Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Lucas Nandico

MASTER THESIS

Entwicklung eines praktischen Ende-zu-Ende-Systems zur 3D-Unterstützung in Service-Wartungsdokumentation im Bereich der Fahrzeugdiagnose

Abgabe am 02.06.2025

Betreuer: Prof. Dr. Dirk Riehle, M. B. A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____

Holzkirchen, 02.06.2025

License

_____

Holzkirchen, 02.06.2025

Abstract

The maintenance and diagnosis of modern vehicles is made considerably more difficult by the increasing system complexity and variety of variants. Technical documentation and software-supported workshop testers are used to support workshop personnel. Three-dimensional representations can make an important contribution here by improving the understanding of spatial relationships and facilitating the localisation of components.

As part of this work, an integrated system for integrating interactive 3D visualisations into existing vehicle documentation and workshop applications was developed. It comprises three main components: a user-guided, modular data preparation pipeline, a versatile, configurable 3D viewer component and a documentation viewer for displaying PDF documents with embedded 3D scenes. The CAD design data is processed automatically and linked to existing documents via a stable reference mechanism.

The system nearly fulfils all the defined requirements, and the evaluation shows a high level of practical suitability in terms of visualisation quality and flexibility. There is potential for optimisation above all in user guidance and support. Overall, the solution offers a reliable basis for productive use in the environment of vehicle component manufacturers and workshops.

Zusammenfassung

Die Wartung und Diagnose moderner Fahrzeuge wird durch die zunehmende Systemkomplexität und Variantenvielfalt deutlich erschwert. Um Werkstattpersonal zu unterstützen, werden technische Dokumentationen und softwaregestützte Werkstatttester eingesetzt. Dreidimensionale Darstellungen können hier einen wichtigen Beitrag leisten, indem sie das Verständnis räumlicher Zusammenhänge verbessern und die Lokalisierung von Komponenten erleichtern.

Im Rahmen dieser Arbeit wurde ein durchgängiges System zur Integration interaktiver 3D-Visualisierungen in bestehende Fahrzeugdokumentationen und Werkstattanwendungen entwickelt. Es umfasst drei Hauptkomponenten: eine nutzergeführte, modular aufgebaute Datenaufbereitungspipeline, eine vielseitig konfigurierbare 3D-Viewer-Komponente, sowie einen Dokumentationsviewer zur Anzeige von PDF-Dokumenten mit eingebetteten 3D-Szenen. Die CAD-Konstruktionsdaten werden automatisiert aufbereitet und über einen stabilen Referenzmechanismus mit bestehenden Dokumenten verknüpft.

Das System erfüllt nahezu alle der definierten Anforderungen vollständig und zeigt in der Evaluation eine hohe Praxistauglichkeit hinsichtlich Visualisierungsqualität und Flexibilität. Optimierungspotenzial besteht vor allem in der Benutzerführung und -unterstützung. Insgesamt bietet die Lösung eine belastbare Grundlage für den produktiven Einsatz im Umfeld von Fahrzeugkomponentenherstellern und Werkstätten.

Inhalt

1 Einleitung

Fahrzeuge sind hochkomplexe Systeme aus Hardware und Software. Die Vielfalt der Varianten sowie die Vielzahl verschiedener kommunizierender Teilsysteme erschweren die Arbeit am Fahrzeug (Rumpe & Schiffers, 2006, S. 1–2). Damit ergeben sich auch erhöhte Anforderungen an die Wartung und den Service für Fahrzeuge, die in der Werkstattpraxis bewältigt werden müssen.

Zur Unterstützung für anfallende Aufgaben dienen technische Dokumentationen und spezialisierte Softwarelösungen. Fahrzeughersteller und Fahrzeugkomponentenhersteller stellen diese den Werkstätten zur Verfügung. Die Dokumentation umfasst typischerweise Funktionsbeschreibungen, strukturierte Komponentenübersichten sowie Handlungsanweisungen zur Wartung und Montage. Insgesamt müssen alle Aufgaben der Instandhaltung und der Instandsetzung unterstützt werden. Technische Dokumentation wird weitverbreitet in Form von PDF-Dokumenten bereitgestellt (Engelke et al., 2015, S. 1, Gattullo et al., 2019, S. 15), die oft durch schematische oder fotografische Darstellungen ergänzt werden.

Unabhängig von dem gewählten Dokumentformat nutzen derartige statische, papierähnliche Dokumentationen grafische Visualisierungen der Fahrzeugkomponenten und der Prozessschritte, um das Verständnis zu erleichtern. Diese sind allerdings in einem statischen 2D-Medium, wie PDF-Dokumenten, nur limitiert von Nutzen. Interaktive, dreidimensionale Darstellungen ermöglichen im Vergleich zu 2D-Bildern ein besseres räumliches Verständnis. Werden solche Darstellungen direkt in bestehende Dokumente integriert, sinkt die Einstiegshürde für die Verwendung für das Werkstattpersonal, da vertraute Arbeitsabläufe beibehalten werden können.

Neben der reinen Dokumentationsunterstützung kann 3D-Visualisierung auch zur Verbesserung softwaregestützter Werkstattprozesse beitragen. Besonders der Bereich der Diagnose, bei der mithilfe von automatisierten Prozeduren mit dem Fahrzeug kommuniziert wird, wird unterstützt durch Software basierend auf Standards wie OTX und ODX (Reif, 2014, S. 417). Sogenannte Werkstatttester-Programme werden von den Fahrzeugherstellern konfiguriert und den Werkstätten auf klassischen Desktop-PCs oder Touch-basierten Geräte bereitgestellt. Das Werkstatttester-Programm liest dann im Betrieb unter anderem die gemeldeten Fehler einzelner elektronischer Steuerelemente im Fahrzeug aus und zeigt sie dem Werkstatt-Mitarbeiter an. Eine 3D-Visualisierung der fehlerhaften Komponenten selbst, aber auch im Kontext des Fahrzeugs, erlaubt eine einfachere Lokalisierung und erleichtert damit den komplexen Diagnose-Prozess.

Für die Entwicklung von dreidimensionalen Darstellungen werden 3D-Daten der Fahrzeuge benötigt. In der Industrie ist die Entwicklung mithilfe von Computer-Aided-Design-(CAD)-Programmen Standard. Dabei werden mithilfe dieser Programme die Konstruktionen der Fahrzeugkomponenten und -systeme für die Fertigung erstellt und die Daten stehen grundsätzlich zur Verfügung (Mohr et al., 2015, S. 1–2). Fokus liegt bei der Entwicklung dieser 3D-Modelle allerdings nicht darauf, dass sie für eine Visualisierung geeignet sind, sondern viel mehr, dass sie für den anschließenden Fertigungsprozess verwendet werden können. Damit die 3D-Daten stattdessen für die Darstellung in der Dokumentation verwendet werden können, müssen sie gezielt transformiert werden. Das betrifft verschiedene Aspekte wie Materialanpassungen, Geometrievereinfachung und anwendungsfallspezifische Umstrukturierungen der Bestandteile der 3D-Daten.

