Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Produktentwicklung 2

Digitales Bedrucken oder Bemalen von dreidimensionalen Objekten

Systemspezifikation

Horw, 10.06.2010

Projekt	Digitales Bedrucken oder Bemalen von dreidimensionalen Objekten
Dokument	Systemspezifikation
Schule	Hochschule Luzern, Technik & Architektur
Modul	TA.PREN2.F1001

Projektteam

Galliker Thomas
Studiengang Informatik (BB)
Panorama
6123 Geiss
Tel. +41 79 504 80 70
thomas.galliker@stud.hslu.ch

Ineichen Reto
Studiengang Informatik (VZ)
Jugiweg 23
6004 Luzern
Tel. +41 79 223 67 81
reto.ineichen@stud.hslu.ch

Oehring Daniel
Studiengang Wirtschaftsingenieur (VZ)
Kastanienbaumstr. 231
6047 Kastanienbaum
Tel. +41 76 463 16 10
daniel.oehring@stud.hslu.ch

Rastedter Thomas
Studiengang Maschinentechnik (VZ)
Goldmattstrasse 13a
6060 Sarnen
Tel. +41 79 708 06 85
thomas.rastedter@stud.hslu.ch

Schwarzentruber Fabian
Studiengang Elektrotechnik (VZ)
Baumgartenweg 12
6218 Ettiswil
Tel. +41 76 360 39 91
fabian.schwarzentruber@stud.hslu.ch

Vonwil Thomas
Studiengang Elektrotechnik (VZ)
Bergmatte 4
6248 Alberswil
Tel. +41 79 723 26 05
thomas.vonwil@stud.hslu.ch

Dozenten: **Prof. dipl. Ing. FH Habegger Jürg**

Letzte Änderung	10. Juni 2010, 18:47:00 Uhr

Änderungsprotokoll

Version	Datum	Autor	Beschreibung
0.1	01.03.2010	inr	Layout erstellt
0.2	18.03.2010	inf	Komponenten Diagramm & Beschreibung
0.3	20.03.2010	inr	Konfigurationsdatei & Systemübersicht beendet
0.4	22.03.2010	gat	Anpassungen Konfigurationsdatei, Sequenzdiagramm
0.5	01.04.2010	gat	Design-Entscheide zu den Anforderungen "Dateneingabe"
0.6	07.04.2010	gat	Design-Entscheide für funktionale Anforderungen fertig
0.7	16.04.2010	gat	Sequenzdiagramm "Heartbeat" erstellt
0.8	20.04.2010	inr	Nicht-funktonale Anforderungen
0.9	02.05.2010	inr	Sequenzdiagramm "Druckablauf" erstellt
1.0	08.05.2010	inr	Externe Schnittstellen dokumentiert & Dokument überarbeitet
1.1	09.05.2010	gat	Interne Schnittstellen dokumentiert, Klassen dokumentiert
1.2	25.05.2010	gat	Beschreibung Main-Klasse, MDI
1.3	26.05.2010	gat	Beschreibung Settings-Klassen
1.4	27.05.2010	inr	Rechtschreibung, Logik und Grammatik überprüft
1.5	07.06.2010	gat	Formatierung, Rechtschreibung, Inhalt überprüft
1.6	08.06.2010	inr	Dokument überarbeitet

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Zweck des Dokuments

Das Dokument definiert technisch und quantifiziert funktionalen und nicht-funktionalen Anforderungen an das System, welches im Module PREN1 & PREN2 erstellt wird. Anhand dieser definierten Merkmale kann am Ende des Projektes verifiziert werden, ob die Anforderungen erreicht wurden.
Folgendes Dokument beschreibt in konkreter Form, wie der Fachbereich Informatik gedenkt die Anforderungen der Kundenanforderung [1] technisch zu realisieren.

1.2 Gültigkeitsbereich

Die Gültigkeit des Dokumentes beschränkt sich auf das Projektmodul Produktentwicklung 2.

1.3 Definitionen und Abkürzungen

Abkürzungen	Erklärung
UC	Use Case / Anwendungsfall
GUI	Graphical User Interface
SW	Software
PREN	Produktentwicklung (Projektmodule der HSLU)
MC	Microcontroller
Designer	Software-Name der Projekt-Software um den 3D-Object-Printer zu verwenden
FS	Filesystem
MDI	Multi-Document Interface

Tabelle 1: Abkürzungen

1.4 Referenzen

ID	Version	Titel	Dateiname
1	0.4	Kundenanforderungen	Kundenanforderungen.doc
2	1.3	Aufgabenstellung	Aufgabenstellung_PREN_F_10_V1_3.pdf
3	1.5	Anforderungsliste	Anforderungsliste.doc
4	1.0	Gesamtkonzept	1_Team_16_Gesamtkonzept_VT3_1.0.pdf
5	0.8	MC Konzept	ET-Konzept.doc

Tabelle 2: Referenzen

2 Systemübersicht

Die Software (auch Designer genannt) realisiert die Schnittstelle zwischen Benutzer und 3D-Object-Printer. Sie ermöglicht dem Benutzer das Gestalten von Drucksujets.

Das System besteht aus folgenden Komponenten:

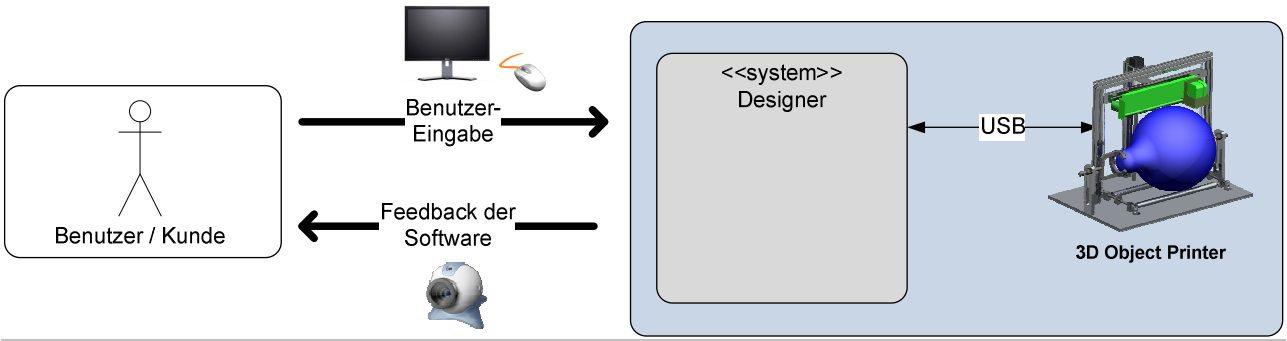


Abbildung 1: Systemübersicht (Kontextmodell)

Komponente	Beschreibung
Benutzer / Kunde	Der Eigentliche Benutzer der Software (fortan Designer genannt). Er ist mit dem Kunde gleichzustellen. Verwendung des Designers um Motive zu erstellen und auf Objekte zu drucken.
Designer	Die eigentliche Software, welche dem Benutzter zur Verfügung steht. Mit ihr kann der Benutzer / Kunde Drucksujets erstellen und modifizieren. Über den Designer wird ein Druckauftrag ausgelöst. Die komplette Druckauftrags-Abwicklung, samt Überwachung des 3D-Object-Printers wird vom Designer gesteuert und koordiniert.
3D-Object-Printer	Hierbei handelt es sich um den eigentlichen Drucker. Diese Komponente umfasst sehr viele Teilsysteme, wie der Microcontroller, die ganze Mechanik, die Druckerlogik usw. Jedoch wird im Dokument nur auf die nötigen Komponenten weiter drauf eingegangen. Die Kommunikation erfolgt mittels USB-Kabel zum 3D-Object-Printer (jedoch seriell).
Benutzer-Eingabe	Diese Kommunikation stellt den Input der Software-Komponente (Designer) dar. Via Bildschirm, Maus und Tastatur können Eingaben gemacht werden.
Feedback der Software	Gegenüber dem Input steht der Output. Dieser wird einerseits mittels einer eingebauten Cam (Livestream zur Druckverfolgung) und andererseits mit hilfreichen Softwarefeedbacks gegeben.

Tabelle 3: Software-Komponenten

Die USB-Schnittstelle zwischen dem Designer und dem 3D-Object-Printer wurde aus folgendem Grund gewählt: Einerseits verfügt jeder Laptop heutzutage über einen derartigen Anschluss, des Weiteren kann so eine zweite Verkabelung zum 3D-Object-Printer vermieden werden. Wie in der Systemübersicht (Abbildung 1: Systemübersicht (Kontextmodell)) zu sehen ist, kann so der Microcontroller als auch die Druckerlogik angesteuert werden. Dazu wird ein USB-Hub auf Seiten der Elektrotechnik verwendet. Welcher die Aufgabe hat, die entsprechenden Kontroll- und Steuersignale für sich abzufangen und der eigentliche Druck-Auftrag an die Druckerlogik des 3D-Object-Printers weiterzuleiten.

3 Architektur-Modelle

Dieses Kapitel beschäftigt sich mit der Architektur der Software. Anhand von verschiedenen Modellen, welche vereinfacht die Realität beschreiben, wird die gewählte Architektur beschrieben. Kriterien für oder gegen ein Modell werden ausgewertet und der Entscheid im Kapitel: 5 - Softwareanforderungen und Design-Entscheide beschrieben.

3.1 Komponenten-Beschreibung

Das folgende Diagramm illustriert aus welchen Komponenten der Designer besteht. Es ist ersichtlich, dass für die Kommunikation zum 3D-Object-Printer (Microcontroller) nur die Komponente *COM* zuständig ist, welche dem Designer das Interface *ICOM* zur Verfügung stellt.

