dieses XElement als Tag zugeordnet. In der XAML-Datei wurde per Datenbindung sichergestellt, dass die Höhe des Feldes, in dem sich die Beschreibung (Mitte rechts in Abbildung 3.2) befindet, immer mindestens so hoch ist wie die derzeitige Höhe des StackPanels, in dem alle Optionsfelder einer Seite (Mitte links in Abbildung 3.2) angeordnet sind. Außerdem werden dem Optionsfeld einige Events zugewiesen, die gewährleisten, dass der Benutzer per Doppelklick zur nächsten Seite wechseln und mit den Pfeiltasten das Optionsfeld anwählen kann. Außerdem wird ein Event registriert, bei dessen Auslösung die passende Beschreibung angezeigt wird. Abbildung 3.2 zeigt ein Beispiel für eine Auswahlseite des Wizards.

Noch nicht betrachtet wurden Eingabe- und Ergebnisseiten. Da wie zuvor schon erwähnt Ergebnisseiten eine spezielle Form von Eingabeseiten darstellen, unterscheidet sich der Seitenaufbau einer Ergebnisseite zunächst nicht von dem einer Eingabeseite. Daher soll zunächst auf die Aspekte des Seitenaufbaus eingegangen werden, welche für beide Seitentypen gleich sind, bevor diejenigen behandelt werden, die Ergebnisseiten speziell betreffen.

Wie bereits zuvor beim Seitenaufbau einer Auswahlseite, werden zunächst die hierarchisch untergeordneten XElemente, die den Inhalt einer Eingabe- bzw. Ergebnisseite repräsentieren, mit Hilfe eines LINQ-Ausdrucks herausgefiltert (siehe Listing 4.2). Hierbei ist „element“ das gerade betrachtete XElement, welches die derzeitige Eingabeseite repräsentiert.

```
1 var inputs = from el in element.Elements()
2 where el.Name == "inputBox" || el.Name == "comboBox"
3 || el.Name == "checkBox" || el.Name == "outputBox"
4 || el.Name == "presentation"
5 select el;
```

Listing 4.2: LINQ-Ausdruck für Eingabe- bzw. Ergebnisseiten

Die Rückgabe dieses Ausdrucks wird an eine separate Methode übergeben, welche die in den XElementen angegebenen Werte in entsprechende Steuerelemente der Seite überführt. Die Beschreibung, die in der Konfiguration zu einem Steuerelement angegeben wurde, wird über das zugehörige Steuerelement mit Hilfe eines StackPanels platziert. Jedem Steuerelement wird das XElement, welches es repräsentiert, als Tag übergeben.

Da in der Konfiguration definierte Eingabetextfelder (Inputboxen) die besondere Eigenschaft besitzen, dass in der Konfiguration reguläre Ausdrücke angegeben werden können, die durch die Implementierung bei jeder Eingabeänderung überprüft werden, soll die Überführung in dieses Steuerelement exemplarisch erläutert werden.

Nach der Erstellung eines Steuerelements werden diesem für die Benutzbarkeit als sinnvoll erachtete Eigenschaften zugewiesen, die alle Steuerelemente des gleichen Typs gemeinsam haben. Eine solche Eigenschaft eines Eingabetextfelds ist beispielsweise, dass der Benutzer durch Drücken der Eingabetaste einen Zeilenumbruch erreichen kann, oder dass der Text ohne manuellem Zeilenumbruch automatisch am Rand umbricht.

Als nächstes werden die durch die Konfiguration definierten Eigenschaften abgearbeitet. Bei einem Eingabetextfeld wären dies beispielsweise die sichtbaren Zeilen, wodurch auch indirekt die Höhe mitbestimmt wird. Hierbei wird konkret der entsprechende Attributwert des XElements abgefragt. Da eine Ganzzahl erwartet wird, wird der Wert zunächst daraufhin überprüft, bevor die entsprechende Eigenschaft gesetzt wird.

Wie zuvor erwähnt, ist auch die mögliche Angabe eines regulären Ausdrucks in der Konfiguration interessant. Nach Auslesen des entsprechenden Attributwerts wird unter Verwendung dieses Werts eine Instanz der Klasse *Regex* erzeugt. Danach wird diese zusammen mit dem Eingabetextfeld an eine Methode übergeben, die als Eventbehandlung für eine Eingabeänderung verwendet wird. In dieser wird die Eingabe auf Validität bezüglich des Ausdrucks überprüft. Ist sie nicht valide, so erhält dieses Eingabefeld eine rote Umrandung und das Fortfahren wird verhindert. Da auf einer Seite mehrere solcher Eingabetextfelder vorhanden sein können, werden diese einer Liste hinzugefügt. Ändert sich die Eingabe in einen validen Wert, so wird das entsprechende Eingabetextfeld aus dieser Liste entfernt. Erst wenn die Liste wieder leer ist, wird ein Fortfahren wieder erlaubt.

Auch erwähnenswert ist, dass die Breite eines Steuerelements relativ oder absolut angegeben werden kann. Bei einer relativen Angabe wird eine Datenbindung verwendet, um die Breite dieses Elements an die derzeitige Breite des StackPanels, in dem sich das Steuerelement befindet, anzupassen.

Bisher unerwähnt blieb, dass vor jedem Seitenaufbau alle letzten Eingaben des Benutzers festgehalten werden. Wichtig ist hierbei zu wissen, dass jedes Steuerelement in der Konfiguration einer Komponente (die auch ein *Plugin* ist, weshalb dieses Attribut „plugin“ genannt wurde) und eine zugehörige Eigenschaft („property“) zugewiesen werden kann. Bis auf komponenteneigene Repräsentationen liefern wie bereits erwähnt alle konfigurierbaren Steuerelemente einen bestimmten Wert eines bestimmten Datentyps. Bei einem Kombinationsfeld wäre dies ein *Integer*, der den Index der derzeitigen Auswahl angibt. Bei einem Eingabetextfeld dagegen wäre dies nun der Text als Zeichenkette (*String*). Für die Festhaltung der derzeitigen Werte wird eine weitere separate Datenstruktur, die Klasse *PluginPropertyValue* genutzt. Wie der Name schon verrät, werden die Informationen, welche Komponente und welche Eigenschaft dieses Steuerelement adressiert, zusammen mit dem Wert, den das Steuerelement zum Zeitpunkt des Seitenwechsels hatte, festgehalten. Erstere werden aus dem Tag, welcher das XElement des repräsentierten Steuerelements enthält, extrahiert. Auch in dieser Datenstruktur festgehalten wird der Vaterknoten dieses XElements, dessen Zweck bei der Erläuterung der Verwendung dieser Daten zutage tritt. Diese Daten werden für jedes Steuerelement in einem *Dictionary* festgehalten. Der Schlüssel dieses Dictionarys wird aus der Bezeichnung der Komponente und der Eigenschaft generiert. Alle Instanzen von PluginPropertyValue, die diese Kombination teilen, werden in einer Liste festgehalten. Diese Liste stellt den Wert zu diesem Schlüssel dar. Diese Art der Umsetzung dient der Möglichkeit der Vorsortierung anhand des Schlüssels, damit bei der Suche nach einer Entsprechung nicht alle PluginPropertyValue-Instanzen durchsucht werden müssen.

Jedes Steuerelement kann in der Konfiguration einen Standardwert zugewiesen bekommen, in diesem Fall, bei einem Eingabetextfeld, Text. Da dieser Text auch natürlichsprachlichen Inhalt haben kann, kann diesem wie bereits erläutert ein Sprachcode zugewiesen werden. Die unterschiedlichen Standardwerte aus der Konfiguration werden

also zunächst nach der derzeitig eingestellten Sprache herausgefiltert. Sollte zuvor nicht bereits ein anderer Wert, also Text, für dieses Eingabetextfeld durch den Benutzer festgelegt worden sein, beispielsweise weil er die Seite zuvor bereits besucht hat, wird diese Standardeingabe aus der Konfiguration gesetzt. Ist dies nicht der Fall, nämlich dann, wenn sich ein passender Eintrag in der zuvor beschriebenen Datenstruktur finden lässt, wird dieser dort angegebene Wert verwendet. Da jedoch der Schlüssel, bestehend aus adressierter Komponente und Eigenschaft dieser Komponente, nicht eindeutig ist, da in jedem Pfad die gleiche Kombination vorkommen kann, wird anhand des in der Datenstruktur festgehaltenen Vater-XElements und diesem während des Seitenaufbaus betrachtetem XElement festgestellt, ob sie sich auf dem gleichen Pfad befinden. Dies ist genau dann der Fall, wenn diese entweder identisch sind, das eine zu den Nachfolgern des anderen gehört, oder umgekehrt.