Ziel dieser Arbeit ist die Entwicklung eines Ende-zu-Ende-Systems von Konstruktionsdaten bis hin zur Integration von interaktiven 3D-Visualisierungen auf Basis dieser Konstruktionsdaten in die bestehende Fahrzeugdokumentation und Software. Das System besteht aus drei Hauptkomponenten:

1. 3D-Viewer-Komponente: Eine Komponente zur interaktiven Darstellung aufbereiteter 3D-Daten, die in bestehende Diagnosesoftware integriert werden kann.
2. Dokumentationsviewer: Eine Anwendung zur Anzeige der Fahrzeugdokumentation ergänzt mit 3D-Visualisierungen.
3. Datenaufbereitungspipeline: Eine weitgehend automatisierte Prozesskette zur Transformation der Konstruktionsdaten für die Verwendung in der 3D-Viewer-Komponente und dem Dokumentationsviewer.

Eine grundsätzliche Anforderung für das Ende-zu-Ende-System ist über die Funktionalität hinaus, dass das System praktisch möglichst einfach nutzbar ist und die Integration des Systems möglichst wenig Modifikationen bestehender Prozesse erfordert.

Teil der Arbeit ist die Bereitstellung geeigneter 3D-Visualisierungsmechanismen für typische Werkstattszenarien – sowohl zur Erweiterung der Dokumentation als auch zur direkten Unterstützung im Werkstatttester. Zu diesem Zweck baut insbesondere die 3D-Viewer-Komponente auf der Arbeit „Entwicklung eines 3D-Viewers zur dynamischen Visualisierung von Fahrzeugkomponenten im Bereich der Fahrzeugdiagnose“ (Nandico, 2023) auf. Im Gegensatz zu dem dort entwickelten 3D-Viewer wird mit dieser Arbeit ein allgemeingültigerer einsetzbarer 3D-Viewer entwickelt, der sich nicht nur auf die Verwendung im Werkstatttester beschränkt, sondern mit seinem Funktionsumfang die Anforderungen als Bestandteil der Dokumentation erfüllt. Der erweiterte 3D-Viewer bietet insbesondere umfangreiche Konfigurationsmöglichkeiten und praxisnahe Funktionen, wie die automatisierte Generation von Explosionsansichten.

Die Arbeit gliedert sich in die folgenden Bereiche:

1. Das Kapitel *Verwandte Arbeiten und Grundlagen der Dokumentation in der Fahrzeugdiagnose* stellt verwandte Arbeiten vor, auf deren Erkenntnisse das Ende-zu-Ende-System aufbaut, und gibt eine kurze Einführung in die Verwendung von Dokumentation in der Fahrzeugdiagnose.

2. In *Anforderungen* werden im ersten Schritt die Anwendungsszenarien bestimmt. Ausgehend von diesen werden im Anschluss die konkreten Anforderungen definiert für die verschiedenen Teilbereiche des Ende-zu-Ende-Systems.
3. Die *Architektur* erläutert den Kontext des Systems, den Aufbau und die grundlegenden Schnittstellen der Komponenten.
4. Das Kapitel zu *Design und Implementierung* konzentriert sich auf die tatsächliche Umsetzung mit dem Framework Qt und erläutert die Kernpunkte der Implementierung der Schritte der Datenaufbereitungspipeline und der Visualisierungsmechanismen. Es endet mit einem beispielhaften Gesamtdurchlauf der Datenaufbereitung in 5.6.
5. Teil der *Evaluation* ist die Überprüfung der Anforderungserfüllung und eine Betrachtung der Praxistauglichkeit des entwickelten Systems.

Die Anforderungen und Vorgehensweisen für die Bereitstellung der Dokumentation orientieren sich an einer produktiv eingesetzten Diagnoselösung eines mittelständischen Unternehmens aus dem Bereich der Fahrzeugdiagnose. Die für die Visualisierungen verwendeten 3D-Daten wurden von diesem Unternehmen bereitgestellt.

2 Verwandte Arbeiten und Grundlagen der Dokumentation in der Fahrzeugdiagnose

Das entwickelte Ende-zu-Ende-System greift auf Konzepte und Erkenntnisse aus mehreren Forschungsgebieten zurück. Im Kapitel 2.1 werden die Arbeiten vorgestellt, die im Wesentlichen die Basis für diese Arbeit darstellen. In Kapitel 2.2 folgt eine Einführung in die strukturellen und organisatorischen Grundlagen der Fahrzeugdiagnose und der Bereitstellung technischer Dokumentation im Werkstattkontext. Diese Einordnung schafft die notwendige Grundlage für das Verständnis der Zielstellung und Anforderungen des entwickelten Systems.

2.1 Verwandte Arbeiten

Konzeptionell weist diese Arbeit Ähnlichkeiten zum von Mohr et al. (2015) entwickelten System auf. Mohr et al. (2015) beschäftigen sich in ihrer Arbeit ebenfalls damit, bestehende technische Dokumentation mit neueren Möglichkeiten zu integrieren: Konkret entwickelten sie ein System, um Darstellungen in technischen Dokumenten für Augmented Reality zu transformieren. Zwar wird in beiden Ansätzen die bereits existierende Dokumentation verwendet, jedoch liegt der Fokus in dieser Arbeit auf der Integration vorhandener 3D-Daten in bestehende Dokumentation – nicht umgekehrt. Das ist grundsätzlich möglich, da die Fahrzeugkomponentenhersteller sowohl die Dokumentation als auch die 3D-Daten besitzen.

Ein zentrales Element des entwickelten Systems ist die 3D-Viewer-Komponente, deren Grundlage auf der Arbeit von Nandico (2023) basiert. Aufbauend auf den Anwendungsszenarien von Nandico (2023) und unter Einbezug der weitergehenden Anforderungen für die Service- und Wartungsdokumentation ergeben sich auch für den 3D-Viewer zusätzliche Anforderungen, die in dem Kapitel 3 erläutert werden. Die Realisierung wesentlicher Zusatzfunktionen des 3D-Viewers auf Basis der Vorarbeit von Nandico (2023) wird genauer im Kapitel 5.5 erläutert. Sie betreffen Darstellungsmechanismen, wie

1. Bestimmung von Positionen für Beschriftungen nach Ali et al. (2005),
2. verbesserte Transparenzberechnungen nach Hummel et al. (2010),
3. vereinfachte Darstellung mit Konturenzeichnungen nach Saito und Takahashi (1990) und
4. automatische Generation von Explosionsdarstellungen nach Li et al. (2008).

Die Aufbereitung der CAD-Konstruktionsdaten und die Konfiguration der Aufbereitung umfassen Bereiche der 3D-Daten-Optimierung mit

1. Vereinfachung der Geometrie nach primär Garland und Heckbert (1997) und
2. Anpassung der visuellen Eigenschaften der Objekte für zielgerichtete Darstellungen mit ausreichendem Realismus nach Burley (2012).

Beide Bereiche stellen gut erforschte Gebiete dar.