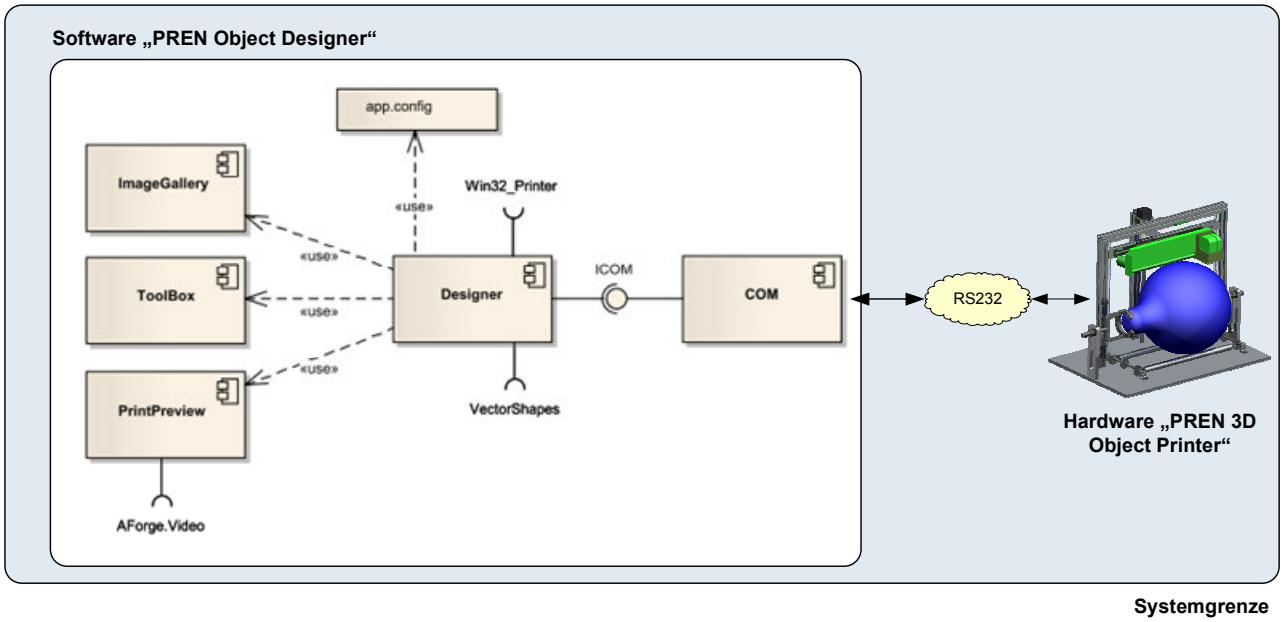


Abbildung 2: Komponenten Diagramm

Komponente	Beschreibung
app.config	Dieses Artefakt enthält die Konfiguration für sämtliche veränderbaren Einstellungsmöglichkeiten des Designers. Der Designer bietet eine GUI-Oberfläche um die Konfiguration einfach und elegant vorzunehmen. Der Aufbau der Konfigurationsdatei "app.config" wird im Kapitel 3.4 - Konfigurationsdatei genauer spezifiziert.
COM	Diese Komponente hat als einzige Aufgabe, Daten via RS323 Schnittstelle vom Designer an den Microcontroller weiterzuleiten und entsprechende Signale vom Microcontroller zu empfangen. Es handelt sich dabei um Status- und Kontroll-Informationen als auch um Steuersignale, welche zwischen Designer und Microcontroller übermittelt werden. Damit die Komponente mit dem MC kommunizieren kann, müssen verschiedene COM-Einstellungen über den Designer (GUI-Oberfläche) vorgenommen werden. Um diese Funktion dem Designer bereit zustellen, bietet die Komponente ein sauber

	definiertes Interface an. Das Interface wird im Kapitel 4.2 - Interne Schnittstellen spezifiziert.
Designer	Der Designer ist die Hauptkomponente der Software. Die Komponente bildet das Zentrale Element unserer Software, welches die anderen Komponenten für den vollen Funktionsumfang benötigt. Auf dem Designer kann ein Bild erstellt und modifiziert werden. Die Komponente verwendet für die Zeichnungsfunktionen die „VectorShapes“ Library. Weiter wird die Library „Win32_Printer“ verwendet um die Kommunikation zur Druckerlogik zu gewähren. Über diese Schnittstelle wird der Druckauftrag, d.h. nur das Bild, an den 3D-Object-Printer übermittelt.
ImageGallery	Diese Komponente ist für das persistente Speichern von Bildern zuständig. Das Speichern der Bilddateien erfolgt auf einfache Art, indem ganz simpel auf dem lokalen Filesystem ein eigens dafür erstelltes Directory verwendet wird. Beim Start des Designers werden von dieser Komponente die im FS enthaltenen Bilddateien eingelesen und sortiert dargestellt. Dies nur, falls der Benutzer die entsprechende Funktion aktiviert hat.
PrintPreview	Die PrintPreview generiert eine Vorschau, welche das entsprechende Sujet virtuell auf dem eingespannten Objekt darstellt. Für diese Funktion wird eine Cam verwendet, welche am 3D-Object-Printer fix befestigt wird.
ToolBox	Bietet übliche Zeichnungswerkzeuge im Designer an.

Tabelle 4: Komponentenbeschreibung

3.2 Sequenzielle Prozeduren

Folgende Sequenzdiagramme stellen die wesentlichsten Programmabläufe grafisch dar. Zu den wichtigsten sequenziellen Prozeduren im Projekt werden einerseits der eigentliche Druckablauf sowie die periodische Heartbeat-Signalisierung gezählt.

3.2.1 Druckablauf

Das folgende Sequenzdiagram zeigt den Ablauf eines Druckes aus Sicht der Designers. Zur Steigerung der Übersichtlichkeit enthält das Diagramm nicht alle Statusbefehle, welche zwischen Designer und Microcontroller ausgetauscht werden und konzentriert sich nur auf die wesentlichen Befehle. Die Befehlsliste ist im Elektrotechnik Konzeptdokument [5] detailliert definiert. Des weitern soll an dieser Stelle auf die Finite State Machine (FSM) des MC verwiesen [5] werden. In der nachfolgenden Beschreibung wird versucht, den sequenziellen Ablauf des Druckvorgangs in Worte zu fassen.

Die wichtigen Aspekte des Diagramms sind die folgenden Abhängigkeiten zwischen Designer und MC:

- Vor dem Auslösen eines neuen Druckauftrags müssen einige Vorbedingungen erfüllt sein: Vorerst mal muss der 3D-Object-Printer angeschlossen und gestartet sein, sodass die Software über den automatisch erkannten COM-Port kommunizieren kann. Weiter muss auch die integrierte Videokamera erkannt werden, bevor ein Druckjob gestartet werden kann.
- Sobald der Microcontroller bereit ist, sendet er das Status Signal STATUS_READY zur Software.
- Hat der Benutzer ein Druckauftrag gestartet, wird zuerst mittels CONTROL_PRINTINIT ein Kommando an den MC abgesetzt, um dem MC über den Start eines neuen Druckjobs zu informieren.

- Die Software kann den Druckauftrag erst an die Druckeinheit des 3D-Object-Printer senden, wenn sie vom MC das Status Signal STATUS_INITIALIZED empfangen hat. Bevor dieses eintritt wird die Software noch mit STATUS_INITIALIZING über den Beginn der Initilaisierungsphase informiert.
- Während des gesamten Druckvorgangs hat der Benutzer die Möglichkeit den laufenden Druckjob über das angezeigte Dialogfenster zu verfolgen. Der Start des Druckvorgangs wird vom MC mit dem Signal STATUS_PRINTING angezeigt.
- Sobald der MC das Status Signal STATUS_PRINTED an den Designer gesendet hat, wird die Software dem Benutzer eine entsprechende Meldung anzeigen und das Druckfenster schliessen.