Noch nicht betrachtet wurden die Aspekte, die nur Ergebnisseiten betreffen. Im Grunde genommen unterscheiden sie sich von Eingabeseiten durch zwei Gegebenheiten: Für Ergebnisseiten sind zwei weitere Inhalte konfigurierbar und im Hintergrund wird bereits die zugehörige Vorlage ausgeführt (hierzu später mehr in diesem Kapitel). Durch die Ausführung der Vorlage im Hintergrund ist es möglich, bereits auf Ergebnisse des darunterliegenden Verfahrens, also der Komponentenkette, zuzugreifen. Hierdurch ist es möglich, Ausgabetextfelder und komponenteneigene, grafische Repräsentationen zuzulassen. Deren Konfiguration unterscheidet sich nicht wesentlich von der anderer Inhalte, außer dass komponenteneigene, grafische Repräsentationen keine Eigenschaft angeben, die sie adressieren, da der Zugriff bei jeder Komponente gleich ist (siehe Abbildung 4.1, Eigenschaft *Presentation*). Die Aufgabe der Implementierung ist hier unter anderem sicherzustellen, dass diese Inhalte wirklich nur Ergebnisseiten zu Verfügung stehen. Sollten diese Inhalte versehentlich für eine Eingabeseite konfiguriert worden sein, so werden diese einfach übersprungen, wobei zuvor schon beim Laden jede XML-Datei mit der DTD abgeglichen wird. Anders als bei Eingabetextfeldern werden Werte bei Ausgabetextfeldern nicht vom Textfeld zur Komponente übertragen, sondern umgekehrt (hierzu ebenfalls später mehr in diesem Kapitel).

4.2.3. Seitenwechsel

Nachdem im letzten Abschnitt der Vorgang des Seitenaufbaus erläutert wurde, sollen nun die Vorgänge eines Seitenwechsels behandelt werden. Der Vorgang variiert, je nachdem, wie der Benutzer einen Seitenwechsel auslöst. Im Folgenden wird hierauf eingegangen.

Wie bereits beschrieben werden vielen grafischen Objekten, die XElemente, aus denen sie generiert wurden, als Tag zugewiesen. Bei einer Auswahlseite waren dies die Optionsfelder und bei der Verlaufsanzeige die Container, die innerhalb des horizontal ausgerichteten StackPanels den Seitennamen und das Icon enthalten. Bisher unerwähnt blieb, dass bei einer Eingabeseite, die keine Auswahl zwischen verschiedenen Seiten zulässt, dieses XElement dem in der Mitte liegenden StackPanel, welches die Steuerelemente enthält, als Tag zugewiesen wird.

Bei einem Seitenwechsel tritt nun der Sinn dieses Vorgehens zutage: Löst der Benutzer einen Seitenwechsel aus, beispielsweise, weil er auf „Weiter“ geklickt hat, so kann anhand dieser XElemente bestimmt werden, welche Seite als nächstes folgt, indem diese zur Generierung der nächsten Seite an die entsprechende Methode weitergegeben werden.

Hierzu ist es je nach Auslöser nötig, eventuell zum Vorgänger oder Nachfolger dieses XElements zu navigieren, je nachdem, ob ein Vor- oder Zurückgehen ausgelöst werden soll. Je nach „Bewegungsrichtung“ wird der Liste, die alle PageInfo-Instanzen enthält, ein Element ans Ende hinzugefügt oder das letzte entfernt, wonach die Verlaufsanzeige wie bereits erläutert neu generiert wird. Sollte der Benutzer jedoch über diese Verlaufsanzeige selbst zu einem früheren Schritt wechseln, so gestaltet sich dieser Vorgang etwas anders. Aus dem Index des StackPanels, in dem sich der angeklickte Container befindet, errechnet sich die Anzahl der Elemente, die vom Ende der Liste entfernt werden müssen.

Bevor jedoch der Seitenaufbau folgt, wird, solange der Benutzer diese nicht per Einstellung unterdrückt hat, eine Animation ausgelöst, welche in der XAML-Datei vordefiniert wurde. Diese schiebt den derzeitigen Seiteninhalt aus dem sichtbaren Bereich. Das Ende der Animation löst den Seitenaufbau aus und dessen Ende wiederum die zweite Animation, die die neue Seite in den sichtbaren Bereich schiebt. Das Vor- unterscheidet sich hierbei vom Zurückgehen nur dadurch, dass die jeweiligen Animationen in die jeweils andere Richtung verlaufen.

4.2.4. Laden der Vorlage

Das Laden einer Vorlage geschieht immer dann, wenn das Ende eines Pfades erreicht wurde. Dies ist dann der Fall, wenn das derzeitige XElement, welches an die Methode zum Seitenaufbau übergeben wird, entweder (durch den Namen) eine Ergebnisseite repräsentiert, oder das bloße Laden einer Vorlage mit anschließendem Öffnen eines CT2-Tabs anzeigt.

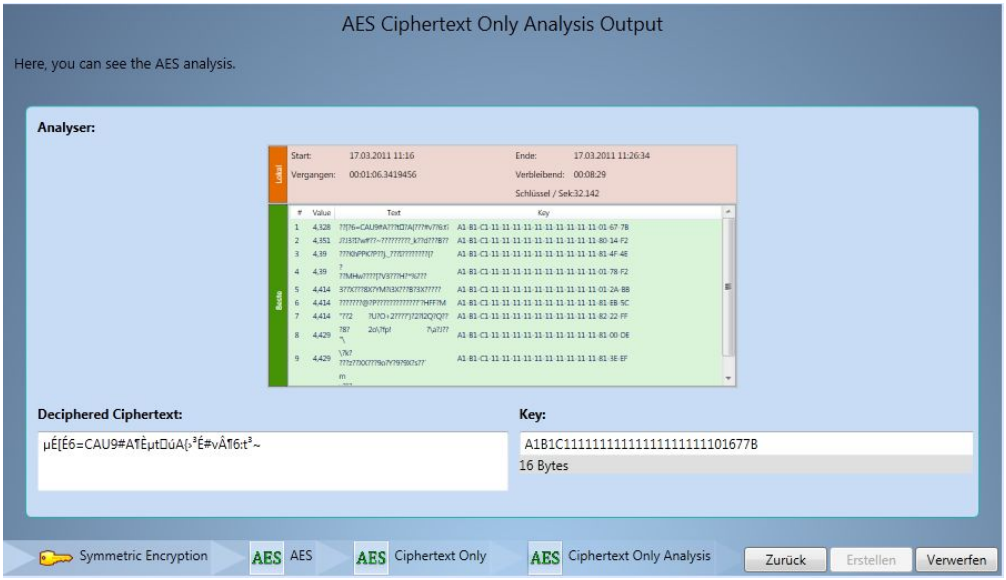


Abbildung 4.3.: Eine Ergebnisseite des Wizards

In beiden Fällen muss zunächst die Datei, die in der Konfiguration angegeben wurde, geladen werden. Hierzu wird eine Instanz eines WorkspaceModels erzeugt, welches dieses durch eine Methode, die diese Datei deserialisiert, realisiert. Dieses WorkspaceModel

muss nun angepasst werden. Wie bereits erwähnt, ermöglicht dieses Modell den Zugriff auf alle dort enthaltenen Komponenten. Um das Modell anzupassen, werden zunächst alle PluginPropertValue-Instanzen, die bis dahin festgehalten wurden, wie zuvor daraufhin überprüft, ob die darin enthaltenen XElemente auf dem selben Pfad wie das XElement, welches die Ergebnisseite repräsentiert oder das Öffnen eines Tabs anzeigt, liegt. Ist dies der Fall, so wird die darin angegebene Komponente im Modell und die zur Komponente zugehörige Eigenschaft, die gesetzt werden soll, gesucht. Ist die Suche erfolgreich, so wird der angegebene Wert gesetzt. Im Gegensatz zu Eingabetextfeldern, bei denen ein Event registriert wird, welches bei Änderung die erneute Setzung dieser Eigenschaft auslöst, wird für Ausgabetextfelder ein Event bei der betreffenden Eigenschaft registriert, welches bei einer Wertänderung das Setzen des Eigenschaftswerts in dieses Textfeld auslöst.

Nach diesen Vorbereitungen wird das WorkspaceModel an eine neue Instanz des WorkspaceManagers übergeben und entweder, bei einer Ergebnisseite, im Hintergrund ausgeführt oder als neuer CT2-Tab geöffnet. Abbildung 4.3 zeigt eine solche Ergebnisseite, welche auch komponenteneigene, grafische Repräsentationen enthält.

5. Evaluation

Dieses Kapitel behandelt die Evaluation dieser Arbeit. Hierzu werden verschiedene Untersuchungsgegenstände betrachtet, die zum einen die Überprüfung dieser Arbeit auf die Umsetzung der in die Arbeit eingeflossenen Richtlinien umfasst, zum anderen auch eine durchgeführte Benutzerbefragung, um den derzeitigen Wert der Arbeit einschätzen und den Wizard in Zukunft verbessern zu können. Auch betrachtet wird, ob das eigentliche Ziel dieser Arbeit erreicht wurde. Hierzu soll zunächst ein Überblick über die verschiedenen Untersuchungsgegenstände gegeben werden, bevor diese der Reihenfolge nach abgearbeitet werden. Dies bedeutet eine aktive Auseinandersetzung mit den verwendeten Richtlinien, die Vorbereitung, Durchführung und Auswertung der Benutzerbefragung und der anschließenden Gesamtbewertung.