2.2 Fahrzeugdiagnose und Dokumentationsbereitstellung

Die Diagnose von Fahrzeugen ist ein integraler Bestandteil der Arbeit in der Werkstatt (Rauner et al., 2002, S. 49) und wird mit Dokumentation und Software gestützt. Die Fahrzeugdiagnose definiert sich nach Reif (2014, S. 417) wie folgt: „Aus konkreten und diffusen Symptomen, die der Fahrer schildert, wird im Service unter Zuhilfenahme der Diagnosesysteme ein exaktes Fehlerbild erstellt und es werden geeignete Reparaturmaßnahmen eingeleitet.“ Diese Diagnosesysteme müssen die Informationen enthalten, welche Komponenten im Fahrzeug enthalten sind, um zu ermitteln, welche spezifischen Prozeduren zur Fehlerfindung eingesetzt werden können. Die Diagnose ist einerseits gesetzlich kontrolliert und standardisiert, sodass gewisse Abfragen für alle Fahrzeuge möglich sein müssen und andererseits bieten Hersteller selbst aus Eigeninteresse spezialisierte Analysesysteme an, da sie ebenso an der Fehleranalyse interessiert sind (Reif, 2014, S. 417).

Insbesondere die elektronischen Steuergeräte im Fahrzeug ermöglichen, dass für sie automatisierte Prozeduren erstellt werden, die dann mit den Steuergeräten kommunizieren. Die Konfiguration dieser Prozeduren ist grundsätzlich spezifisch für Steuergeräte. Eine Diagnoseprozedur für ein Motorsteuergerät ergibt keinen Sinn für die Diagnose des Antiblockiersystems. Fahrzeugkomponentenhersteller definieren deswegen einerseits die Diagnoseprozeduren und andererseits legen sie fest, welche Steuergeräte für welche Prozeduren geeignet sind.

Im Rahmen der exemplarischen Diagnoselösung werden drei Anwendungen unterschieden: eine Entwicklungs- und Konfigurationsanwendung, eine Serverapplikation und ein Werkstatttester. Die Entwicklungsumgebung dient der Erstellung der steuergeräte- und problemspezifischer Diagnoseprozeduren mit Benutzeroberfläche, Diagnosedaten, zugrunde liegende Diagnosedienste und Diagnoseabläufe. Über die Serverapplikation als Benutzer- und Rechteadministrationssystem werden die entstandenen Pakete an die Werkstätten verteilt. Das Werkstatttester-Programm verarbeitet die Diagnoseprojekte und stellt den Werkstattmitarbeitern die relevanten Prozeduren zur Verfügung.

Die Abbildung 1 zeigt verallgemeinert den Bereitstellungsprozess. Der Fahrzeugkomponentenhersteller definiert das Diagnoseprojekt mit den Diagnosefunktionen und der Fahrzeugmodellhierarchie, um eine Referenzierung zu ermöglichen. Ebenso wie Diagnosefunktionen können Dokumente bestimmten Fahrzeugmodellen zugewiesen werden, die dann in der Werkstatt bei Bedarf zur Verfügung stehen. An dieser Struktur orientiert sich im Kapitel 4.1 die Integration in den bestehenden Datenbereitstellungsprozess.

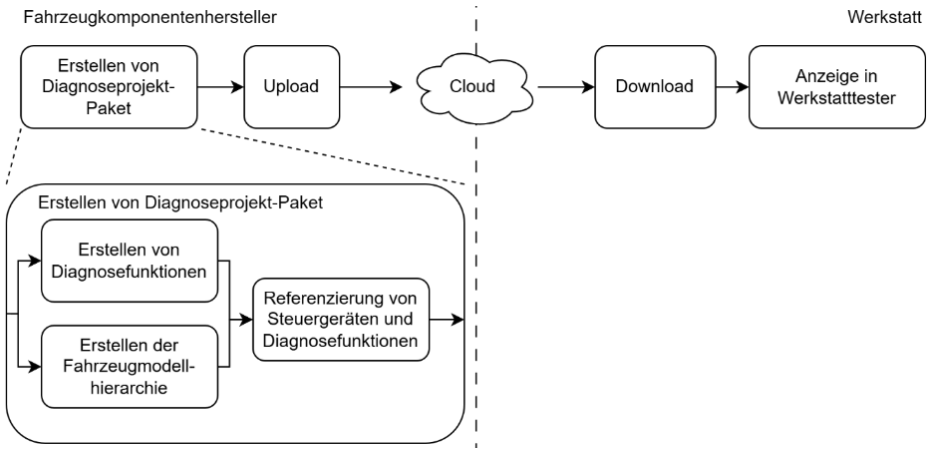


Abbildung 1: Bereitstellung eines Diagnoseprojekt-Pakets für Werkstatttester (Eigene Darstellung)

3 Anforderungen

Die Anforderungen an die Datenaufbereitungspipeline, den Dokumentationsviewer und an den zugehörigen Viewer ergeben sich aus den Anwendungsszenarien der Werkstätten und der Fahrzeugkomponentenhersteller. In 3.1 werden die spezifischen Anwendungsszenarien erläutert, die die Basis für die abgeleiteten Anforderungen in 3.2 darstellen.

3.1 Anwendungsszenarien

Für eine sinnvolle Analyse der Anforderungen müssen zusätzlich zu den Szenarien aus der Werkstatt (3.1.1) auch die Szenarien aus den bereits existierenden Datenbereitstellungsverfahren und Dokumentationsprozessen der Komponentenhersteller betrachtet werden (3.1.2) um ein praktisch nutzbares und integrierbares System zu entwickeln.

3.1.1 Szenarien aus der Werkstattpraxis

Die Tätigkeiten eines Werkstattmitarbeiters umfassen eine Vielzahl verschiedener Aufgaben. Eine Aufteilung ist nach Rauner et al. (2002, S. 47) in die folgenden Bereiche möglich:

1. Standardservice
2. Diagnoseaufgaben
3. Reparaturaufgaben
4. Zusatzinstallationen und Konfigurationsaufgaben
5. Karosserieinstandsetzung / Unfallschadensbehebung

Nach Nandico (2023, S. 6–9) existieren die folgenden Anwendungsszenarien für eine Nutzung von 3D-Visualisierungen integriert in den Werkstatttester:

1. „Als Werkstattmitarbeiter möchte ich eine schnelle Übersicht über die Störungen im Fahrzeug erhalten, damit ich den Zustand besser einschätzen und meine Arbeit besser angepasst planen kann.“
2. „Als Werkstattmitarbeiter möchte ich wissen, wo sich die fehlerhafte Komponente im Fahrzeug befindet, damit ich eine zielgerichtete Reparatur durchführen kann.“
3. „Als Werkstattmitarbeiter möchte ich wissen, mit welchen Komponenten ein Fehler zusammenhängt, damit ich eine besser informierte Reparatur beziehungsweise Diagnose durchführen kann.“
4. „Als Werkstattmitarbeiter möchte ich schnell zu bestimmten Steuergeräten navigieren, damit ich spezifische Diagnoseabläufe für diese Steuergeräte durchführen kann.“