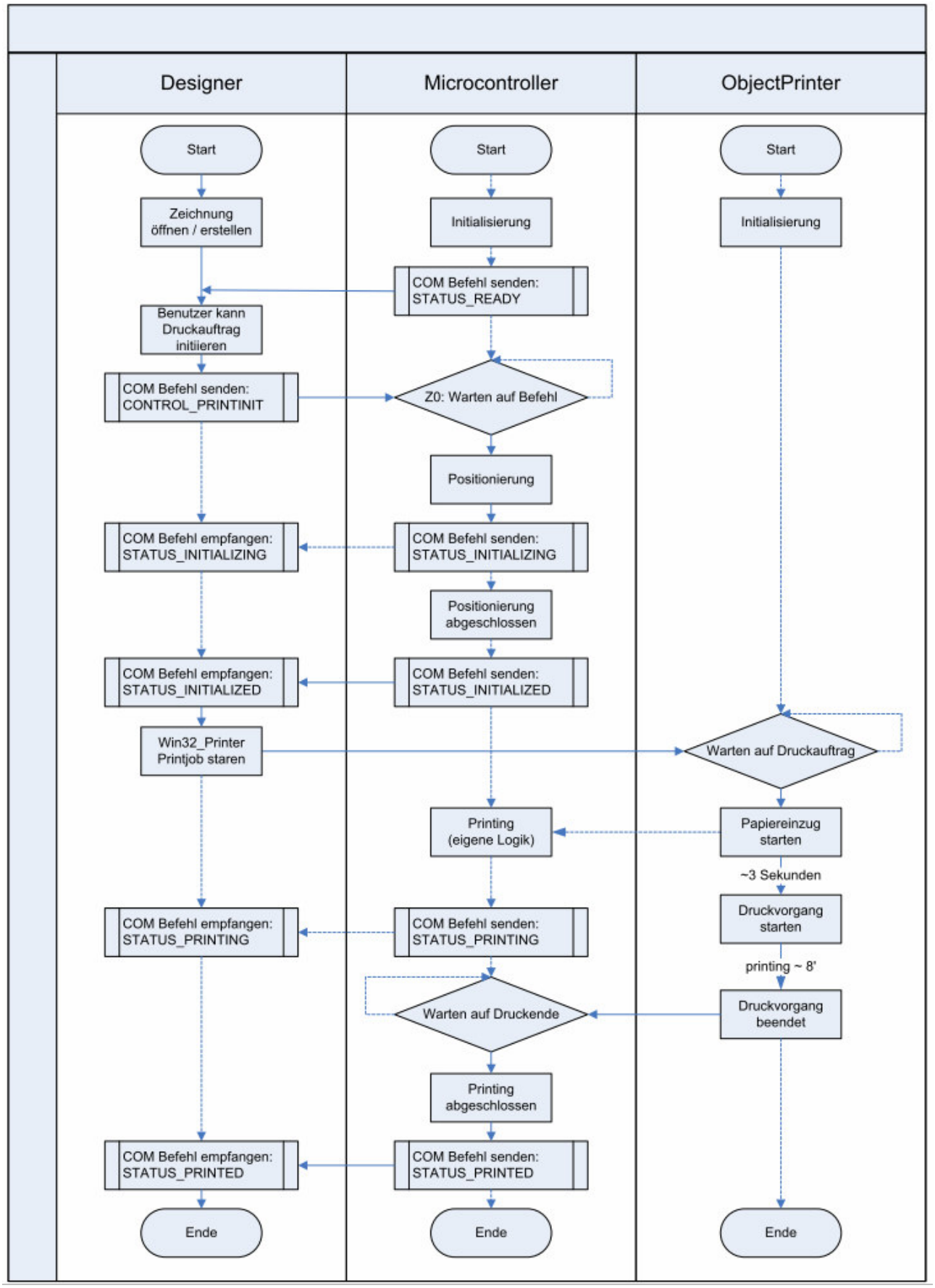


Abbildung 3: Sequenz-Diagramm: Druckauftrag

3.2.2 Heartbeat-Signalisierung

Bei der Heartbeat-Signalisierung wird automatisch beim Starten des Designers ein Thread (siehe "StartHeartbeat") gestartet, welcher in konfigurierten Zeitabständen eine HEARTBEAT Status-Nachricht an den MC des 3D-Object-Printer sendet. Kann dieser die Status-Nachricht innerhalb der im Timer festgelegten Antwortzeit quittieren (siehe "raiseDataEvent"), so wird dem Designer zurückgemeldet, dass der 3D-Object-Printer "Online" ist. Kann der MC jedoch innerhalb der definierten Zeitperiode keine Antwort zurück senden, so wird diese Tatsache im Designer mit einer "Offline"-Meldung angezeigt.
Diese Prozedur wird so oft ausgeführt, bis die COM Klasse zerstört, die close-Methode von COM ausgeführt oder die Software beendet wird.

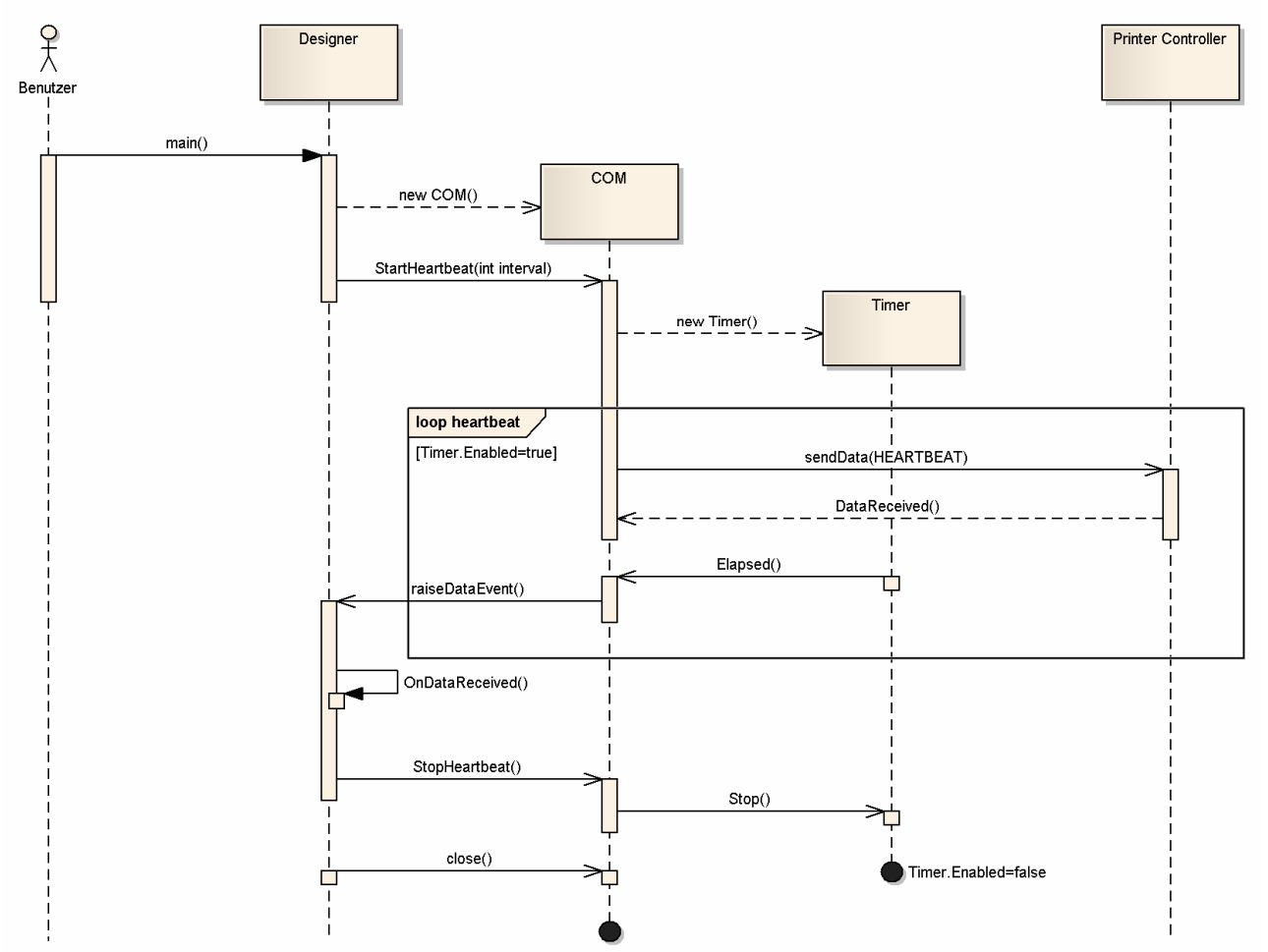


Abbildung 4: Sequenz-Diagramm: Heartbeat-Signalisierung

3.3 Klassen

Die Software besteht bis kurz vor dem Release 1.0 aus nicht weniger als vierzig verschiedenen Klassen. Die Klassen wurden jeweils weitgehend mit C# Code-Kommentar ausgeschmückt. Bei einigen elementaren Klassen ist es jedoch trotzdem sinnvoll, weitergehende Erklärungen festzuhalten. In den nachfolgenden Abschnitten wird versucht, die Funktionsweise dieser Klassen in Worte zu fassen.

3.3.1 Main

Wie jede Software hat auch die PREN Designer Software einen definierten Einstiegspunkt. Im vorliegenden Fall ist dies die Main-Klasse. Sie beherbergt viele wichtige Komponenten, wie z.B. das Menü, die Komponente der Zeichenwerkzeuge, die ImageGallery, die Statusliste und vieles mehr. Die Main-Klasse implementiert auch einen ansprechenden SplashScreen beim Starten der Anwendung.
Im Zentrum von Main steht eine leere Fläche, auf welche eine oder mehrere Zeichnungen (so genannte PaintForms) geladen werden können.

Eine wichtige Eigenschaft der Main-Klasse ist der MDI Container (MDI). Main ist also ein Träger von MDI Windows. Konkret bedeutet dies, dass in Main mehrere "Documents" – oder in diesem Fall eben Zeichnungen – simultan offen gehalten werden können. Platziert werden die Zeichnungsfenster auf der erwähnten leeren Fläche im Zentrum der Anwendung.
Für den Benutzer bringt MDI den Vorteil, dass er mehrere Zeichnungen (quasi) gleichzeitig bearbeiten kann. Er hat über entsprechende Menüfunktionen auch die Möglichkeit, bestimmte Operationen auf sämtliche offenen Fenster anzuwenden.

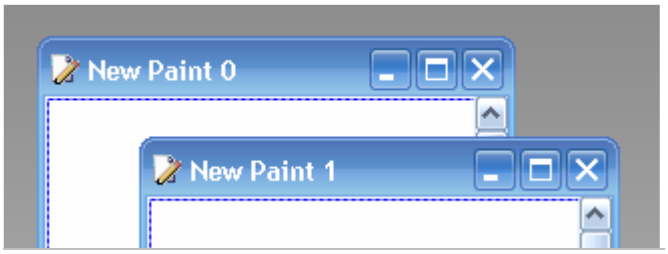


Abbildung 5: Main fungiert als Träger von MDI-Windows

Bei der Entwicklung von MDI-basierten Applikationen gilt es aber zu berücksichtigen, dass jedes Fenster zu einem beliebigen Zeitpunkt aktiv sein kann. Damit der Benutzer seine Zeichenmanipulationen auch tatsächlich in der selektierten Zeichnung vornehmen kann, muss festgehalten werden, welches die aktive Zeichnung ist. Dafür wurde der Event "MdiChildActivate" genutzt, welcher vom sog. MDI-Parent (hier: die Main Klasse) bereitgestellt wird. In diesem Event wird die Referenz auf das aktuell selektierte Zeichnungsfenster ausgelesen und in der Variablen "lastActiveMdiChild" abgelegt. Zudem – und das ist nicht weniger wichtig – wird in diesem Event die ToolBox über den Wechsel der aktiven Zeichnung informiert. Die ToolBox muss stets in Kenntnis davon sein, auf welche Zeichnung die gewünschten Zeichenmanipulationen ausgeführt werden müssen.