5.1. Untersuchungsgegenstände

In diesem Abschnitt soll ein kurzer Überblick über die betrachteten Untersuchungsgegenstände gegeben werden.

5.1.1. Einhaltung der Richtlinien

Einer der folgenden Untersuchungsgegenstände wird sein, zu überprüfen, ob diese Arbeit den Konzepten und Richtlinien gerecht wird, die in diese mit einfließen sollten. Dies stellt einen wichtigen Punkt in der Evaluation dieser Arbeit dar, da die Entscheidung, besagte Richtlinien und Konzepte aufzugreifen, aus dem Grunde gefallen ist, weil sie dieser Arbeit zuträglich erschienen. Würde diese Arbeit nicht den Prämissen der Richtlinien unterliegen, könnte nicht der Anspruch erhoben werden, dass diese Arbeit mit diesen Richtlinien in Einklang steht.

Was hierbei nicht außer Acht gelassen werden darf ist, dass eine Richtlinie oft keine genauen Aussagen trifft, wie sie umzusetzen ist, da sie immer auch verallgemeinert und die Umgebung, in der sie umgesetzt werden kann, sehr variabel ist. Aus diesem Grund kann das Ziel hierbei nicht sein, einen Beweis zu liefern, dass diese Richtlinien bis ins letzte Detail umgesetzt wurden, sondern vielmehr, über den Ansatz der Umsetzung kritisch zu reflektieren (in Abschnitt 5.2).

5.1.2. Benutzerakzeptanz

Die Akzeptanz von Benutzern gegenüber eines neuen *Features* (einem neuen Programm-Merkmal) sollte überprüft werden. Dies kann beispielsweise über Benutzerbefragungen geschehen, welche auch hier durchgeführt wurden. Eine solche Erhebung kann niemals

ein Beweis sein dafür, dass alle Benutzer dieses neue Feature akzeptieren, sie kann jedoch *Hinweise* geben. Wenn ein Großteil aller Befragten eine positive Antwort geben, so kann dies beispielsweise ein Hinweis dafür sein, dass die derzeitige Ausprägung von diesem Teil der Benutzer bereits akzeptiert wird, was aber dennoch nicht zwangsläufig bedeutet, dass nicht selbst diese Benutzer einiges an diesem Feature verbesserungswürdig fänden.

Die Überprüfung der Benutzerakzeptanz dient in erster Linie dazu, den Wert der derzeitigen Ausprägung des Features einzuschätzen und Ansatzpunkte zu suchen, um diesen zu steigern. Wenn eine Vielzahl von Benutzern angibt, eine spezielle Funktionsweise zu vermissen, so kann dies aufgegriffen und umgesetzt werden. Die Kriterien, die Durchführung und die Auswertung der Benutzerbefragung werden in Abschnitt 5.3 behandelt.

Was bei einer solchen Überprüfung jedoch außer Acht gelassen wird, ist die komplexe Funktionsweise, die dahinter steht, da sie (nicht ohne Grund) dem Benutzer verborgen bleibt, weshalb diese gesondert betrachtet werden muss.

5.1.3. Zielerfüllung

Ein weiterer Untersuchungsgegenstand ist ein kritischer Vergleich der Arbeit mit dem angedachten Ziel, also denen in der Aufgabenstellung beschriebenen Eigenschaften, die diese Arbeit erfüllen sollte. Da alle vorherigen Untersuchungen hierbei einfließen müssen, ist dies der letzte Untersuchungsgegenstand und wird daher zuletzt betrachtet. Diese kritische Auseinandersetzung geschieht in der abschließenden Gesamtbewertung (Abschnitt 5.4). Auch betrachtet werden hier die technischen Aspekte der Umsetzung.

5.2. Richtlinien

Nachdem kurz auf die Untersuchungsgegenstände eingegangen wurde, soll nun eine kritische Auseinandersetzung mit der Umsetzung der verwendeten Richtlinien folgen.

5.2.1. Microsoft Inductive User Interface Guidelines

Die Microsoft Inductive User Interface Guidelines geben wie bereits erläutert vier Prämissen vor, denen die Implementierung zur Gestaltung eines IUI unterliegen soll. Da aber viele gestalterische Maßnahmen, die das User Interface betreffen, von der Konfiguration abhängen, wurde ein Kompromiss angestrebt.

Das Ziel der Implementierung war daher, nicht das User Interface vollständig nach dieser Richtlinie auszugestalten, sondern Rahmenbedingungen zu schaffen, die eine Konfiguration nach diesen Richtlinien fördert (siehe Abschnitt 3.5). Hierzu wurden Maßnahmen ergriffen, die im Wesentlichen den grundlegenden Aufbau der grafischen Oberfläche betreffen, also das *Layout*. Dieses wurde in der Weise gestaltet, dass sich ein Vorgehen nach diesen Prämissen als günstig erweist. Ein ungünstiges Vorgehen bedeutet, eine Abfrageseite des Wizards mit Inhalt zu „überladen“, was dem Grundgedanken der Prämissen entgegensteht, eine Seite auf eine Aufgabe zu fokussieren. Auch die strikte Unterscheidung zwischen Auswahl- und Eingabeseiten trägt dazu bei, die Seiten aufgabenorientiert zu gestalten.

Da diese Arbeit keine „echte“ Ausprägung dieser Richtlinie darstellt, kann von keiner strikten Umsetzung dieser Richtlinie gesprochen werden. Dies war aber auch nicht das Ziel. Ziel war es, diese Richtlinie in diese Arbeit einfließen zu lassen, was durch die zuvor erwähnten Maßnahmen geschehen ist. Hierdurch kann abschließend festgestellt werden, dass dieses Ziel erreicht wurde.

5.2.2. Die acht goldenen Regeln des UI-Designs

In Abschnitt 2.4.1 wurden die „acht goldenen Regeln des UI-Designs“ beschrieben, während in Kapitel 3 einige Vorgehensweisen mit diesen begründet wurden. Im Folgenden soll zu jeder Regel zusammenfassend untersucht werden, inwiefern diese in die Arbeit eingeflossen sind.

Die erste Regel, „Nach Konsistenz streben“, zielt im Wesentlichen darauf sicherzustellen, dass der Benutzer sich zurechtfindet und sich nicht mit abweichenden Stilen oder Abläufen des Programms auseinandersetzen muss. Im Rahmen dieser Arbeit wurden Abläufe des Seitenaufbaus (siehe Abschnitt 4.2.2) definiert, die sich nur von den Seitentypen her unterscheiden. Grundlegend unterschiedliche Seitentypen sind hierbei nur Auswahl- und Eingabeseiten, die dem Benutzer aber immer in der gleichen Weise präsentiert werden. Dies betrifft beispielsweise Farben, Formen und Schriftstile, wobei nur die dargestellten Informationen variieren.

Universale Einsetzbarkeit ist ein weiteres Ziel des UI-Designs. Die Lernsoftware CT2 richtet sich in erster Linie an kryptographisch Interessierte, die mehr über dieses Thema erfahren wollen. Da wie in Abschnitt 1.2 die Internationalisierung von CT2 angestrebt wurde, wurde die Möglichkeit, Übersetzungen anzubieten, über die Konfiguration ermöglicht, wodurch der Wizard einer breiteren Masse zugänglich wird. Sollte die Benutzerbefragung ergeben, dass die meisten der Befragten den Wizard als für Einsteiger geeignet befinden, so bedeutet der Wizard selbst eine Erschließung einer neuen Benutzergruppe, für die das Lösen bestimmter Aufgaben mit Hilfe von CT2 zuvor nicht möglich war.

Die Verlaufsanzeige des Wizards stellt im Wesentlichen das informative Feedback für den Benutzer zu Verfügung. An der Verlaufsanzeige erkennt er, in welchem Schritt er sich zuletzt befand und kann den vorherigen Verlauf nachvollziehen. Durch die Möglichkeit, bei Eingaben in Textfelder Gültigkeit bezüglich eines in der Konfiguration angegebenen Ausdrucks zu verlangen, ermöglicht es, dem Benutzer eine Rückmeldung darüber zu geben, ob seine Eingabe valide ist. Auch das direkte Anzeigen von Resultaten in Ergebnisseiten stellt ein informatives Feedback dar.

In Abschnitt 5.2.1 wurden bereits die Einflüsse der „Microsoft Inductive User Interface Guidelines“ in diese Arbeit erläutert. Da das Entwerfen in sich geschlossener Dialoge maßgeblich von der Konfiguration der Inhalte abhängt, war es nicht Teil dieser Arbeit, Dialoge bis ins kleinste Detail vollständig zu gestalten. Was jedoch unternommen wurde, waren Maßnahmen, die die Geschlossenheit unterstützen, wie in Abschnitt 5.2.1 ebenfalls erläutert wurde.