Die Analyse der obigen Aufgabenbereiche und der Anwendungsszenarien für 3D-Darstellungen im Werkstatttester nach Nandico (2023, S. 6–9) ergibt die in der folgenden Tabelle 1 extrahierten relevanten Szenarien für die 3D-Unterstützung in der Werkstatt:

ID	Szenario
SC_01_COMP_LOC	Als Werkstattmitarbeiter möchte ich die Komponenten des Fahrzeugs schnell lokalisieren können, um 1. die auszutauschenden Verschleißteile zu finden, 2. Prüforte für die Inspektion und Diagnose zu identifizieren (Anschlussorte für externe Diagnosegeräte, Aktoren etc.) 3. die Montage / Demontage von Teilen durchzuführen.
SC_02_GROUPING	Als Werkstattmitarbeiter möchte ich einen Überblick über die Teile einer (logischen und/oder physischen) Baugruppe des Fahrzeugs erhalten, um den Aufbau von Komponenten oder Zusammenhänge zwischen Komponenten besser zu verstehen.
SC_03_ANNOTATION	Als Werkstattmitarbeiter möchte ich die Informationen erhalten, die für eine Montage / Demontage von Teilen oder anderer fest definierter Abläufe notwendig sind.
SC_04_INTERF_OUT	Als Werkstattmitarbeiter möchte ich über die 3D-Ansicht als Teil des Werkstatttesters mit anderen Programmbereichen kommunizieren und ggf. zu ihnen navigieren, um mit einer spezifischeren Diagnose fortzufahren.
SC_05_ INTERF_IN	Als Werkstattmitarbeiter möchte ich über die 3D-Ansicht als Teil des Werkstatttesters eine Übersicht über aktuelle Störungen im Fahrzeug erhalten.

Tabelle 1: Szenarien aus der Werkstatt (Eigene Darstellung auf Basis von Nandico (2023, S. 6–9) und Rauner et al. (2002, S. 47–51))

In den folgenden Kapiteln werden die einzelnen Szenarien und ihre Relevanz erläutert.

3.1.1.1 Komponenten-Lokalisierung (SC_01_COMP_LOC)

Das Lokalisieren von Komponenten ist eine Kernaufgabe der Dokumentation. Jede Aufgabe am Fahrzeug, die durch den Werkstattmitarbeiter durchgeführt wird, bedeutet ein Lokalisieren der relevanten Bereiche. Damit kann jegliche Form der Reparatur von einer erleichterten Lokalisierung von Komponenten profitieren. Der Standardservice umfasst nach Rauner et al. (2002, S. 48) mit der Wartung und der Inspektion im Wesentlichen die Durchführung vordefinierter Abläufe am Fahrzeug und Überprüfung der Funktionsfähigkeit ausgewählter Systeme. Das beinhaltet ebenso ein Lokalisieren von den zu inspizierenden Elementen. Die Diagnose benötigt zusätzlich zu den genannten Anwendungen auch ein Lokalisieren von Anschlussorten für externe Diagnosegeräte: sogenannte Prüfpunkte. Für diverse Tätigkeiten ist es notwendig, andere Teile des Fahrzeugs zu demontieren. Eine präzise Kenntnis darüber, welche Teile entfernt werden müssen und wo sich insbesondere ihre Befestigungen befinden, ist von Vorteil.

Damit ergibt sich das folgende Szenario:

> „Als Werkstattmitarbeiter möchte ich die Komponenten des Fahrzeugs schnell lokalisieren können, um 1. die auszutauschenden Verschleißteile zu finden, 2. Prüforte für die Inspektion und Diagnose zu identifizieren (Anschlussorte für externe Diagnosegeräte, Aktoren etc.) 3. die Montage / Demontage von Teilen durchzuführen“

3.1.1.2 Logische und/oder physische Gruppierung (SC_02_GROUPING)

Eine Gruppierung von Komponenten ist nötig, wenn ein Fahrzeugteil aus der Perspektive der Diagnose in der Realität mehreren Objekten entspricht. Beispielsweise besteht die Fahrzeugkomponente *Sicherungskasten* wieder aus kleineren Teilelementen. Für das Lokalisieren des Sicherungskastens im größeren Fahrzeugkontext müssen alle Teilelemente als eine zusammenhängende, gruppierte Komponente behandelt werden.

Bei der Aggregate-Reparatur als Teilbereich ist es nötig, eine „komplette Instandsetzung einer Baugruppe wie z. B. des Getriebes, des Motors oder auch von Teilen solcher Baugruppen, wie z. B. eine Zylinderkopfüberholung“ (Rauner et al., 2002, S. 49) durchzuführen. Damit ist es auch notwendig, dass diese physischen Baugruppen in der Diagnose als Gruppen verfügbar und adressierbar sind.

Eine Gruppierung kann allerdings auch auf Basis anderer Kriterien erfolgen: So gibt das 1. Szenario des Werkstatttesters vor, dass eine Übersicht der Störungen erwünscht ist. Eine *Störung* kann nur für Komponenten des Fahrzeugs bestimmt werden, die in der Lage sind, selbst diagnostiziert einen Fehler zu identifizieren. Das betrifft damit eine Teilgruppe der Steuergeräte des Fahrzeugs, die gruppiert angezeigt werden müssen.

Das 3. Szenario für die Anwendung von 3D-Visualisierungen im Werkstatttester wünscht eine Übersicht von fehlerbezogenen Zusammenhängen, was ebenso, auf die einfachste Darstellung reduziert, einer simplen Gruppierung entspricht.

Damit ergibt sich das folgende Szenario:

> „Als Werkstattmitarbeiter möchte ich einen Überblick über die Teile einer (logischen und/oder physischen) Baugruppe des Fahrzeugs erhalten, um den Aufbau von Komponenten oder Zusammenhänge zwischen Komponenten besser zu verstehen.“

3.1.1.3 Ablaufbezogene Informationsergänzung (SC_03_ANNOTATION)

Eine bloße Darstellung der 3D-Daten des Fahrzeugs ist in bestimmten Fällen unzureichend. Beispielsweise wird für Schrauben mithilfe einer Drehmoment-Angabe festgelegt, wie stark diese verschraubt werden sollen. Auch gibt es bestimmte Komponenten, wie Dichtungsringe, die in bestimmten Intervallen ausgetauscht werden müssen. Das kann Teil einer Reparatur sein, aber auch Bestandteil des Standardservices. Auch ist es möglich, dass bei einer Demontage eine bestimmte Reihenfolge bei den zu entfernenden Schrauben gewünscht ist, sodass eine Ergänzung der Darstellung mit einer Nummerierung naheliegt. Es gibt damit gewisse Abläufe der Reparatur, für die Zusatzinformationen in den 3D-Darstellungen von Vorteil sind.

Bezogen auf die Anwendungsfälle der 3D-Darstellung als Teil des Werkstatttesters ist eine Ergänzung der 3D-Darstellung nur im 1. Szenario der Störungsübersicht von möglichem Vorteil: hier kann der konkrete Fehler zusätzlich zu der fehlerhaften Komponente dargestellt werden.