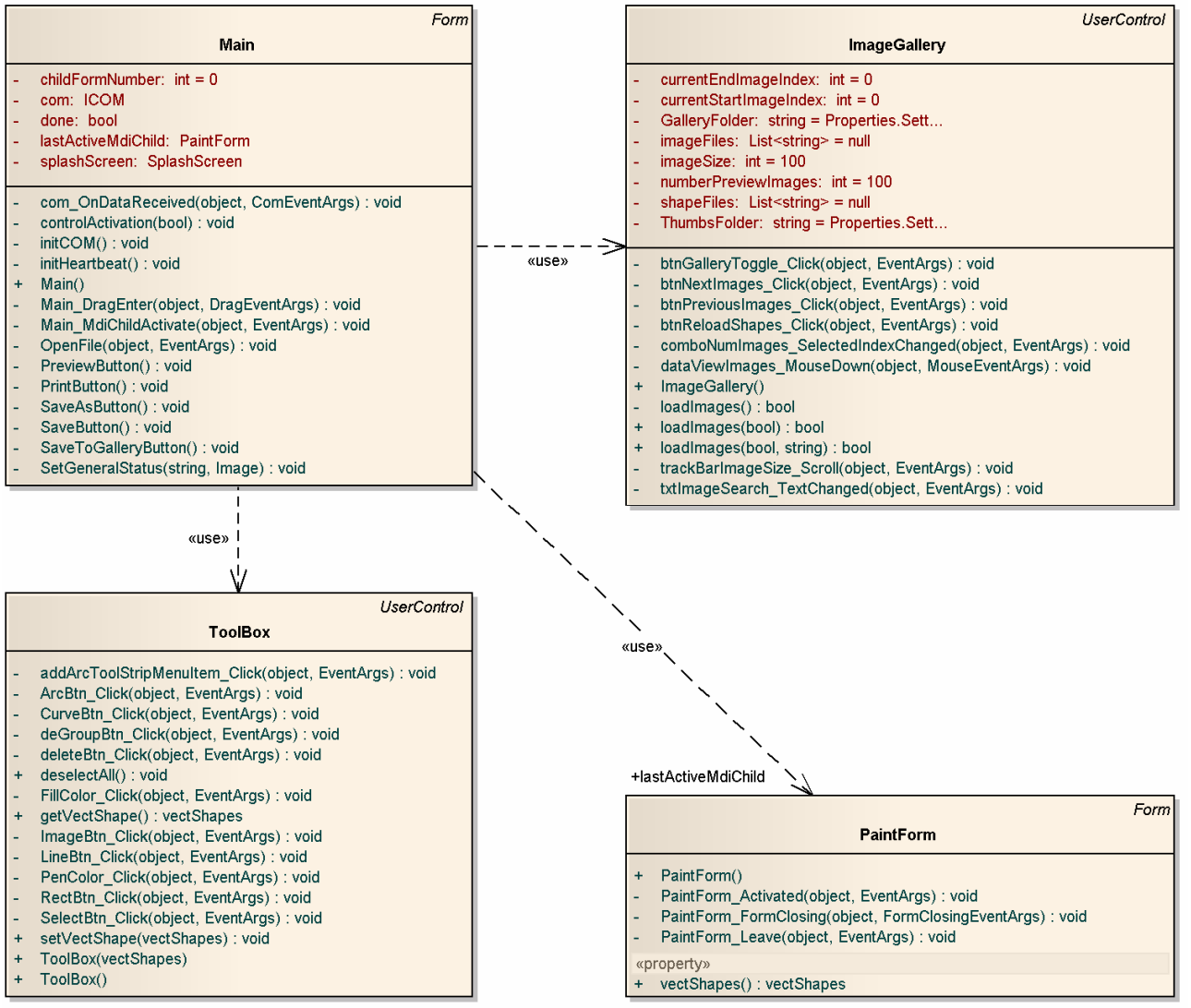


Abbildung 6: Klassendiagramm der Hauptklasse Main

3.3.2 Settings und SettingsForm

Einstellungen können im 3D-Object-Printer über die Dialogbox "SettingsForm" vorgenommen werden. "SettingsForm" stellt lediglich die grafische Benutzerschnittstelle für die einzelnen Einstellungen bereit. Die Steuerelemente auf SettingsForm müssen dem Benutzer eine einfache Möglichkeit bieten, die Konfiguration an die persönlichen Bedürfnisse anzupassen.

Das effektive Lesen und Schreiben der Einstellungswerte wird über die Klasse "Settings" vollzogen. "Settings" ist letztlich auch dafür verantwortlich, dass die Einstellungen in Form einer XML-Konfigurationsdatei ("app.config") im Dateisystem abgelegt werden. Des Weiteren haben andere Klassen die Möglichkeit, mit der Settings-Klasse zu interagieren: Die Klasse COM nutzt beispielsweise den "OnPropertyChanged"-Event, welcher von System.Configuration.Properties bereitgestellt wird. Mit Hilfe dieses Events bemerkt COM wenn der Benutzer die COM-Schnittstelle umkonfiguriert hat. Ist dies der Fall, so wird die Instanz von COM automatisch neu initialisiert und somit die Benutzereinstellung – ohne Neustart der Anwendung – übernommen.

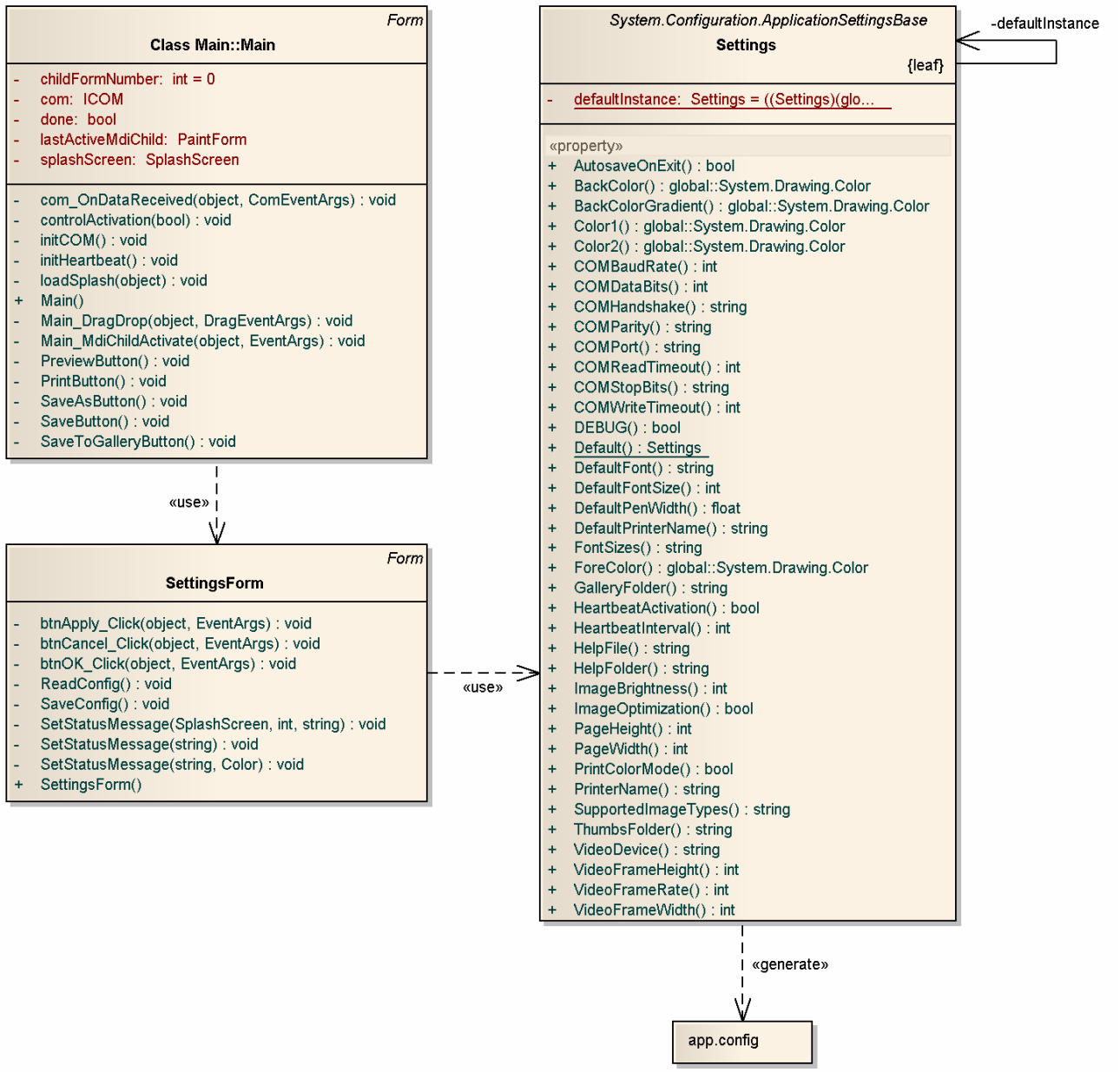


Abbildung 7: Klassendiagramm der Hauptklasse Main

3.3.3 COM und ICOM

Das Herzstück der Software befindet sich in den Kommunikationsklassen. Deshalb wird dieses Konstrukt nachfolgend detailliert dokumentiert.