Die Überprüfung der Benutzereingaben in Textfelder dient der Fehlervermeidung, ebenso die Überprüfung der Werte, die in der Konfiguration angegeben wurden (siehe Abschnitt 3.4.4). Die für die Konfiguration definierte DTD gibt die Struktur der Konfiguration

vor und Entwicklungsumgebungen wie *Visual Studio 2010*, die diese DTD unterstützen, geben Hilfestellung, diese Struktur zu wahren.

Leichte Umkehr von Aktionen ist innerhalb des Wizards ohne Probleme möglich. Der Benutzer findet in jedem Pfad den er während eines Ablaufs besucht hat seine Eingaben so vor, wie er sie zuletzt belassen hat, es sei denn, er verwirft sie. Die Verlaufsanzeige bietet hierzu noch die Möglichkeit, per Doppelklick zu einem weiter zurückliegenden Schritt zurückzukehren.

Durch die zuvor erwähnte Konsistenz in Ablauf und Stil kann das innere Kontrollbedürfnis unterstützt werden. Der Benutzer sieht sich während des Ablaufs keinen abweichenden Verhaltensweisen des Wizards ausgesetzt und muss daher nicht ständig umdenken. Erfahrenere Benutzer möchten den Ablauf vielleicht beschleunigen. Da die Animationen während des Seitenaufbaus den Prozess künstlich verlangsamen, kann der Benutzer diese durch eine Einstellung abschalten. Auch die Möglichkeit der Navigation per Pfeiltasten wurde bereitgestellt, die das Wechseln von einer Seite zur Nächsten wahrnehmbar beschleunigt im Vergleich zu der Benutzung mit der Maus.

Von einer Belastung des Kurzzeitgedächtnisses durch die Benutzung des Wizards kann nicht gesprochen werden. Der Benutzer muss sich von einer Seite zur Nächsten nichts merken, er könnte sogar, sollte er eine Eingabe überprüfen wollen, weil er den Inhalt der Eingabe vergessen hat, einfach zu dieser Seite zurückkehren. Adressieren zwei Eingabefelder auf einem Pfad die gleiche Komponente und die gleiche Eigenschaft dieser Komponente, so wird die Eingabe übertragen, selbst wenn sie beispielsweise mehrere Schritte zurückliegt.

Durch diese zusammenfassende Betrachtung des Einflusses der Regeln in dieser Arbeit wurde dargelegt, dass keine dieser Regeln vollkommen unberührt blieb. Hierdurch kann abschließend festgestellt werden, dass dieser Einfluss sich in sehr vielen Aspekten des Wizards wiederfindet.

5.3. Benutzerbefragung

Wie zuvor schon erwähnt wurde im Rahmen dieser Arbeit eine Benutzerbefragung durchgeführt, um Hinweise darauf zu erhalten, wie hoch der derzeitige Wert der Ausprägung des Wizards zur Zeit der Evaluation war. Dieser Abschnitt behandelt nun die Kriterien, die Durchführung und die Auswertung dieser Benutzerbefragung.

5.3.1. Kriterien

Zur Evaluation müssen bestimmte Kriterien angesetzt werden, nach denen die Arbeit durch Benutzer bewertet werden wird. Erst nach dieser Festlegung können die Fragen entschieden werden, die im Hinblick auf diese Kriterien in den Fragebogen aufgenommen und den Benutzern gestellt werden. Die zur Evaluation entworfene Benutzerbefragung umfasst folgende Kriterien, auf die im Folgenden kurz eingegangen werden soll:

- Bedienbarkeit
- Optik

- Wert der Hilfestellungen
- Gesamtbewertung der Benutzer

Die Bedienbarkeit umfasst im Wesentlichen die Navigation innerhalb des Wizards. Auch die Möglichkeit, bestimmte Aufgaben zu lösen, fällt hierunter. Hierzu gehört auch, ob der Benutzer sich während des Ablaufs zurechtfinden kann. Unter der Optik fassen sich alle Aspekte zusammen, die das Äußere betreffen, wie beispielsweise Animationen und Farben. Mit dem Wert der Hilfestellungen sind die Erklärungen und auch die Verlaufsanzeige gemeint. Die Gesamtbewertung der Benutzer über denn Sinn und Zweck des Wizards rundet die Kriterien ab.

5.3.2. Durchführung

Für die Benutzerbefragung wurde ein Fragebogen erstellt (siehe Anhang A). Zunächst wurde gebeten, einige Angaben über die eigenen Vorkenntnisse, beispielsweise bezüglich des Umgangs mit dem Computer, Kryptographie oder des Umgangs mit CT2 zu machen. Die Angaben hierüber sind insofern interessant, als dass sie Hinweise darauf geben können, warum die drauf folgenden Antworten auf die Fragen, die den Wizard betreffen, so ausgefallen sind, wie sie ausgefallen sind. Als nächstes erfolgt eine kurze Erläuterung darüber, auf welchem Wege CT2 bezogen werden kann mit der Bitte, die Versionsnummer einzutragen. Die Versionsnummer kann hilfreich sein, falls eine Evaluation aufgrund eines Programmfehlers fehlgeschlagen ist, da hierdurch die fehlerhafte Version identifiziert werden kann. Darauf folgt eine kurze Erläuterung über die Bedienung des Wizards, die Funktionen erklärt, die nicht sofort offensichtlich sein könnten. Als nächstes folgen zwei Aufgaben, die der Benutzer mit Hilfe des Wizards lösen soll mit anschließenden Fragen, die zum Ermessen der ersten drei genannten Kriterien dienen sollen. Hiernach erfolgt die Bitte, mit den bereits zuvor vorhandenen Möglichkeiten von CT2, dieselben Aufgaben ohne Wizard zu lösen. Hierzu wird ebenso eine kurze Erläuterung gegeben. Nachdem der Benutzer nun einen Vergleich ziehen kann zu einer Bedienung ohne Wizard, folgen abschließend Fragen, die die Gesamtbewertung betreffen, zusammen mit der Möglichkeit für Kritik, Anregungen und Anmerkungen zum Wizard.

Entsprechend der verschiedenartigen Fragen wurden passende Antwortmöglichkeiten gewählt, welche die Auswertung und den Vergleich vereinfachen sollen. Den Benutzern wurden größtenteils Fragen gestellt, bei denen sie die Antwort ankreuzen können. Textfelder wurden fast ausschließlich für begleitende Fragen eingesetzt, die sich auf eine vorherige Frage beziehen und bei denen es eventuell interessant sein könnte zu erfahren, warum ein Benutzer eine bestimmte Antwort angekreuzt hat. Die übrigen Fragen umfassen Ja-/Nein-Antwortmöglichkeiten und das Ankreuzen auf einer Skala von Eins bis Fünf, repräsentiert durch Kästchen, wobei Eins die schwächste und Fünf die stärkste Ausprägung ist. Eine Drei wäre als neutral, beziehungsweise mittelmäßig, zu bewerten.

5.3.3. Auswertung

ID	Frage
F1	Welchen Beruf üben Sie aus bzw. was studieren Sie?
F2	Wie schätzen Sie ihre Fähigkeiten im Umgang mit dem Computer ein?
F3	Wie schätzen Sie ihre Vorkenntnisse im Bereich Programmierung ein?
F4	Wie schätzen Sie ihre Vorkenntnisse im Bereich Kryptographie ein?
F5	Wie schätzen Sie ihre Vorkenntnisse im Umgang mit CrypTool 2.0 ein?
F6	Haben Sie bereits mit kryptographischen Verfahren in CrypTool 2.0 gearbeitet?
F7	Falls ja: Inwiefern? Was hat Ihnen besonders gut oder gar nicht gefallen?
F8	Haben Sie bereits Erfahrungen mit einem solchen Assistenten eines anderen Programms gesammelt?
F9	Falls ja: Was hat Ihnen besonders gut oder gar nicht gefallen?
F10	Wie sind Sie nach Erledigung der ersten Aufgabe in Hinsicht auf die zweite Aufgabe vorgegangen?
F11	Wie sind Sie nach Erledigung der ersten Aufgabe in Hinsicht auf die zweite Aufgabe vorgegangen?
F12	War es leicht für Sie, diese Aufgaben zu bewältigen?
F13	Hatten Sie während der Benutzung das Gefühl, sich nicht zurechtfinden zu können?
F14	Falls ja: Wobei? Warum?
F15	Wie hilfreich fanden Sie die Erklärungen?
F16	Wie hilfreich fanden Sie die Verlaufsanzeige?
F17	Wie beurteilen Sie die Bedienung?
F18	Wie beurteilen Sie die Gesamtoptik des Wizards?
F19	Wie beurteilen Sie die Animationen des Wizards?
F20	Welche der beiden Möglichkeiten haben Sie gewählt bzw. ausprobiert? Wie sind Sie vorgegangen?
F21	War es leicht für Sie, diese Aufgaben zu bewältigen?
F22	Wie viel Zeit haben Sie schätzungsweise benötigt im Vergleich zu der Bedienung mit Wizard?
F23	Würden Sie den Wizard zur Erledigung einer solchen Aufgabe vorziehen?
F24	Aus welchen Gründen?
F25	Finden Sie den Wizard für Einsteiger geeignet?
F26	Aus welchen Gründen?
F27	Bewerten Sie den Wizard insgesamt als sinnvoll?