Damit ergibt sich das folgende Szenario:

> „Als Werkstattmitarbeiter möchte ich die Informationen erhalten, die für eine Montage / Demontage von Teilen oder anderer fest definierter Abläufe notwendig sind."

3.1.1.4 3D-Ansicht zur Programmnavigation (SC_04_INTERF_OUT)

Das 4. Szenario für die 3D-Darstellung als Teil des Werkstatttesters gibt das Bedürfnis für eine „schnelle" Navigation zu den Steuergeräten an. Das ist insbesondere im Fall von Steuergeräten mit aktuellem Fehler von Interesse, um dann Steuergerät-spezifische Funktionen im Werkstatttester aufzurufen. Gleichzeitig kann hier auch der räumliche Zusammenhang visualisiert werden, wenn etwa ein örtlich begrenzter Schaden vorliegt, um die Ausmaße und eventuelle Einflüsse auf naheliegende Steuergeräte zu prüfen. Verallgemeinert ist es von Vorteil, wenn die Auswahl der Werkstattmitarbeiter für die Kommunikation mit dem restlichen Werkstatttester-Programm genutzt werden kann.

Damit ergibt sich das folgende Szenario:

> „Als Werkstattmitarbeiter möchte ich über die 3D-Ansicht als Teil des Werkstatttesters mit anderen Programmbereichen kommunizieren und ggf. zu ihnen navigieren, um mit einer spezifischeren Diagnose fortzufahren."

3.1.1.5 Laufzeitabhängige Darstellung (SC_05_ INTERF_IN)

Gemäß dem 1. Szenario für die 3D-Darstellung als Teil des Werkstatttesters ist eine Übersicht über die aktuellen Störungen gewünscht. Ein wichtiger Aspekt dieses Szenarios ist, dass diese Störungen sich auf die aktuellen, zur Laufzeit ausgelesenen Daten beziehen und damit nicht vorher vom Komponentenhersteller definiert werden (im Gegensatz zu den ablaufbezogenen Informationen in 3.1.1.3). Um diese Fälle zu berücksichtigen, wird das Szenario mit aufgenommen.

Damit ergibt sich das folgende Szenario:

> „Als Werkstattmitarbeiter möchte ich über die 3D-Ansicht eine Übersicht über aktuelle Störungen im Fahrzeug erhalten."

3.1.2 Szenarien aus Sicht der Komponentenhersteller

Um in der Werkstatt grundsätzlich fahrzeugbezogene Daten zu visualisieren, müssen diese durch die Komponentenhersteller bereitgestellt werden. Wie in 2.2 beschrieben, kann das im Rahmen der Werkstatttester über die bereits existierenden Bereitstellungsprozesse der Diagnosepakete erfolgen. Von Interesse für die Entwicklung der Anforderungen sind im Wesentlichen die folgenden zwei Szenarien in Tabelle 2:

ID	Szenario
SC_06_LAW_MIN	Als Komponentenhersteller muss ich die gesetzlich vorgeschriebenen Daten zur Reparierbarkeit bereitstellen.
SC_07_DOCU	Als Komponentenhersteller möchte ich mit möglichst wenig Aufwand reproduzierbar zielführende Dokumentation zur Diagnose und Wartung bereitstellen.

Tabelle 2: Szenarien der Komponentenhersteller (Eigene Darstellung)

Nach Reif (2020, S. 458) sind Fahrzeughersteller seit 2009 bereits als Teil der Emissionsgesetzgebung dazu verpflichtet, „Diagnose- als auch Service-, Reparatur- und Wartungsdaten bereitzustellen“. Damit müssen Fahrzeughersteller eine gewisse Basis-Dokumentation bieten. Auf der anderen Seite ist die Dokumentation zur Diagnose und Wartung als Teil des Produktangebots auch ein Verkaufsargument: einfachere und damit billigere Diagnose und Wartung führt zu zufriedeneren Kunden. Fahrzeugkomponentenhersteller sind dementsprechend auch aus Eigeninteresse an besseren Fehleranalysen interessiert (Reif, 2014, S. 417). Daraus folgen die zwei in Tabelle 2 genannten Szenarien.

3.2 Abgeleitete Anforderungen

Aus den Anwendungsszenarien werden die konkreten funktionalen und nicht-funktionalen Anforderungen abgeleitet. Es wird grundsätzlich unterschieden zwischen den Anforderungen an den 3D-Viewer in 3.2.1, den Dokumentationsviewer in 3.2.2 und an die Datenaufbereitungspipeline in 3.2.3. Die Anforderungen für den 3D-Viewer werden zuerst aufgestellt, da daraus Anforderungen für die Pipeline folgen.

3.2.1 Anforderungen an den 3D-Viewer

Für den 3D-Viewer ergeben sich die in der folgenden Tabelle 3 gelisteten, funktionalen Anforderungen:

ID	Anforderung
REQ_F_V_01_SHOW	Anzeige von interaktiven 3D-Darstellungen auf Basis passend aufbereiteter Konstruktionsdaten
REQ_F_V_02_STRUC	Unterstützung einer logischen und physischen (Bau-)Gruppenstruktur zur Strukturierung der Komponenten
REQ_F_V_03_TRAV	Bereitstellung von UI- und programmatischen Mechanismen zur Traversierung der Gruppenstruktur
REQ_F_V_04_HIGHL	Konfigurierbare Hervorhebung (z. B. Farbe, Transparenz, Fokus) von Komponenten
REQ_F_V_05_ANNO	Anzeige von teilebezogenen Zusatzinformationen (z. B. Drehmoment, Einbauhinweise, Diagnosezustände)
REQ_F_V_06_EXPL	Bereitstellung von Explosionsdarstellungen für z. B. (De-)Montageprozesse
REQ_F_V_07_SELEC	Rückgabe der aktuellen Auswahl aus der 3D-Darstellung an andere Systemkomponenten (z. B. Werkstatttester)
REQ_F_V_08_CONF	Parametrierbare Konfiguration der Darstellung (Kamera, Sichtbarkeit, Interaktionsregeln)

Tabelle 3: Funktionale Anforderungen an den 3D-Viewer (Eigene Darstellung)

Für die 3D-Viewerkomponente gelten zudem die folgenden nicht-funktionalen Anforderungen (Tabelle 4):

ID	Anforderung
REQ_NF_V_01_PERF	Stabile Echtzeitperformanz bei Nutzung realer Konstruktionsdaten (stabile Anzahl der Bilder pro Sekunde bei Modellen mit der Größenordnung von 500.000 Einzelflächen und 200 Teilkomponenten)
REQ_NF_V_02_PLTF	Grundsätzliche Plattformunabhängigkeit bezüglich Touch- und Desktop-Nutzung
REQ_NF_V_03_SIZE	Anpassung an variable Fenstergrößen (responsives Verhalten)
REQ_NF_V_04_HIGHL	Bereitstellung anwendungsbezogener, zielführender Hervorhebungsmechanismen

Tabelle 4: Nicht-Funktionale Anforderungen an den 3D-Viewer (Eigene Darstellung)

Die Anforderungen ergeben sich direkt aus den Nutzerszenarien. Das Hervorheben von bestimmten, ausgewählten Orten (SC_01_COMP_LOC) führt zu der Forderung nach einer grundsätzlichen Anzeige der Daten (REQ_F_V_01_SHOW) und zur Hervorhebung von (Teil)-Komponenten (REQ_F_V_04_HIGHL). Welche Art und Weise der Hervorhebung gewählt wird, ist abhängig vom Kontext der hervorzuhebenden Komponenten innerhalb der 3D-Darstellung. Für kompakte Objekte (im Verhältnis zur Gesamtdarstellung) eignen sich andere Hervorhebungsmechanismen als für stark verzweigende Objekte (Bernhard Preim & Felix Ritter, 2002, S. 13). Letztlich muss die Art und Weise der Darstellung für die jeweils verfolgten Visualisierungsziele sinnvoll sein (REQ_NF_V_04_HIGHL).