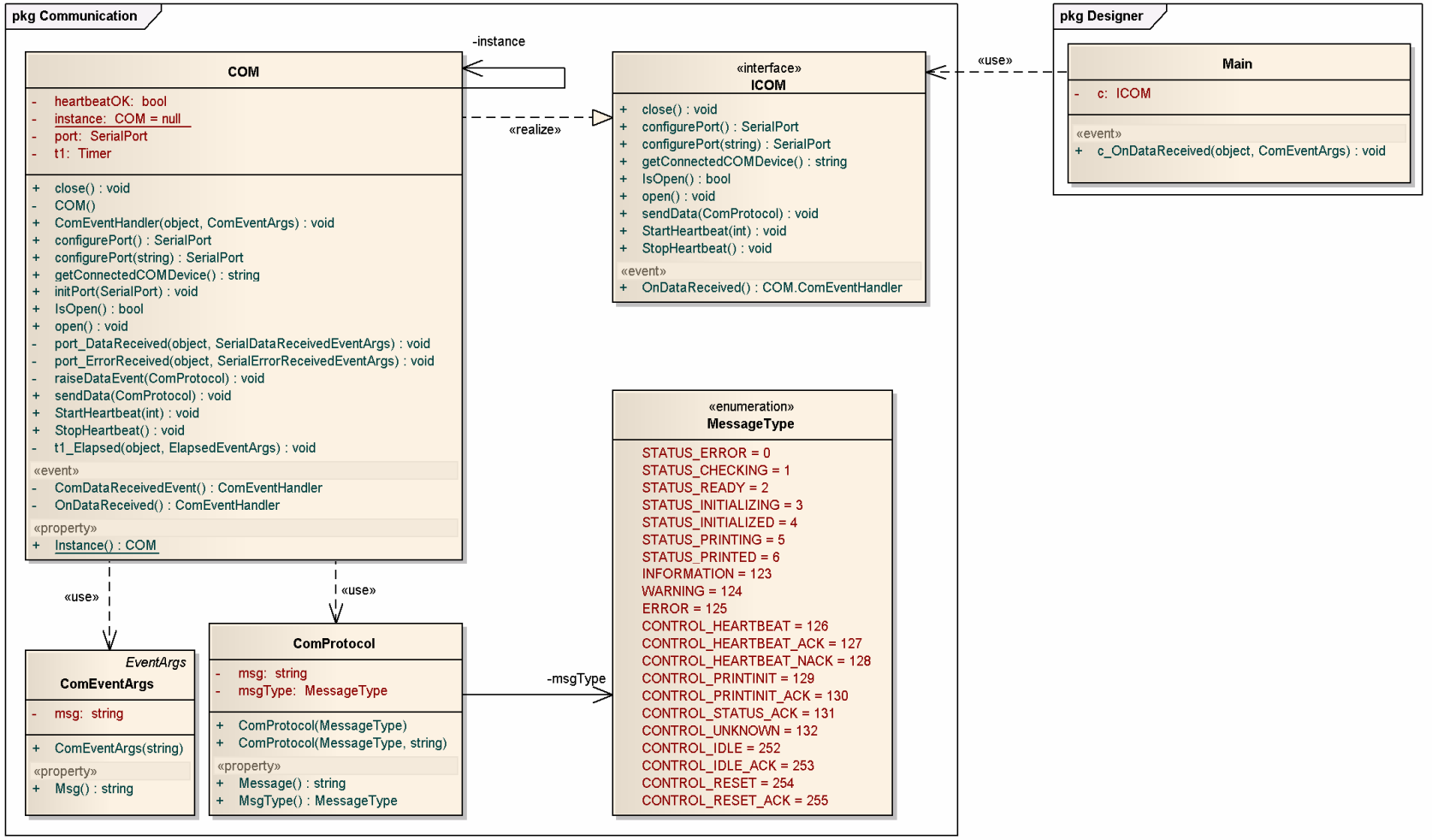


Abbildung 8: Klassendiagramm der Kommunikationsklassen ICOM und COM

Das Interface ICOM wird von der Klasse COM implementiert. Damit wird die Kommunikation mit dem MC komplett vom Designer getrennt, wodurch eine erhöhte Spezialisierung der Klassen erreicht wird und dadurch die Wartbarkeit der Software erheblich gesteigert.

3.3.3.1 Unified Event Handling

Die ganze Kommunikation mit COM basiert auf Events. Dabei verarbeitet COM verschiedenste Events wie beispielsweise "port_DataReceived" für eingehende SerialPort-Daten, "port_ErrorReceived" für eingehende SerialPort-Fehlermeldungen oder etwa t1_Elapsed, wenn der Timer t1 einen Überlauf meldet. Damit all diese Event-Schnittstellen nicht allesamt via ICOM publiziert werden müssen, wurde ein neuer Event geschaffen, genannt "OnDataReceived". Das ist ein allgemeiner Event, welcher über ICOM publiziert wird und von einer Client Anwendung (in unserem Fall die Designer Software) genutzt werden kann. Sämtliche Informationen, welche für die Subscriber Anwendung notwendig sind, werden nun über diesen abstrahierenden "OnDataReceived"-Event übertragen.

3.3.3.2 Kommunikationsprotokoll ComProtocol

Neben der Vereinheitlichung der Events wurden natürlich auch die Event-Nachrichten vereinheitlicht. Dafür wurde ein eigenes Kommunikationsprotokoll, "ComProtocol", entwickelt. Dieses Protokoll kann grundsätzlich zwei verschiedene Informationstypen übertragen: Eine verbindliche Status- oder Control-Nachricht wird in jedem Fall versendet. Dafür wurde die aus der gemeinsam erarbeiteten Schnittstellenspezifikation entstandene Enumeration "MessageType" verwendet. Zudem ist es möglich mit ComProtocol auch Nachrichten vom Typ "String" zu übermitteln. Dieses Feld wird beispielsweise zur Übermittlung von Fehlermeldungen oder Zusatzinformationen genutzt.

3.3.3.3 Singleton Design Pattern mit Thread-Safety

Wie der obigen Abbildung zu entnehmen ist, implementiert die Klasse "COM" das Singleton Entwurfsmuster. Dieses verhindert, dass von der Klasse "COM" mehr als ein Objekt erzeugt werden kann. COM verfügt dementsprechend nur über einen private-Konstruktor. Wer eine Referenz auf das globale COM Objekt möchte, kann dieses über das Property "Instance" abholen.
Die Singleton Implementierung hat sich bereits nach kurzer Entwicklungszeit als grossen Erfolg herausgestellt. So kann beispielsweise in Dialog-Fenstern einfach auf COM zugegriffen werden – ohne dabei aufwändige Initialisierungsmethoden anzustossen oder den OnDataReceived-Event erneut anzubinden.
Zudem wurde sichergestellt, dass die Erzeugung eines COM Objekts Thread-sicher ist. Zwei konkurrierende Thread haben also keine Möglichkeit, versehentlich zwei Objekte der Klasse COM zu erstellen.

3.4 Konfigurationsdatei

In diesem Unterkapitel wird definiert wie das Konfigurations-File für den Designer aussieht und wie und was geändert werden kann, oder welche Konfigurationen zwar konfiguriert werden, aber für den Standart Benutzer eher nicht von Interesse sind.
Grundsätzlich wird die Software so geplant und realisiert, dass das Konfigurations-File via Designer-GUI modifiziert werden kann. Es wird eine Standart Konfiguration zur Verfügung gestellt, welche auch nachträglich wieder eingespielt werden kann.
Das Konfigurations-File unterscheidet Applikations-Settings und User-Settings.

3.4.1 Konfigurationsdialog

Mit Hilfe einer Handskizze sowie einer kurzen Beschreibung wird nachfolgend versucht, die Konfigurationsmöglichkeiten sowie deren Visualisierungsmöglichkeiten aufzuzeigen.

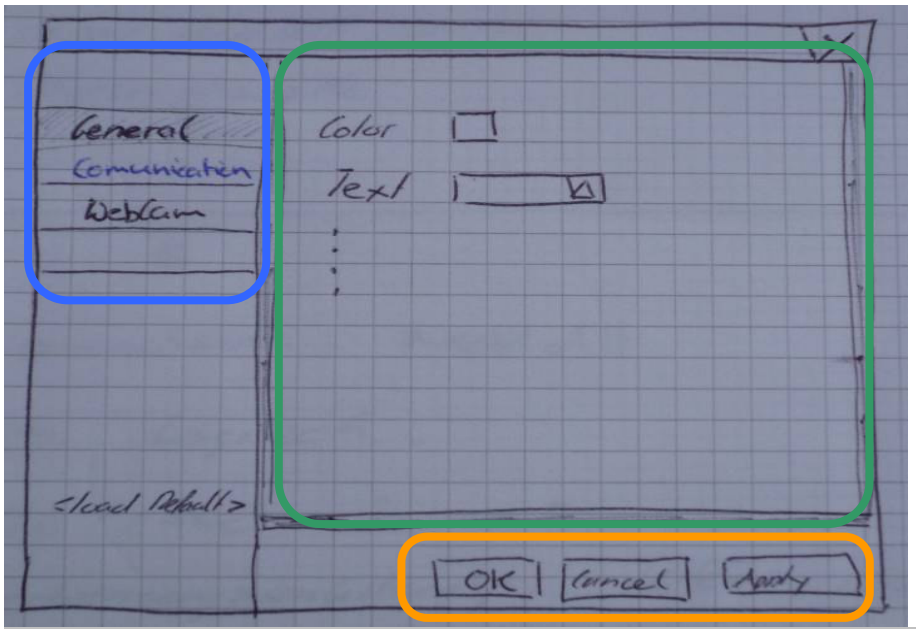


Abbildung 9: Einstellungen werden im Konfigurations-GUI angepasst

Der modale Konfigurationsdialog wird im Wesentlichen in drei Sektionen unterteilt:

- Eine Gliederung der verschiedenen Konfigurationskategorien. Die Kategorien ermöglichen eine bessere Übersicht der Konfigurationsparameter. Die Informationsmenge, welche den Benutzer erreicht, kann bewusst gedrosselt werden (Usability Aspekt).
- Ein Bereich auf welchem die verschiedenen Konfigurationsparameter dargestellt werden können. Eine sinnvolle Menge an Optionen pro Seite muss gefunden werden. Notfalls kann eine Checkbox für fortgeschrittene Benutzer eingerichtet werden, über welche eine erweiterte Auswahl an Konfigurationsoptionen angezeigt werden kann.
- Die Schaltflächen zum Speichern/Verwerfen der vorgenommenen Einstellungsänderungen: "OK" speichert und schliesst das Fenster, "Cancel" verwirft die vorgenommenen Änderungen, "Apply" übernimmt die vorgenommenen Änderungen ohne das Fenster zu schliessen.