Tabelle 5.1.: Aufschlüsselung aller Fragen mit zugehöriger ID

Zur Auswertung der ausgefüllten Fragebögen wurde zunächst jeder Frage eine eindeutige Identifikationsnummer (kurz ID) zugewiesen. Die Fragen wurden von „F1“ für die erste Frage bis „FN“ (wobei N die Gesamtanzahl der Fragen darstellt) durchnummeriert (siehe Tabelle 5.3.3). Ebenso wurde jedem ausgefüllten Fragebogen eine eindeutige ID zugeordnet. Alle Antworten (außer den textuellen und denen zu den Vorkenntnissen) können so in eine Tabelle eingetragen und ausgewertet werden. Die textuellen Antworten, und auch die zu den Vorkenntnissen, dienen wie zuvor schon erläutert dazu, herauszufinden,

welche Intention oder welcher Grund hinter bestimmten Antworten steht und werden daher in der Auswertung nicht betrachtet, da sie nicht den derzeitigen Wert des Wizards widerspiegeln. Diese dienen als Hilfe für künftige Weiterentwicklungen des Wizards und werden dem CT2-Projektteam zur weiteren Analyse übergeben. Wurden trotz der Bitte, immer nur ein Kästchen anzukreuzen, mehrere Kästchen angekreuzt, so wurde die semantisch für den Wert des Wizards negativste Ausprägung, die angekreuzt wurde, als Wert in die Tabelle (siehe Tabelle 5.2) aufgenommen. Fehlende Angaben (die mit „/“ gekennzeichnet sind) gehen nicht in die Bewertung ein. Die in den nachfolgenden Diagrammen dargestellten Werte sind Prozentangaben, welche auf die letzten zwei Nachkommastellen gerundet wurden. Die Fragen F12 bis F20 wurden nach der Lösung der Aufgaben mit Hilfe des Wizards gestellt, während die folgenden nach der Lösung der Aufgaben ohne die Verwendung des Wizards gestellt wurden.

ID	F13	F14	F16	F17	F18	F19	F20	F22	F23	F24	F26	F28
1	5	nein	4	5	5	4	3	5	4	ja	ja	5
2	5	nein	4	5	5	4	4	5	4	ja	ja	5
3	5	nein	3	5	4	5	5	4	4	ja	ja	5
4	5	nein	3	5	5	4	2	4	4	ja	ja	5
5	5	ja	4	5	5	4	4	5	3	ja	ja	4
6	4	nein	4	3	4	4	3	4	4	ja	ja	5
7	5	nein	5	5	5	5	5	4	4	ja	ja	5
8	4	nein	4	3	4	5	3	5	1	ja	ja	5
9	5	nein	4	5	3	4	4	1	4	ja	ja	5
10	5	nein	4	5	5	4	4	4	4	ja	ja	5
11	3	nein	4	3	3	4	5	3	4	ja	nein	5
12	5	nein	5	5	5	4	3	3	5	ja	ja	5
13	5	nein	4	5	5	4	4	5	3	nein	ja	4
14	3	ja	4	5	4	4	3	3	3	ja	ja	5
15	5	nein	4	4	4	4	4	5	3	nein	nein	3
16	2	nein	4	1	3	4	2	1	4	ja	ja	5
17	5	ja	4	5	5	2	4	4	4	ja	ja	5
18	5	nein	5	5	5	4	5	4	4	ja	ja	5
19	5	nein	5	5	5	4	4	4	3	ja	ja	5
20	5	nein	3	5	4	3	2	4	3	ja	ja	4
21	5	nein	4	5	4	4	4	2	4	ja	ja	5
22	3	nein	4	3	4	4	4	2	3	nein	nein	3
23	5	nein	4	2	5	4	2	5	3	ja	ja	5
24	4	ja	5	4	4	5	5	3	3	ja	ja	5
25	5	ja	4	5	5	5	3	3	5	ja	ja	5
26	5	nein	5	5	5	5	5	4	4	ja	ja	5
27	4	ja	4	5	4	3	1	4	4	ja	ja	4
28	5	nein	5	5	5	5	5	4	3	ja	ja	5
29	5	nein	3	3	4	2	1	5	2	ja	ja	4
30	5	nein	5	4	5	3	3	4	4	ja	ja	5
31	5	nein	5	4	5	4	1	3	4	ja	ja	5
32	3	nein	3	/	4	/	/	3	/	/	/	/
33	5	ja	4	4	4	3	2	3	4	ja	ja	5
34	3	ja	2	3	4	3	2	4	4	ja	ja	2
35	1	nein	2	5	3	4	3	4	2	ja	ja	4
36	1	ja	4	4	5	5	5	4	4	ja	ja	4
37	3	ja	3	4	4	2	4	5	3	ja	ja	4
38	3	ja	2	5	4	4	3	4	4	nein	ja	3
39	5	nein	5	5	5	3	1	4	4	ja	ja	5
40	5	nein	5	5	5	5	5	4	2	ja	ja	5

Tabelle 5.2.: Auswertung der Benutzerbefragung

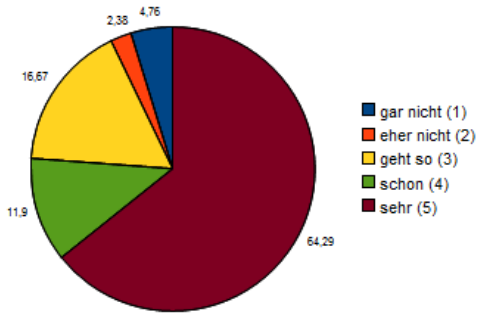


Abbildung 5.1.: F12: War es leicht für Sie, diese Aufgaben zu bewältigen?

Abbildung 5.1 zeigt ein sehr positives Ergebnis in Bezug auf die Lösbarkeit der Aufgaben mit Hilfe des Wizards. Es spielen wahrscheinlich viele Aspekte in die Beantwortung dieser Frage hinein, die ebenfalls positiv bewertet wurden und diese Aussagen unterstützen.

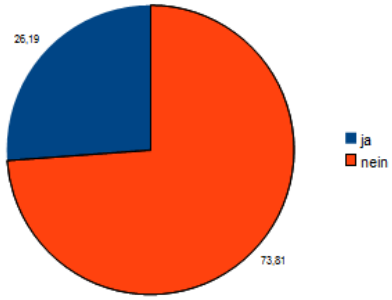


Abbildung 5.2.: F13: Hatten Sie während der Benutzung das Gefühl, sich nicht zurechtfinden zu können?

Einige der Befragten, welche die Frage F13 mit „ja“ beantworteten, erklärten diese Aussage damit, dass sie nicht auf Anhieb den Pfad gefunden haben, um die zweite der beiden Aufgaben zu lösen. Eventuell sollte eine andere Unterteilung erfolgen.

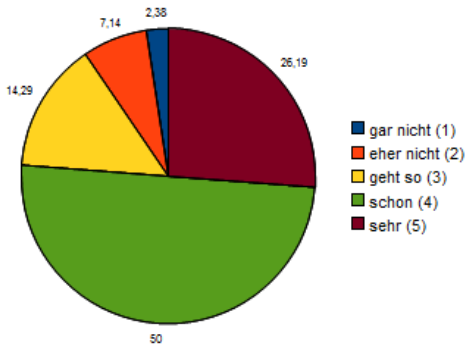


Abbildung 5.3.: F15: Wie hilfreich fanden Sie die Erklärungen?

Die Hälfte aller Befragten empfanden die Erklärungen als hilfreich, wenn auch nicht sehr hilfreich. Vielleicht sollte in Zukunft angestrebt werden, die Erläuterungen zu den einzelnen Verfahren zu erweitern und sie auf diese Weise anschaulicher zu machen.

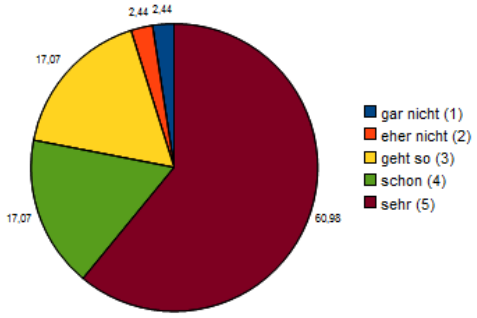


Abbildung 5.4.: F16: Wie hilfreich fanden Sie die Verlaufsanzeige?