Da eine „Komponente“ abhängig vom Kontext aus verschiedenen anderen, individuellen Komponenten besteht, muss eine Struktur in Form eines hierarchischen Aufbaus, wie es grundsätzlich für 3D-Daten üblich ist, existieren. Zusammen mit dem Szenario der Baugruppen-Übersicht (SC_02_GROUPING) ergibt sich damit die Anforderung nach der grundsätzlichen Existenz und Traversierungsmöglichkeit einer solchen Struktur der Daten (REQ_F_V_02_STRUC und REQ_F_V_03_TRAV).

Für die Nutzung der 3D-Ansicht für die Montage bzw. Demontage sind zusätzliche Informationen nötig, die üblicherweise Teil der Dokumentation sind. Das umfasst beispielsweise das Drehmoment für Schrauben, mit denen diese anzuziehen sind, aber auch ergänzende Hinweise, welche Elemente bei einer Inspektion oder während eines Demontagevorgangs ausgetauscht werden müssen (SC_03_ANNOTATION). Verallgemeinert ergibt sich die Anforderung nach der Ergänzung der 3D-Darstellung mit teilebezogenen Zusatzinformationen (REQ_F_V_05_ANNO). Diese Zusatzinformationen können auch genutzt werden, um die Störungsübersicht (SC_02_GROUPING) mit den konkreten Fehlern zu erweitern.

Wird die 3D-Ansicht als Teil des Werkstatttesters genutzt, ist ein Szenario, dass der Nutzer über die 3D-Ansicht zu bauteilspezifischen Programmteilen navigieren kann (SC_04_INTERF_OUT). Für dieses Szenario muss der 3D-Viewer die aktuelle Auswahl anderen Systemkomponenten zur Verfügung stellen (REQ_F_V_07_SELEC).

Für eine sinnvolle Nutzung müssen reale Konstruktionsdaten stabil angezeigt werden können. Bereitgestellte 3D-Daten von einem international agierenden Fahrzeughersteller mit ~9 Milliarden Euro Umsatz (Stand 2022) umfassen über 35 Millionen Einzelflächen. Ein weiteres bereitgestelltes Fahrzeugmodell besitzt etwa 13 Millionen Einzelflächen. In beiden Fällen handelt es sich dabei um die Gesamt-Fahrzeugmodelle inklusive aller Einzelteile. Eine derartige detailreiche Darstellung ist nach derzeitigem Stand für die Dokumentation nicht nötig. Wenn beispielsweise eine Außenansicht des Fahrzeugs benötigt wird, können alle inneren Bestandteile weggelassen werden. Für das Ende-zu-Ende-System wurde ein Elektromotor mit 215 individuell adressierbaren Teilkomponenten als repräsentativer Testfall gewählt. Er besteht insgesamt aus ca. 500.000 Einzelflächen. Mit dieser Größenordnung muss das System – insbesondere der 3D-Viewer – performant arbeiten können (REQ_NF_V_01_PERF).

Wie eingangs bereits beschrieben, wird der Werkstatttester sowohl in klassischen Desktopumgebungen als auch auf Touchgeräten verwendet. Dementsprechend sollte auch die Nutzung in beiden Umgebungen möglich sein. Für eine Nutzung auf unterschiedlichen Gerätetypen und bei der Integration in die Standard-Dokumentation ist ein responsives Design wichtig (REQ_NF_V_03_SIZE).

3.2.2 Anforderungen an den Dokumentationsviewer

Die Kernaufgabe des Dokumentationsviewers ist die Anzeige der ursprünglichen Dokumentation, ergänzt mit den 3D-Darstellungen. Daraus ergeben sich zwei funktionale Anforderungen: erstens die Darstellung der bestehenden Dokumentationsinhalte (REQ_F_DV_01_DOC), zweitens die Integration der 3D-Ansicht in die Oberfläche der Dokumentation (REQ_F_DV_02_INT). Damit können die Bestandteile der Dokumentation beispielsweise bezüglich der Lokalisierung von Komponenten zur Inspektion (SC_01_COMP_LOC) oder als Unterstützung der Montageprozesse (SC_03_ANNOTATION) direkt mit der 3D-Ansicht referenziert werden. Um eine gleichzeitige, flexible Nutzung mehrerer Darstellungen zu erlauben, wie es für komplexere Prozesse nötig ist, sollte es möglich sein, die 3D-Ansicht außerhalb der direkten Dokumentationsdarstellung anzuzeigen (REQ_F_DV_03_INT).

Die entsprechenden funktionalen Anforderungen sind in Tabelle 5 gelistet:

ID	Anforderung
REQ_F_DV_01_DOC	Darstellung bestehender Dokumentationsinhalte
REQ_F_DV_02_INT	Integrierbare Anzeige der interaktiven 3D-Ansichten als Bestandteil der Dokumentation
REQ_F_DV_03_EXT	Möglichkeit zur parallelen, externen Darstellung von 3D-Inhalten außerhalb der eingebetteten Dokumentationsansicht

Tabelle 5: Funktionale Anforderungen an den Dokumentationsviewer (Eigene Darstellung)

Für eine tatsächliche Nutzung ist es wichtig, die Hemmschwelle für den Einsatz zu minimieren (SC_07_DOCU). Industriestandard ist nach Engelke et al. (2015, S. 1) die Bereitstellung der Dokumentation im Stil von Papier-Vorlagen, die dann als PDF ausgeliefert wird (SC_06_LAW_MIN). Für eine Integration der 3D-Darstellungen ist es damit von Interesse, diese bereits existierende Dokumentation ohne tiefgreifende Veränderung verwenden zu können (REQ_NF_DV_01_MOD).

Arbeiten ohne Dokumentation ist nahezu unmöglich aufgrund der hohen Modellvielfalt der Fahrzeuge. Besonders Inspektionen müssen nach vorgegebenen Punkten abgearbeitet werden und sind „hoch strukturiert“ (Rauner et al., 2002, S. 27). Von besonderer Relevanz ist deshalb ein fehlerrobustes Verhalten des Dokumentationsviewers bei dem Laden und Anzeigen der 3D-Darstellungen, um die grundsätzliche Verfügbarkeit der Dokumentation möglichst abzusichern (REQ_NF_DV_02_ROB).