3.4.2 Einstellungsmöglichkeiten

Zu den wichtigsten Einstellungsmöglichkeiten gehören mitunter:

- Standardwerte für Farbeinstellungen
- Standardwerte für Schriftart / Schriftgrösse
- COM-Einstellungen für die Verbindung zum Microcontroller
- Optionen für die Image Gallery
- Optionen für die Anzeige der Druckvorschau bzw. des Live Streamings

3.4.3 Deployment

Das Konfigruations-File wird zusammen mit dem Designer ausgeliefert und muss beim ersten Start zwingend vorhanden sein. Danach werden die User-Settings in ein separates File in das entsprechenden User-Profile kopiert. (*%USERPROFILE%*\Local Settings*Application Data**<Company Name>**<appdomainname>_<eid>_<hash>**<verison>**user.config*, Quelle: MSDN).

Durch das Speichern der User-Settings im Profil wird gewährleistet, dass jeder Benutzer seine eigene Konfiguration des Designers vornehmen kann.
Eine Routine prüft beim ersten Start, an welchem COM Port der 3D-Object-Printer angeschlossen ist und schreibt den entsprechenden Port automatisch in die Konfiguration.

3.4.4 Datenstruktur

Das Konfigurations-File wird in zwei grundsätzliche Sektionen unterteilt:

- **userSettings**
  Benutzereinstellungen können vom Benutzer während der Laufzeit des Programms angepasst werden. Die Einstellungen werden im Profil des Benutzers abgelegt.
- **applicationSettings**
  Anwendungseinstellungen dienen der persistenten Festhaltung von Informationen, welche im Verlauf der Benutzung der Software nicht geändert werden müssen. Änderungen an Anwendungseinstellungen werden ausschliesslich vom Entwickler und nicht während der Laufzeit vorgenommen. "ApplicationSettings" ersetzen seit .Net Framework 2.0 die Registry Hives, welche jeweils für Anwendungen angelegt wurden.