Die Verlaufsanzeige wurde nach den Angaben zu Urteilen von den Befragten oftmals genutzt, um zur Lösung einer der Aufgaben zu einem früheren Schritt zurückzuspringen. Dieses durchaus positive Ergebnis könnte hierdurch erklärt werden.

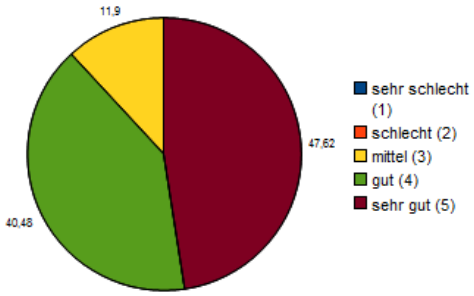


Abbildung 5.5.: F17: Wie beurteilen Sie die Bedienung?

Das Urteil über die Bedienung ist ebenfalls sehr positiv ausgefallen. Hierbei könnte die Möglichkeit, über die Verlaufsanzeige zu einem vorherigen Schritt zurückzuspringen, Einfluss genommen haben.

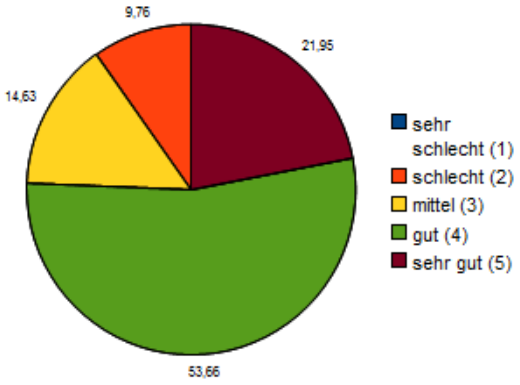


Abbildung 5.6.: F18: Wie beurteilen Sie die Gesamtoptik des Wizards?

Die meisten der Befragten beurteilten die Gesamtoptik als „gut“. Es wurden einige unterschiedliche Verbesserungsvorschläge von den Befragten gemacht, die zur Analyse herangezogen werden können.

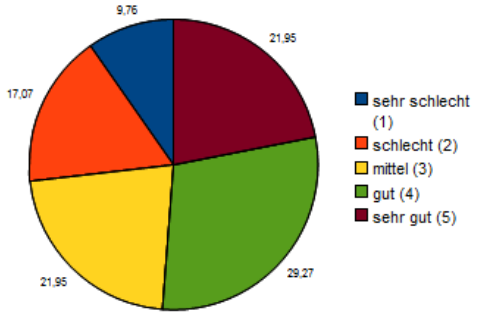


Abbildung 5.7.: F19: Wie beurteilen Sie die Animationen des Wizards?

Über die Bewertung der Animationen des Wizards lässt sich keine allgemeine Aussage treffen. Eventuell sollten hierzu andere Quellen herangezogen werden, wie beispielsweise eine spezifischere Umfrage, um Lösungsansätze zu finden.

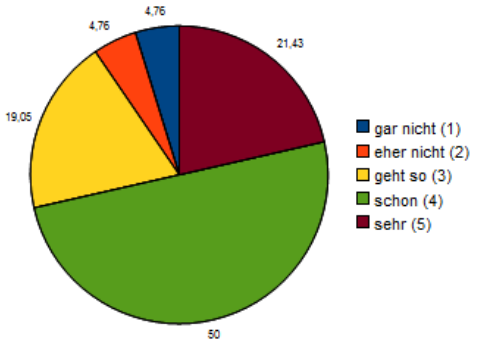


Abbildung 5.8.: F21: War es leicht für Sie, diese Aufgaben zu bewältigen?

Diese Frage wurde nach einer erneuten Lösung der Aufgaben ohne Verwendung des Wizards gestellt. Verglichen mit dem vorherigen Ergebnis zur Benutzung des Wizards ist erkennbar, dass sich die Lösung der Aufgaben schwieriger gestaltete.

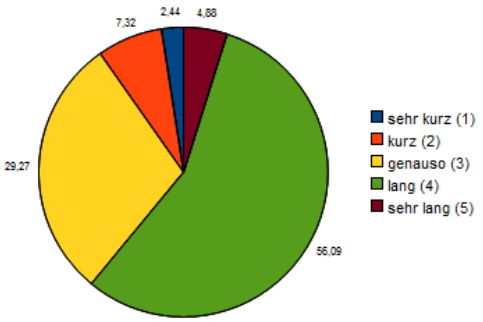


Abbildung 5.9.: F22: Wie viel Zeit haben Sie schätzungsweise benötigt im Vergleich zu der Bedienung mit Wizard?

Dass die Befragten größtenteils angaben, entweder gleich schnell, oder etwas länger für eine Aufgabe benötigt zu haben, erklärt sich eventuell durch die erleichterte Aufgabenlösung, die aber dennoch mehr Schritte erfordert.

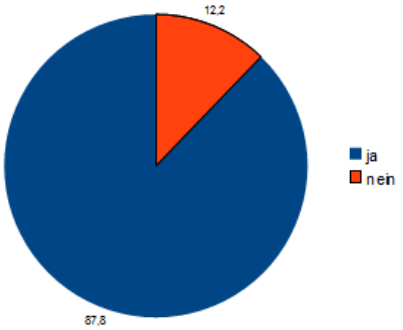


Abbildung 5.10.: F23: Würden Sie den Wizard zur Erledigung einer solchen Aufgabe vorziehen?

Die überwiegend positiven Antworten könnte sich daraus herleiten, dass die meisten der Befragten auch angaben, dass die Lösung der Aufgabe mit Hilfe des Wizards sehr einfach war.

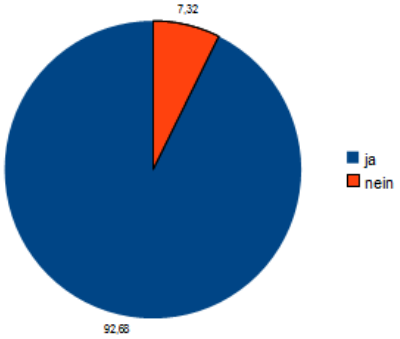


Abbildung 5.11.: F25: Finden Sie den Wizard für Einsteiger geeignet?

Dieses Ergebnis könnte sich von der von den meisten Befragten als „gut“ bis „sehr gut“ bewerteten Bedienung und der scheinbar sehr einfachen Aufgabenlösung ableiten.

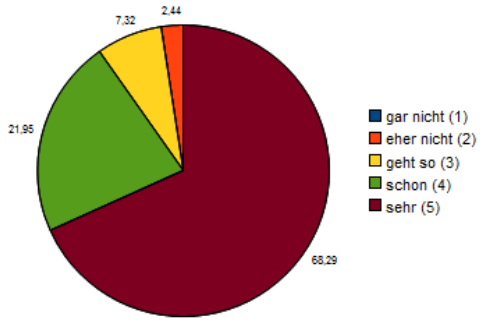


Abbildung 5.12.: F27: Bewerten Sie den Wizard insgesamt als sinnvoll?

Die überwiegend positiven Antworten auf die meisten Fragen unterstützen diese Gesamtbewertung.

5.4. Gesamtbewertung

Nachdem in den letzten Abschnitten die in die Arbeit eingeflossenen Richtlinien und die Benutzerbefragung betrachtet wurden, soll nun die Gesamtbewertung erfolgen. Diese umfasst noch einmal die Zusammenfassung der Ergebnisse, um anschließend ein Gesamtfazit über den Erfolg dieser Arbeit ziehen zu können.

Der Erfolg dieser Arbeit kann als bestätigt angesehen werden, wenn die Aufgabenstellung (siehe Abschnitt 1.2) erfüllt wurde. Diese umfasste folgende Punkte, die diese Arbeit erfüllen sollte:

- Schrittweise Heranführung des Benutzers
- Öffnen eines aufgabenspezifischen CT2-Programms
- Konfigurierbare und beliebig erweiterbare Abfrageseiten
- Unterstützung von Mehrsprachigkeit
- Orientierung am Benutzer („benutzerorientiertes Startcenter“)

Die schrittweise Heranführung des Benutzers von der Aufgabenstellung zu einer Lösung erfolgt durch die Abfrageseiten, die konfiguriert werden können. Hierbei können sinnhafte Beschreibungen angegeben werden, die den Benutzer leiten. Die Verlaufsanzeige soll den Benutzer hierbei der Navigation und seiner Orientierung unterstützen. Die schrittweise Heranführung wurde also in dieser Arbeit verwirklicht.

Das Öffnen eines CT2-Programms nach der Spezifikation durch den Benutzer ist ein weiteres Merkmal dieser Arbeit, welches umgesetzt wurde. Dies geschieht durch die zuvor in Kapitel 4 beschriebene Modellmanipulation, die es ermöglicht, eine Vorlage anzupassen und auszuführen.