Die entsprechenden nicht-funktionalen Anforderungen sind in Tabelle 6 gelistet:

ID	Anforderung
REQ_NF_DV_01_MOD	Integration der 3D-Darstellungen ohne tiefgreifende Modifikationen bestehender Dokumentationsprozesse
REQ_NF_DV_02_ROB	Fehlerrobustes Verhalten bei nicht ladbaren 3D-Inhalten oder beschädigten Konfigurationen

Tabelle 6: Nicht-Funktionale Anforderungen an den Dokumentationsviewer (Eigene Darstellung)

3.2.3 Anforderungen an die Datenaufbereitungspipeline

Die Datenaufbereitungspipeline muss als Kernaufgabe die CAD-Daten so transformieren und konvertieren, dass sie für die Anzeige im 3D-Viewer genutzt werden können. Wichtig ist, dass dabei die Visualisierung im Vordergrund steht und nicht etwa die Detailtreue (REQ_F_P_01_CAD).

Um das Szenario der logischen und/oder physischen Baugruppen (SC_02_GROUPING) zu unterstützen, muss es die Möglichkeit geben, diese als Teil der Pipeline zu definieren (REQ_F_P_01_STRUC). Die Neustrukturierung der Daten gehört ebenso zur Konfiguration einer einzelnen Darstellung, wie die Angabe initialer Kamerapositionen oder Hervorhebungsparameter. Insgesamt sollten alle Konfigurationsmöglichkeiten des 3D-Viewers über die Pipeline in Form einer „Meta“-Konfiguration spezifiziert werden können (REQ_F_P_03_SCENE).

Als Teil des Dokumentationsbereitstellungsprozesses ist es notwendig festzulegen, welche Stellen in der bestehenden Dokumentation mit welchen 3D-Darstellungen verknüpft werden sollen (REQ_F_P_04_DOC). Daraus müssen dann auslieferungsbereite Pakete generiert werden können, um die Dokumentation der Komponentenhersteller den Werkstätten bereitstellen zu können (REQ_F_P_05_EXP). Damit können dann die Szenarien zur Anzeige der Baugruppen (SC_02_GROUPING) oder auch zur Lokalisierung bestimmter Komponenten (SC_01_COMP_LOC) an den jeweiligen Stellen in der Dokumentation zielführend durch die 3D-Darstellung unterstützt werden (SC_07_DOCU).

Die entsprechenden funktionalen Anforderungen sind in Tabelle 7 gelistet:

ID	Anforderung
REQ_F_P_01_CAD	Aufbereitung von CAD-Daten zu visualisierungsorientiertem Format
REQ_F_P_02_STRUC	Möglichkeit für nutzergesteuerte Neustrukturierung der Daten gemäß logischer oder physischer Gruppierung (z. B. Baugruppen)
REQ_F_P_03_SCENE	Generierung konfigurierbarer Darstellungsbeschreibungen

REQ_F_P_04_DOC	Konfiguration der Verknüpfung von 3D-Darstellungen mit Dokumentationsinhalten
REQ_F_P_05_EXP	Erstellung auslieferungsgeeigneter Datenpakete

Tabelle 7: Funktionale Anforderungen an die Datenaufbereitungspipeline (Eigene Darstellung)

Als grundlegende Eigenschaft sollte die Pipeline für gleichen Input auch die gleichen Ergebnisse liefern (REQ_NF_P_01_REPR), mit einem möglichst hohen Grad der Automatisierung (REQ_NF_P_03_AUTO) und geringen Anforderungen an den Konfigurator (REQ_NF_P_04_NOEX). Das entspricht der Anforderung nach möglichst wenig Aufwand zur Erstellung und Reproduzierbarkeit der Dokumentation (SC_07_DOCU).

Da die 3D-Daten Teil der Dokumentation werden, muss für sie ebenso ein sinnvolles Update- und Änderungsmanagement existieren. Das betrifft einerseits das Änderungsmanagement der Konfiguration der Aufbereitung für die entsprechenden Darstellungen (REQ_NF_P_06_UPD) und andererseits die Erweiterbarkeit der Pipeline selbst (REQ_NF_P_05_MOD).

Um die Allgemeingültigkeit der Datenaufbereitungspipeline zu sichern, sollten gängige CAD-Formate unterstützt werden (REQ_NF_P_02_SUPP).

Die entsprechenden nicht-funktionalen Anforderungen sind in Tabelle 8 gelistet:

ID	**Anforderung**
REQ_NF_P_01_REPR	Reproduzierbarkeit der Pipeline-Ergebnisse bei identischen Eingabedaten
REQ_NF_P_02_SUPP	Unterstützung gängiger CAD-Datenformate (z. B. STEP, JT etc.)
REQ_NF_P_03_AUTO	Hoher Automatisierungsgrad mit minimal manuellen Schritten
REQ_NF_P_04_NOEX	Keine Notwendigkeit von Expertenwissen zur Konfiguration der Pipeline
REQ_NF_P_05_MOD	Unterstützung modularer Erweiterung (z. B. zusätzliche Verarbeitungsschritte)
REQ_NF_P_06_UPD	Unterstützung für Update- und Änderungsmanagement zur Nachverfolgbarkeit von Input-, Output- und Konfigurationsänderungen

Tabelle 8: Nicht-funktionale Anforderungen an die Datenaufbereitungspipeline (Eigene Darstellung)

4 Architektur

In den folgenden Kapiteln wird die Architektur des Ende-zu-Ende-Systems erläutert. Zu Beginn wird in 4.1 der Systemkontext beschrieben und erklärt, inwieweit die bestehenden Datenbereitstellungsprozesse ergänzt werden müssen. In 4.2 werden die grundlegenden Konzepte definiert, auf denen die Architektur und die spätere Implementierung basieren. Die hierarchisch aufgebauten Bausteinsichten finden sich in 4.3 mit der Gesamtübersicht und in 4.4 mit den einzelnen Systemmodulen.

Wesentliche Implementierungsdetails und -entscheidungen sind erst im nachfolgenden Teil 5 zu den jeweiligen Systemmodulen beschrieben.

4.1 Systemkontext und Integration in bestehende Datenbereitstellungsprozesse

Wie in 2.2 bereits beschrieben, muss der Fahrzeugkomponentenhersteller die Verwendung des Werkstatttesters vorbereiten. Das umfasst die Programmierung von Diagnosefunktionen und die Definition der Fahrzeugmodellhierarchie. Das Ergebnis ist ein Diagnoseprojekt. Erstellt wird das Diagnoseprojekt mit einem entsprechend spezialisierten separaten Programm.

Eine 3D-Darstellung ist gemäß den Anforderungen für eine Gesamtübersicht der Steuergeräte als Teil des Werkstatttesters gewünscht. Um die passenden Darstellungen für die jeweiligen Fahrzeuge anzuzeigen, muss ein Datenmapping zwischen den Fahrzeugdaten und den 3D-Darstellungen als Teil des Diagnoseprojekts festgelegt werden.