Nachfolgende Abbildung illustriert ein Auszug aus der Konfigurationsdatei:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="applicationSettings"></sectionGroup>
        <sectionGroup name="userSettings"></sectionGroup>
    </configSections>
    <applicationSettings>
        <designer.Properties.Settings>
            <setting name="BackColor" serializeAs="String">
                <value>238, 237, 240</value>
            </setting>
            <setting name="ForeColor" serializeAs="String">
                <value>WindowText</value>
            </setting>
            <setting name="GalleryFolder" serializeAs="String">
                <value>gallery</value>
            </setting>
        </designer.Properties.Settings>
    </applicationSettings>
    <userSettings>
        <designer.Properties.Settings>
            <setting name="Color1" serializeAs="String">
                <value>Black</value>
            </setting>
            <setting name="Color2" serializeAs="String">
                <value>White</value>
            </setting>
            <setting name="DefaultFont" serializeAs="String">
                <value>Arial</value>
            </setting>
            <setting name="COMBaudRate" serializeAs="String">
                <value>9600</value>
            </setting>
            <setting name="DefaultFontSize" serializeAs="String">
                <value>10</value>
            </setting>
        </designer.Properties.Settings>
    </userSettings>
</configuration>
```

Abbildung 10: Struktur der Konfigurationsdatei

3.5 Softwareverteilung

Der Entwickler jeder Software sollte sich bis spätestens vor dem ersten Feldtest Gedanken über die Softwareverteilung gemacht haben. Bei verteilten Systemen sollte die Verteilung der Software bereits im Konzept thematisiert werden. Im Fall von PREN Designer war das nicht nötig, da bereits in der Konzeptphase absehbar war, dass auch im besten Fall nur ein einziges Computersystem für die Steuerung des 3D-Object-Printers zuständig sein würde. Insofern ist die Verteilung der Software nicht allzu komplex – aber dennoch beachtenswert.

Der von Microsoft propagierte Weg, eine .Net-Software auf einen Windows PC zu installieren, führt über ein Windows Installer Setup Package. Diese allbekannten Setup Wizards führen ein Benutzer durch den Installationsprozess. Das Entwickeln solcher Windows Installer Pakete (auch "MSI" genannt) erfordert einwenig Übung.

4 Spezifikation der Schnittstellen

4.1 Externe Schnittstellen

Im Kontextmodel (Abbildung 1: Systemübersicht (Kontextmodell)) ist gut ersichtlich mit welchen Systemen der Designer in Verbindung steht. Folgende Liste beschreibt kurz die externen Schnittstellen.

Schnittstelle	**Beschreibung / Definition**
USB	Schnittstelle zum 3D-Object-Printer. Realisiert wird sie mittels USB-Kabel. Die Übertragung der Kontroll- und Statusmeldungen wird aber seriell vorgenommen.
Win32_Printer	Die Schnittstelle "Win32_Printer" wird nicht als konkrete Klasse implementiert, wie dies bei etlichen anderen Schnittstellen der Fall ist. Für die Druckausgabe wurde eine Klasse "PrintJob" entwickelt. Die Klasse "PrintJob" implementiert die im Namespace "System.Drawing.Printing" enthaltene Klasse "PrintDocument". Diese wiederum abstrahiert Methoden zur Druckausgabe an einen gewünschten Win32_Printer. Zusammenfassend ist wichtig zu wissen, dass sämtliche Druckfunktionen (Rendering und Spooling) von einer .Net Klasse in System.Drawing.Printing erledigt wird.
Kunde / Benutzer	Eine der kritischen Schnittstellen im Projekt. Einerseits muss die Software möglichst viele Funktionen implementieren und anbieten können. Andererseits soll die Verwendung dem Benutzer nicht allzu komplex erscheinen. Diese Schnittstelle, die Benutzerschnittstelle, wurde unter den Gesichtspunkten der Usability entwickelt und getestet. Die Kriterien sind unter "Nicht-funktionale Anforderungen" im Konzeptdokument [4] zu finden.

Tabelle 5: Externe Schnittstellen

4.2 Interne Schnittstellen

Schnittstelle	**Beschreibung / Definition**
Cam	Die Cam wird mittels AForgeVideo Library angebunden.
ImageGallery	Die ImageGallery wurde als Benutzersteuerelement (engl. UserControl) erstellt. So wird ein hoher Abstraktionsgrad erreicht. Die ImageGallery ist hoch kohärent und stellt nur jene Methoden nach aussen, welche für die Steuerung der Komponente benötigt werden. ImageGallery wird im Hauptfenster des Designers instanziert und dockt am rechten Bildrand.
ComProtocol	Über diese Schnittstelle kommuniziert der Designer mit dem Microcontroller. Das ComProtocol wird unter folgendem Kapitel näher beschrieben: 3.3.3 - COM und ICOM.
ToolBox	Die ToolBox wurde als Benutzersteuerelement realisiert. Die ToolBox enthält sämtliche Grafikwerkzeuge welche zur Manipulation von Zeichnungen notwendig sind. Damit diese Werkzeuge Zugriff auf die Elemente einer

	Zeichnung erlangen, muss bei der Initialiserung der Komponente eine Referenz zur gegenwärtigen Zeichnung übergeben werden. Ferner ist zu beachten, dass auch beim Fokuswechsel (bspw. von Zeichnung 1 zu Zeichnung 2) eine neue Referenz an die ToolBox übergeben werden muss. Ansonsten würden die Werkzeuge der ToolBox die gewünschten Manipulationen auf einer falschen Zeichnung ausführen oder gar in eine NullPointerException laufen, sofern die referenzierte Zeichnung nicht mehr geöffnet ist. ToolBox wird im Hauptfenster des Designers instanziert und dockt am linken Bildrand.
Config-File	Der Designer verfügt über ein Config-File, in welchem alle Einstellungen abgelegt werden. Die Usersettings werden im Profile des Benutzers abgelegt und können von ihm auch selber konfiguriert werden. Mehr dazu im Kapitel: 3.4 - Konfigurationsdatei.
VectorShapes	VectorShapes enthält ein komplexes Vektorgrafik-Framework. Dieses stellt umfangreiche Methoden zur Manipulation von Zeichnungen bereit. Das Framework war Teil eines kompletten C# Grafikstudios. Der relevante Code wurde extrahiert und als Benutzersteuerelement bereitgestellt. Diese Komponente wird schliesslich in jedem MDI-Zeichnungsfenster als Zeichenfläche verwendet. VectorShapes enthält Events für Mauszeigerbefehle, welche beispielsweise für das Zeichnen von geometrischen Elementen oder für das Editieren von Textboxen oder Rastergrafiken verwendet werden. Insofern wird hier also eine Benutzerschnittstelle implementiert.

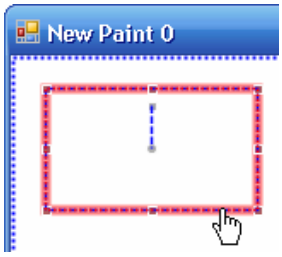
Tabelle 6: Interne Schnittstellen

5 Softwareanforderungen und Design-Entscheide

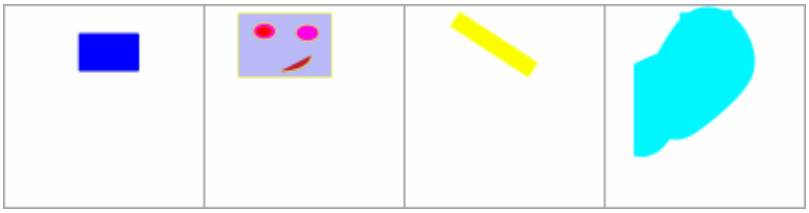
Folgendes Kapitel hält Softwareanforderungen sowie auszugsweise die wesentlichsten Design-Entscheide fest. Zum Teil wird auf bereits bestehende Dokumente verwiesen um keine Redundanz aufzubauen.

5.1 Funktionale Anforderungen

Die Anforderungen wurden bereits während dem PREN1 in der Konzeptionsphase [4] erarbeitet und definiert. Basis für diese Anforderungen ist einerseits die Aufgabenstellung [2] aus dem PREN1 & PREN2 und entsprechend der daraus mit dem Kunden erstellten Anforderungsliste [3].

Anforderung	**Lösungsstrategie**
Dateneingabe	
Dateneingabe	Zur Erstellung einer Freihandzeichnung muss dem Benutzer eine Zeichenfläche mit Grafikwerkzeugen bereitgestellt werden. Einfache geometrische Objekte wie Punkte, Linien, Rechtecke, Kreise können relativ einfach mit der .Net API "Graphics Device Interface" (GDI) realisiert werden. Nach einer kurzen Recherche auf dem Portal der .Net Community "codeproject.com" konnte eine bereits ansatzweise umgesetzte Vektorgrafik-Bibliothek gefunden werden. Diese Bibliothek bot ziemlich viele Möglichkeiten zur kreativen Gestaltung einer Zeichenoberfläche. Nachteile dieser Vektorkomponenten: Der komplexe Aufbau sowie die teilweise markanten Fehler. Das Konzept des Information Hidings wurde oft auf üble Art und Weise verletzt. Es musste also entschieden werden, ob 1. die Grafikkomponenten von Grund auf selber aufgebaut werden oder 2. eine halbfertige, fehlerhafte Komponente so angepasst wird, dass sie in der PREN Software die nötigen Aufgaben erfüllen kann Der Entscheid fiel aufgrund der überzeugenden Funktionalitäten auf die Codeproject Vektorkomponente. Ein zusätzlicher Grund für diesen Entscheid liegt im knappen Zeitplan von PREN2. Es müssen noch etliche andere Komponenten entwickelt werden, welche keineswegs mindere Priorität haben. 
Dateneingabe	Um dem Benutzer eine Bildgalerie bereitstellen zu können, bedarf es der Entwicklung einer Komponente. Hier wurden drei Varianten evaluiert, um der Anforderung gerecht zu werden: 1. Framework Komponente "ListView" mit ListItem als Bildvorschau analog der Darstellung des Windows Exporers.

2. Framework Komponente "DataGridView" mit Rows und Cols in welchen die Vorschaubilder dargestellt werden.



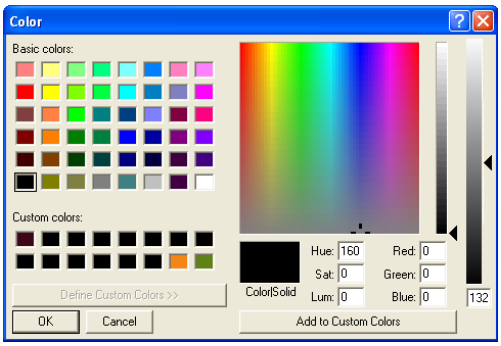
3. Eine externe Bibliothek importieren, welche die Funktionen einer ImageGallery implementiert.

Variante 3 wurde ausgeschlossen, da das .Net Framework bereits genügend Mittel bereitstellt, um die Anforderung zu realisieren. Der Entscheid zwischen Variante 1 und 2 war nicht einfach: Einerseits bot das ListView gute Drag&Drop Events, andererseits war die Darstellung im DataGridView besser als im ListView. Ausschlaggebend war die einfache Grössenveränderung der Vorschaubilder im DataGridView. Zudem bestanden mehr und mehr Laufzeitfehler in der Implementation mit der ListView Komponente, sodass diese schliesslich komplett aus der Software entfernt wurde.

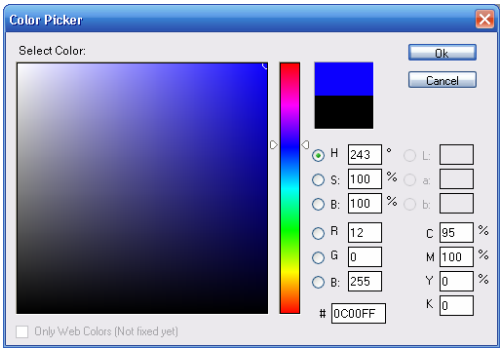
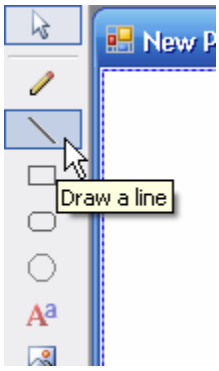
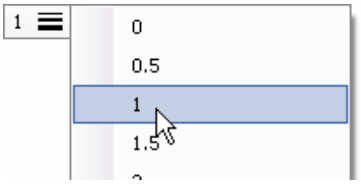
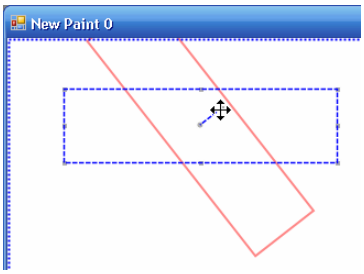
Farbauswahl

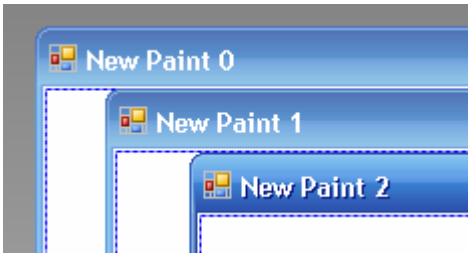
Eine Anforderung der Software bestand darin, dass der Benutzer verschiedene Malfarben wählen kann. Da bieten sich wieder mehrere Implementationsmöglichkeiten an:

1. Komponente "ColorDialog" aus dem System.Windows.Forms Namespace der .Net Framework API.

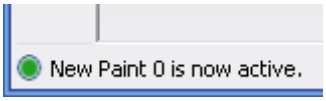


2. Eine externe Bibliothek importieren, welche die Funktionen einer ImageGallery implementiert. Zum Beispiel " Adobe Color Picker Clone", "ColorPicker.NET" und vgl.

	 Der Entscheid fiel letztlich auf den Adobe Color Picker Clone. Adobe hat in ihrer professionellen Gestaltungssoftware "Photoshop" einen sehr durchdachten Color Picket implementiert. Der "Adobe Color Picker Clone" versucht diesen Adobe Picker weitgehend zu imitieren, was ihm sehr gut gelingt. Der Standard Picker "Color Dialog" hätte den Anforderungen auch genügt – stellte aber beispielsweise keine hexadezimale Farbeingabe bereit.