In Abschnitt 3.4 wurde bereits sehr detailliert auf die Konfigurationsmöglichkeiten eingegangen, die im Rahmen dieser Arbeit ermöglicht wurden. Der gesamte Ablauf, wie auch die relevanten Inhalte, können in einer zufriedenstellenden Weise konfiguriert werden. Daher kann festgehalten werden, dass auch dieser Punkt erfolgreich umgesetzt wurde.

Die Unterstützung der Mehrsprachigkeit wurde in der Konfiguration über die Angabe von Sprachcodes ermöglicht. Die Verarbeitung der Sprachcodes wurde in Kapitel 4 erläutert und liefert damit die Begründung für eine mögliche und vollständige Übersetzung in eine beliebige Sprache. Aus diesem Grund kann festgestellt werden, dass auch dieser Punkt erfolgreich umgesetzt wurde.

Die Feststellung der Orientierung am Benutzer gestaltet sich jedoch etwas umfangreicher. Um Benutzern ein einfaches Arbeiten mit dem Wizard zu ermöglichen, wurde angestrebt, Richtlinien mit in diese Arbeit einfließen zu lassen. Wie bereits in Abschnitt 5.2 festgestellt, nahmen diese Richtlinien, soweit wie es der Rahmen zuließ, Einfluss auf diese Arbeit. Weiterhin wurde eine Benutzerbefragung durchgeführt, die Hinweise auf die derzeitige Akzeptanz des Wizards durch die Benutzer geben sollte. Bei Betrachtung der Ergebnisse wird deutlich, dass sich die meisten Antworten im neutralen bis positiven Bereich bewegen, wobei der positive Bereich überwiegt. Besonders fällt hierbei auf, dass die meisten der Befragten angegeben haben, dass die Lösung der Aufgaben mit Hilfe des Wizards sehr einfach war und im Vergleich zur Lösung der Aufgaben ohne Wizard

einfacher (siehe Abbildungen 5.1 und 5.8). Ebenfalls besonders gut angenommen wurden die Verlaufsanzeige (siehe Abbildung 5.4) und die Bedienung (siehe Abbildung 5.5). Sehr unterschiedlich wurden die Animationen bewertet (siehe Abbildung 5.7), während die Gesamtoptik jedoch vorwiegend positiv bewertet wurde (siehe Abbildung 5.6). Fast drei Viertel der Befragten gab an, dass sie während der Benutzung nie das Gefühl hatten, sich nicht zurechtfinden zu können (siehe Abbildung 5.2). Ungefähr neunzig Prozent der Befragten würden den Wizard für das Lösen einer solchen Aufgabe vorziehen und befinden ihn als für Einsteiger geeignet (siehe Abbildungen 5.10 und 5.12). Die meisten Befragten gaben abschließend an, den Wizard als sehr sinnvoll zu bewerten (siehe Abbildung 5.12). Insgesamt ergab die Befragung also, dass die derzeitige Ausprägung des Wizards keine gravierenden Schwächen aufzuweisen scheint. Das Ergebnis spricht für eine erleichterte Lösung von Aufgaben mit einer offenbar intuitiven Bedienung, da die grundlegende Bedienung des Wizards im Fragebogen bewusst nicht näher erläutert wurde (siehe Anhang A) und die meisten der Befragten dennoch angaben, dass die Aufgaben mit Hilfe des Wizards sehr einfach zu lösen gewesen seien. Abschließend kann also festgestellt werden, dass die Maßnahmen, die vorgenommen wurden, um ein einfaches Arbeiten mit dem Wizard zu ermöglichen, offenbar einen positiven Einfluss genommen haben, sodass von einer Orientierung am Benutzer gesprochen werden kann.

6. Zusammenfassung und Ausblick

In diesem Kapitel soll nun eine kurze Zusammenfassung dieser Arbeit erfolgen mit einem abschließenden Ausblick für mögliche künftige Erweiterungen und Verbesserungen, die den im Rahmen dieser Arbeit entwickelten Wizard betreffen.

6.1. Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde die Entwicklung eines benutzerorientierten Startcenters für CrypTool 2.0 vorgestellt. Hierzu wurde zunächst der Grund für diese Entwicklung dargestellt und die Anforderungen an dieses Startcenter beschrieben. Darauf folgte die Erläuterung der für diese Arbeit notwendigen Grundlagen, die sowohl verwendete Technologien als auch Richtlinien umfasste. Anschließend folgte eine Abhandlung über das Konzept und das Design dieser Arbeit, in der dargelegt wurde, wie die zuvor erläuterten die Grundlagen in diese Arbeit einfließen sollten. Auch wurden die eigens für diese Arbeit entwickelten Konzepte dargestellt. Danach erfolgte die Beschreibung der Implementierung und eine abschließende Evaluation dieser Arbeit, welche umfassend darstellt, dass das Ziel dieser Arbeit erreicht wurde.

6.2. Ausblick

In der Evaluation dieser Arbeit wurden Angaben der Benutzer über mögliche Verbesserungen bisher noch nicht betrachtet. Diese sollten für künftige Entwicklungen gesondert betrachtet werden, vor allem im Hinblick auf die Gründe, die diese angegeben haben. Auch die Vorkenntnisse der befragten Benutzer sollten hierbei betrachtet werden, da der Wizard vor allem Einsteiger adressieren soll.

Der Wizard sollte bei der Entwicklung von anderen Komponenten berücksichtigt werden. Dies ist insofern wichtig, als dass die Komponenten möglichst viele Einstellungen besitzen sollten, die dynamisch verändert werden können. Auf diese Weise wird die Benutzererfahrung bei der Verwendung des Wizards mit denen im Hintergrund ausgeführten Komponenten bereichert und es gibt so vielfältigere Möglichkeiten der Konfiguration.

Es wäre durchaus vorstellbar, sich von der Idee, den Wizard selbst als Startcenter zu verwenden, zu lösen und den Wizard vielmehr als einen von mehreren Einstiegen in das Programm zu betrachten. Nach dem Start von CT2 würde also nicht der Wizard selbst als Startumgebung erscheinen, sondern ein gesondertes Startcenter, das neben anderen Aufgaben den Wizard als Einstiegsmöglichkeit in das Programm anbietet. Dies würde unter anderem den Vorteil bieten, dass erfahrenere Benutzer direkt einen „fortgeschritteneren“ Einstieg wählen können.

Es sollte in Zukunft angestrebt werden, für jedes spezifische Szenario eine entsprechende Vorlage zu erstellen und den Wizard entsprechend zu konfigurieren, um den Umfang der Verfahren, die durch den Wizard bereitgestellt werden können, stetig zu erweitern. Die derzeitige Konfiguration wurde mit dem Ziel vorgenommen, die Funktionalitäten des Wizards zu demonstrieren. Sollte sich in der Zukunft herausstellen, dass diese Konfiguration unvollständig oder die verwendeten Einteilungen der Verfahren nicht schlüssig sind, so kann und sollte diese verändert werden.

Anhang A.

Fragebogen

Evaluation des CrypTool-2.0-Wizards

CrypTool 2.0 ist ein Lernprogramm für kryptographische und kryptoanalytische Verfahren. Der im Rahmen meiner Bachelorarbeit „Entwicklung einer benutzerorientierten Startumgebung für CrypTool 2.0“ entwickelte „Wizard“ soll den Einstieg in das Programm erleichtern. Aus diesem Grund möchte ich den Wizard im Hinblick darauf als Teil meiner Arbeit evaluieren und möchte Sie bitten, einige Fragen zu diesem Zweck zu beantworten. Alle Angaben werden anonym behandelt. Zunächst möchte ich Sie um einige Angaben bezüglich Ihrer Vorkenntnisse bitten.

Hinweis: Bitte kreuzen Sie pro Frage immer nur ein Kästchen an.

Welchen Beruf üben Sie aus bzw. was studieren Sie? ☐

Wie schätzen Sie ihre Fähigkeiten im Umgang mit dem Computer ein? sehr schlecht ☐ ☐ ☐ ☐ ☐ sehr gut

Wie schätzen Sie ihre Vorkenntnisse im Bereich Programmierung ein? gar keine ☐ ☐ ☐ ☐ ☐ sehr gute

Wie schätzen Sie ihre Vorkenntnisse im Bereich Kryptographie ein? gar keine ☐ ☐ ☐ ☐ ☐ sehr gute

Wie schätzen Sie ihre Vorkenntnisse im Umgang mit CrypTool 2.0 ein? gar keine ☐ ☐ ☐ ☐ ☐ sehr gute

Haben Sie bereits mit kryptographischen Verfahren in CrypTool 2.0 gearbeitet? ☐ ja ☐ nein

- Falls ja: Inwiefern? Was hat Ihnen besonders gut oder gar nicht gefallen?

Ein Wizard (dt. Zauberer) bezeichnet einen Assistenten, der eine Hilfestellung für eine meist mehrschrittige Dateneingabe in einem Programm bietet.

Haben Sie bereits Erfahrungen mit einem solchen Assistenten eines anderen Programms gesammelt? ☐ ja ☐ nein

- Falls ja: Was hat Ihnen besonders gut oder gar nicht gefallen?

Vielen Dank für Ihre Angaben. Nun würde ich Sie bitten, CrypTool 2.0 zu starten. Bitte verwenden Sie eine der neuesten Nightly Builds (Download: http://cryptool2.vs.uni-due.de/index.php?page=90&lm=1) und tragen die Versionsnummer hier ein: [ ] (Bspw.: 3052.1). Zur Ausführung wird .NET 4.0 benötigt. Sie können den Wizard nach dem Start über das Menü oben links (wie im Bild gezeigt) öffnen.

Ich bitte Sie nun, die folgenden Aufgaben mit Hilfe des Wizards zu lösen.

Hinweis: Sie können im Wizard auch mit Hilfe der Pfeiltasten navigieren. Beachten Sie bitte auch die Verlaufsanzeige unten. Mit einem Doppelklick auf einen Schritt können Sie zu diesem zurückkehren:

1) Caesar-Verschlüsselung:

Bei der Caesar-Verschlüsselung handelt es sich um eine auf Julius Caesar zurückgehende Verschlüsselung, die eine einfache Substitutionsregel benutzt: Die Buchstaben des Klartextes werden um einen bestimmten Wert verschoben. Der Schlüssel einer Caesar-Verschlüsselung besteht aus einem einzelnen Buchstaben.

1. Verwenden Sie den Wizard, um den Text „Ave Caesar“ mit dem Schlüssel 'K' zu verschlüsseln.
2. Verwenden Sie anschließend den Wizard, um den mit Caesar verschlüsselten Geheimtext „Oxgb obwb obvb“ und dem Schlüssel 'T' zu entschlüsseln.

Bitte beantworten Sie nun eine kleine Zwischenfrage:
Wie sind Sie nach Erledigung der ersten Aufgabe in Hinsicht auf die zweite Aufgabe vorgegangen?

2) Hashwerte erzeugen:

Eine Hashfunktion bildet beliebige Daten auf einen eindeutigen Wert fester Länge ab. Auch wenn Hashfunktionen kein Mittel zur Ver- oder Entschlüsselung sind, werden sie in der Kryptologie häufig verwendet, um Manipulation von Daten zu erkennen. Bei SHA-1 und Whirlpool handelt es sich um solche Hashfunktionen, von denen SHA-1 in der Vergangenheit weit verbreitet war.

1. Verwenden Sie den Wizard, um den SHA-1-Hashwert des folgenden Textes zu ermitteln: „Dieser Text hat einen eindeutigen SHA-1-Hashwert.“
2. Verwenden Sie den Wizard, um von dem Text aus der letzten Aufgabe den Hashwert der Whirlpool-Hashfunktion zu ermitteln.

Bitte beantworten Sie erneut folgende Frage:
Wie sind Sie nach Erledigung der ersten Aufgabe in Hinsicht auf die zweite Aufgabe vorgegangen?

Nun haben Sie Erfahrung in der Bedienung des Wizards gesammelt. Als nächstes würde ich Sie bitten, einige Angaben über Ihre Meinung zu machen.

Hinweis: Bitte kreuzen Sie pro Frage immer nur ein Kästchen an.

War es leicht für Sie, diese Aufgaben zu bewältigen? gar nicht ☐ ☐ ☐ ☐ ☐ sehr

Hatten Sie während der Benutzung das Gefühl, sich nicht zurechtfinden zu können? ☐ ja ☐ nein

- Falls ja: Wobei? Warum?

Wie hilfreich fanden Sie die Erklärungen? gar nicht ☐ ☐ ☐ ☐ ☐ sehr

Wie hilfreich fanden Sie die Verlaufsanzeige? gar nicht ☐ ☐ ☐ ☐ ☐ sehr

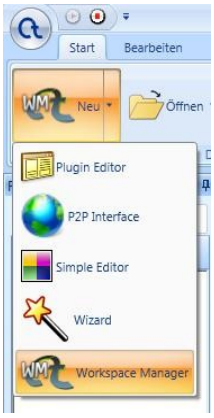
Wie beurteilen Sie die Bedienung? sehr schlecht ☐ ☐ ☐ ☐ ☐ sehr gut

Wie beurteilen Sie die Gesamtoptik des Wizards? sehr schlecht ☐ ☐ ☐ ☐ ☐ sehr gut

Wie beurteilen Sie die Animationen des Wizards? sehr schlecht ☐ ☐ ☐ ☐ ☐ sehr gut

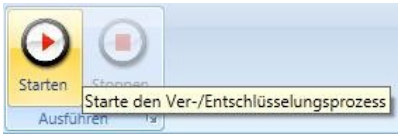
Vielen Dank für Ihre Angaben. Um nun einen Vergleich zu einer Bedienung ohne Wizard ziehen zu können, bitte ich Sie nun, die zuvor genannten Aufgaben nun ohne Benutzung des Wizards zu lösen. Hierzu gibt es zwei Möglichkeiten: Die eine Möglichkeit besteht darin, dass Sie ein eigenes kryptographisches Szenario zusammenstellen, indem Sie den Workspace-Manager (ein Editor, mit dem ein solches Szenario erstellt werden kann) wie in Bild (1) öffnen und die entsprechenden Plugins aus der Leiste links (2) in die Mitte ziehen. Bitte beachten Sie, dass Texteingabe- und Textausgabeplugins nötig sind, um diese mit dem eigentlichen Verfahren zu verbinden.

1)



2)

Bitte beachten Sie auch, dass sich eventuell nicht alle Ein- bzw. Ausgänge miteinander verbinden lassen, da diese unterschiedliche Datentypen unterstützen. Die andere Möglichkeit besteht darin, aus der linken Leiste (2) unter „Vorlagen“ eine entsprechende Vorlage per Doppelklick zu öffnen und sie anzupassen. Sie anzupassen bedeutet hier, die in der Aufgabenstellung vorgegebenen Werte in die entsprechenden Felder einzutragen. Für beide Varianten gilt, dass der Startknopf oben geklickt werden muss, um die Ausführung zu starten:



Vielleicht könnte es hilfreich für Sie sein, vor Erledigung einer Aufgabe ein paar Vorlagen zu öffnen und damit herumzuexperimentieren. Nachdem Sie nun einen Einblick in die Bedienung ohne den Wizard erhalten haben, bitte ich Sie, ein paar Fragen hierzu zu beantworten.

Welche der beiden Möglichkeiten haben Sie gewählt bzw. ausprobiert? Wie sind Sie vorgegangen?

War es leicht für Sie, diese Aufgaben zu bewältigen? gar nicht ☐ ☐ ☐ ☐ ☐ sehr

Wie viel Zeit haben Sie schätzungsweise benötigt im Vergleich zu der Bedienung mit Wizard? sehr kurz ☐ ☐ ☐ ☐ ☐ sehr lang

Würden Sie den Wizard zur Erledigung einer solchen Aufgabe vorziehen? ☐ ja ☐ nein

- Aus welchen Gründen?

Finden Sie den Wizard für Einsteiger geeignet? ☐ ja ☐ nein

- Aus welchen Gründen?

Bewerten Sie den Wizard insgesamt als sinnvoll? gar nicht ☐ ☐ ☐ ☐ ☐ sehr

Platz für Anmerkungen/Anregungen/Kritik zum Wizard:

Vielen Dank für Ihre Teilnahme!

Simone Sauer

Literaturverzeichnis

[Bra00] Bradley, Neil: *The XML companion.* Addison-Wesley, 2. Auflage, 2000.

[Hub10] Huber, Thomas Claudius: *Windows Presentation Foundation - Das umfassende Handbuch.* Galileo Computing, 2. Auflage, 2010.

[Kop10] Kopal, Nils: *Design und Entwicklung einer schnellen Laufzeitumgebung für CrypTool 2.0.* Bachelorarbeit, Universität Duisburg-Essen, 2010.

[Kü10] Kühnel, Andreas: *Visual C# 2010 - Das umfassende Handbuch.* Galileo Computing, 5. Auflage, 2010.

[Mey09] Meyer, Christian: *Evaluation der Software „CrypTool 2" unter didaktischen Aspekten.* Diplomarbeit, Universität Koblenz-Landau in Koblenz, 2009.

[Mic01] Microsoft: *Microsoft Inductive User Interface Guidelines.* URL: http://msdn.microsoft.com/en-us/library/ms997506.aspx, Februar 2001. [letzter Aufruf 15.03.2011].

[Mic11] Microsoft: *Wizards.* URL: http://msdn.microsoft.com/en-us/library/aa511260.aspx, 2011. [letzter Aufruf 16.03.2011].

[PW99] Pott, Oliver und Gunter Wielage: *XML - Praxis und Referenz.* Markt & Technik, 1999.

[SP10] Shneiderman, Ben und Catherine Plaisant: *Designing the User Interface.* Pearson, 5. Auflage, 2010.