Werkzeugauswahl	 Durch die Wahl der Vektorgrafik-Bibliothek war auch das Thema "Werkzeugwahl" erschlagen: Die in der Vektorgrafik-Bibliothek enthaltene Toolbox beinhaltet einfache Werkzeuge wie - Freihandstift - Linie - Rechteck - Ellipse Die Toolbox erlaubt es im Weiteren auch Bilder und Richtext-Elemente einzufügen, womit ein weiteres Problem gelöst wurde.
Werkzeugoptionen	 Die Stiftgrösse muss durch den Benutzer veränderbar sein. Diese Anforderung wurde mittels einer Methode `setPenWidth` gelöst, welche die Eigenschaft `penWidth` eines Grafikelements verändert. Dem Benutzer wird diese Methode über die Toolbox zugänglich gemacht. Über ein Dropdown-Menü kann er die Pixelbreite wählen.
Bildbearbeitung	 Die elementaren Bildbearbeitungsfunktionen werden bereits in der Vektorgrafik-Bibliothek mitgeliefert. In den Anforderungen wurden diese als "Wunschziele" aufgeführt. Effektiv implementiert wird zurzeit nur die stufenlose Rotation eines Grafikelements. Die Flip-Funktion wurde bis dato noch ausgelassen. Es wäre aber jederzeit möglich, diese Funktion als Erweiterung zu implementieren.

Zeichenfläche	Es wurde gewünscht, dass mehrere Zeichenflächen gleichzeitig offen gehalten werden können. Um dieser Anforderung gerecht zu werden, musste die Hauptanwendung ein MDI Parent Container bereitstellen. Diesem Container können dann unzählige MDI Child Container angehängt werden. 
Datenausgabe	

Speicherung von Shapes	Die Speicherung einer Zeichnung wurde über eine Objekt-Serialisierung realisiert. Ein OpenFileDialog sorgt dafür, dass der Benutzer den Speicherort im Filesystem wählen kann. An diesen Ort wird dann ein binärer Stream geschrieben.
	`if ((StreamWrite = dialog.OpenFile()) != null)`
	`{`
	`  BinaryFormatter BinaryWrite = new BinaryFormatter();`
	`  BinaryWrite.Serialize(StreamWrite, this.s);`
	`  StreamWrite.Close();`
	`}`
	Der Vorteil dieser Technik liegt in der Einfachheit des Schreibens sowie des Lesens (im Fall eines Öffnens einer Zeichnung). Zudem wird mit diesem Verfahren keine Pixelgrafik sondern eine echte Vektorgrafik gespeichert!

Speicherung für Bildgalerie	Sofern der Benutzer ein Bild in die Bildgalerie speichern möchte, kann er ein kreiertes Motiv in einen definierten Ordner ablegen. Anhand des Dateinamens, welcher der Benutzer wählt, kann das Motiv später auch über die Suchfunktion der Bildgalerie wieder gefunden werden.
Statusanzeige	Am Bildrand links-unten wird der Benutzer über den gegenwärtigen Status der Software informiert. Mit verschieden farbigen Icons wird der gegenwärtige Status untermalt. 
Drucker	Der Drucker wird von der Software automatisch als PREN 3D-Object-Printer erkannt. Die originalen HP Druckertreiber wurden so angepasst, dass die Software eindeutig erkennen kann, ob es sich bei einem angeschlossenen Drucker um ein Drittgerät oder den PREN 3D-Object-Printer handelt.
Druckvorschau	Die Druckvorschau wird über ein modales Dialogfenster gelöst. Es soll laut Anforderungsliste möglich sein, ein kreiertes Bild über das Videobild der im 3D-Object-Printer eingebauten Kamera zu legen. Das Hauptproblem dieser Aufgabe liegt darin, dass das erstellte Bild an sämtliche Weissbereichen transparent gemacht wird und es gleichzeitig über dem Echtzeitbild der Kamera darzustellen. Grundsätzlich stehen mehrere verschiedene Lösungswege zur Auswahl, von welchen jeweils ein funktioneller Prototyp erstellt wurde. - DirectShow und abgeleitete Klasse "Panel", welche in hoher Frequenz über das Webcam-Bild gelegt wird. Problem: DirectShow (Webcam-Bild) übersteuert das kreierte Motiv. Es kommt zu einem unschönen Flackern.

	- DirectShow und abgeleitete Klasse "PictureBox", bei welcher jeder weisse X- und Y-Pixel transparent gemacht wird. Problem: Dieser Lösungsweg ist viel zu zeitintensiv und deshalb nicht anwendbar.
	- AForge.Video mit Bildübersteuerung während dem "`NewFrame`"-Event der Webcam.
	
	`private void videoSourcePlayer_NewFrame(object sender, ref Bitmap image)`
	`{`
	`g.DrawImage(image, new Rectangle(0,0,image.Width,image.Height));`
	`g.Dispose();`
	`}`

Druckverfolgung	Für die Druckverfolgung kommt dasselbe Video-Framework (AForge.Video) zum Einsatz. Es wird ebenfalls ein modaler Dialog erstellt, welcher während dem Druckvorgange Informationen und natürlich das Livebild darstellt.
Kommunikation	
Schnittstelle ICOM	Die Software nutzt ein Interface "ICOM" für den Informationsaustausch mit dem 3D-Object-Printer. Der Clou an diesem Interface besteht in der sehr losen Kopplung zwischen Hauptsoftware und der Kommunikationslogik. Die Klasse "COM" ist schliesslich zuständig für die Implementation der Schnittstelle "ICOM". COM könnte jederzeit durch eine neue Version oder eine auf neuer Technologie basierenden COM-Klasse ersetzt werden.
Protokoll ComProtocol	Das Nachrichtenformat, welches zwischen Hauptsoftware und Kommunikationslogik verwendet wird, wird durch das eigenhändig definierte Protokoll "ComProtocol" vorgegeben. Das Protokoll enthält im ersten Release nur gerade mal zwei Felder: - Eine Textnachricht im String-Format - Eine Zustandsmeldung als Integer-Enumerator "MessageType" Die Kommunikation ist bidirektional und beruht auf einem Event-basierten "Publish/Subscribe" Prinzip. Ein Protokoll kann sowohl von der Hauptsoftware an die Kommunikationslogik, als auch von der Kommunikationslogik zur Hauptsoftware gesendet werden. Interessiert sich jedoch keine Hauptsoftware für Meldungen von der Kommunikationslogik, so werden diese einfach zerstört.
Heartbeat Signal	Ein Thread der Hauptsoftware kontaktiert den 3D-Object-Printer in regelmässigen Zeitabschnitten mit einem definierten Signal. Der Microcontroller quittiert diese Meldungen unverzüglich mit einer Bestätigung. Wird über einen bestimmten Zeitabschnitt vom Thread keine Bestätigung empfangen, so wechselt der Zustand des 3D-Object-Printers von "online" auf "offline". Der Benutzer wird in der Statusanzeige über den gegenwärtigen Zustand informiert.
Automatische Geräteerkennung	Beim Initialisieren der Software wird auf den COM Ports nach dem 3D-Object-Printer gesucht. Wird ein Gerät gefunden, so wird dessen COM Port Nummer automatisch in die Konfiguration geschrieben. Der dafür benötigte Programmcode wurde in den Kommunikationskomponenten COM/ICOM umgesetzt.

Tabelle 7: Funktonale Anforderungen: Lösungsstrategien

5.2 Nicht-funktionale Anforderungen

Analog Funktionale Anforderungen (Siehe 5.1 Funktionale Anforderungen).

Anforderung	Lösungsstrategie
Usability der Software	
GUI ist intuitiv bedienbar	Die Software (Designer) wird mittels .NET programmiert, was bereits eine grosse optische Anlehnung an die weit gängige Microsoft-Oberfläche bringt. Durch die Verwendung ähnlicher, resp. genau gleichen Objekten wird eine Oberfläche geschaffen, welche dem Benutzer auf den ersten Blick vertraut erscheint. Neben der Statusliste (Datei, Bearbeiten usw.) werden auch sämtliche eingesetzte Grafiktools mit den üblichen Grafiken bildlich markiert.
Erfolgreiches Benutzen	Durch die saubere Definition von UseCases ist gewährleistet, dass die vom Benutzer (Aufgabenstellung) verlangten Funktionen realisiert werden. Die iterative Vorgehensweise nach HTAgil schreibt dem Testen eine wichtige Rolle zu. Der Testplan definiert verschiedene Testfälle, basierend auf den UseCases, und weitere kleine Test. Das Testen und festhalten der Resultate in einem entsprechenden Protokoll garantiert die erfolgreiche Anwendung der Funktionen.
Logischer Aufbau	Der Aufwand der Software basiert auf einzelnen Komponenten, was das gesamte Erscheinungsbild sehr flexibel erschienen lässt. Jede Komponente (z.B. ImageGallery oder Toolbox) können individuell ein- und ausgeblendet werden. Die Benutzersteuerung wurde absichtlich so gestaltet, wie sie in den meisten Windows Programmen vorzufinden ist.
Geringe Fehlerrate	Da die Fehlerfreiheit in keiner Software garantiert werden kann, wurde sichergestellt, dass sämtliche Methoden durch try-catch Fehlerbehandlungen abgesichert sind. Werden Fehler entdeckt, sollen diese vom Tester reproduziert werden können, sodass die bestimmte fehlerbehaftete Anwendungssituation gefunden werden kann.
Effiziente Benutzbarkeit	Die Software soll so erstellt werden, dass dem Benutzer eine möglichst effiziente Verwendung der Software möglich ist.
Benutzung der Software	
Look and Feel	Das GUI wurde mit vielen Grafik-Icons ausgestattet. Die Icons basieren auf den standardmässigen Microsoft Icons, sodass der Benutzer bereits anhand des Icons erraten kann, welche Funktion hinter einer Schaltfläche steckt.
Erwartungskonformität	Die Software soll bei der Verwendung von üblichen Funktionen stark auf die aktuellen Standards setzten und dabei die Komplexität bei der Benutzung minimieren.
Cursor	Der Cursor zeigt bei blockierenden Aufrufen das Sanduhr-Icon an. Der Benutzer weiss damit, dass er im Moment keine weiteren Aufrufe machen kann. Wo möglich wurden Threads verwendet, um die Blockierung des UI Threads zu verhindern.

Tabelle 8: Nicht-funktonale Anforderungen: Lösungsstrategien

6 Environment-Anforderungen

Die Anforderungen an die Hardware sind minimal zu halten. Diese Systemvoraussetzungen wurden bereits in der Konzeptphase [4] genau spezifiziert. Hier noch einmal kurz die wesentlichsten Leistungseigenschaften, welche ein Zielsystem vorweisen muss:

	Empfohlener Mindestwert	Beschreibung
Anforderungen an die Hardware		
Festplattenspeicher	20MB	Bei wachsender Bildgalerie wird entsprechend mehr Festplattenspeicher benötigt.
Arbeitsspeicher (RAM)	128MB	
Prozessorleistung	1GHz	Geringere Werte möglich. Die Qualität des Livestreams und der Druckvorschau könnten jedoch darunter leiden.
Peripherie	1 USB Anschluss	
Anforderungen an die Software		
Microsoft .Net Framework	2.0.50727	Sehr umfangreiche Runtime Environment mit einer umfangreichen Sammlung von Klassenbibilotheken, API und Services.
Windows Installer	3.0	Runtime Environment für Installationsroutinen unter Microsoft Windows Betriebssystemen. Neuere Versionen werden problemlos unterstützt.
Betriebssystem	Windows XP SP2	Applikation soll auch auf aktuelleren Betriebssystemen von Microsoft laufen, wie z.B. Windows 7.

Tabelle 9: Environment-Anforderungen