Die Abbildung 2 beinhaltet dieses Datenmapping als Interaktion zwischen den 3D-Darstellungen und dem Diagnoseprojekt-Erstell-Tool. Die 3D-Darstellungen müssen grundsätzlich durch den Fahrzeugkomponentenhersteller auf Basis der Fertigungs-CAD-Daten erstellt werden, damit sie dann später beispielsweise für die Störungsübersicht oder als Visualisierungsverbesserung in der Dokumentationsdarstellung genutzt werden können (siehe rechte Seite in Abbildung 2).

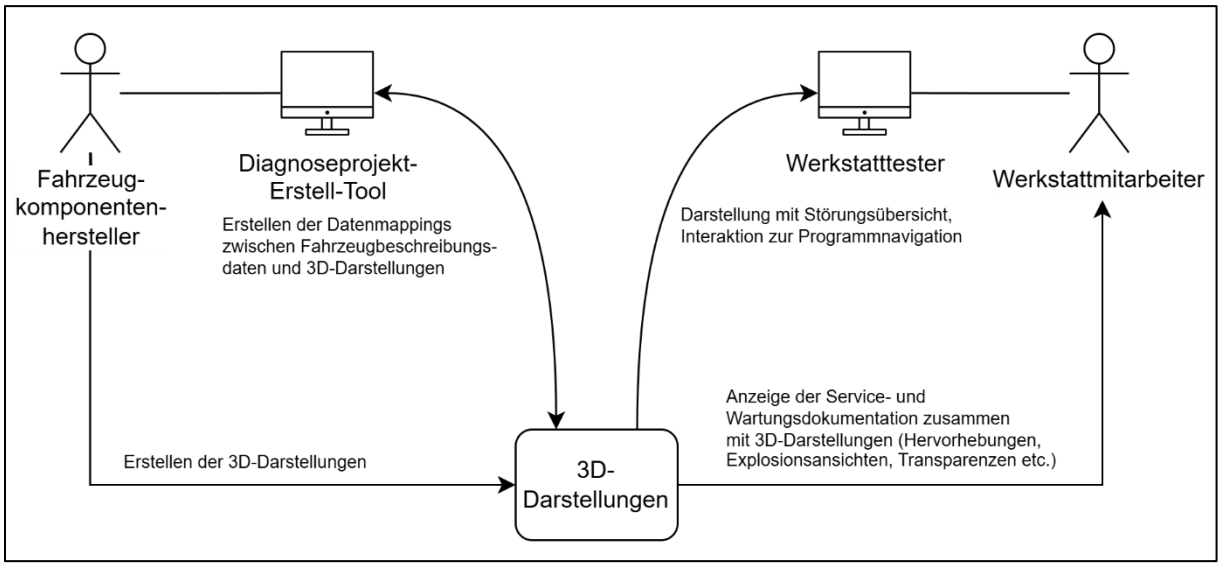


Abbildung 2: Initialer Kontext der 3D-Darstellungen (Eigene Darstellung)

Eine Integration der 3D-Darstellung ist darauf aufbauend Teil der Erstellung der Diagnoseprojekte für den Werkstatttester mit zwei Schritten:

1. dem Erstellen der 3D-Darstellungen und
2. dem Datenmapping zwischen 3D-Darstellungen und der Fahrzeugmodellhierarchie

Die Abbildung 3 zeigt vereinfacht den Prozess der Bereitstellung der 3D-Darstellungen für den Werkstatttester. Die linke Seite zeigt die Vorbereitung durch den Fahrzeugkomponentenhersteller und die rechte Seite zeigt die Verwendung in der Werkstatt. Blau hinterlegte Kästen sind die Bereiche, die sich mit der Integration der 3D-Darstellungen ändern.

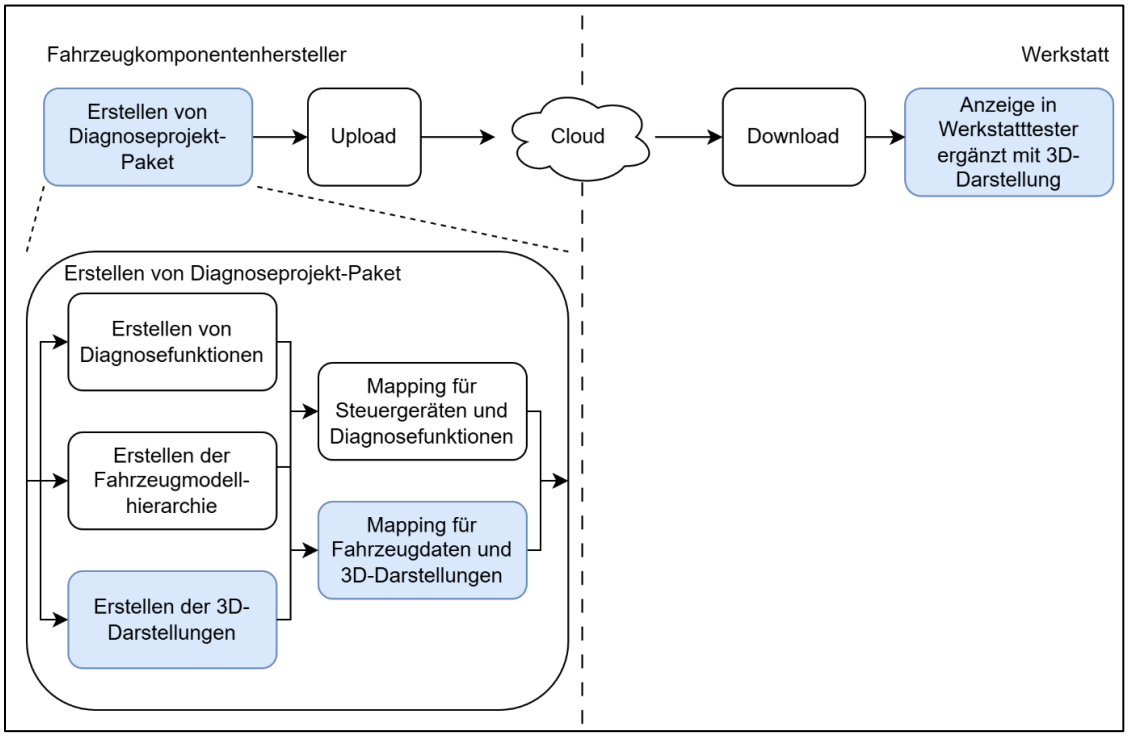


Abbildung 3: Bereitstellung der 3D-Darstellungen als Teil der Diagnoseprojekte für den Werkstatttester (Eigene Darstellung)

Die Abbildung 4 zeigt vereinfacht den Prozess der Bereitstellung der 3D-Darstellungen als Teil der Service- und Wartungsdokumente. Unabhängig von der konkreten Gestaltung der Dokumente als PDF oder HTML, muss festgelegt werden, welche 3D-Darstellung an welchem Ort in der Dokumentation angezeigt werden soll. Zusätzlich muss die Anzeige in der Werkstatt so erweitert werden, dass sie in der Lage ist, die neuen 3D-Darstellungen anzuzeigen.

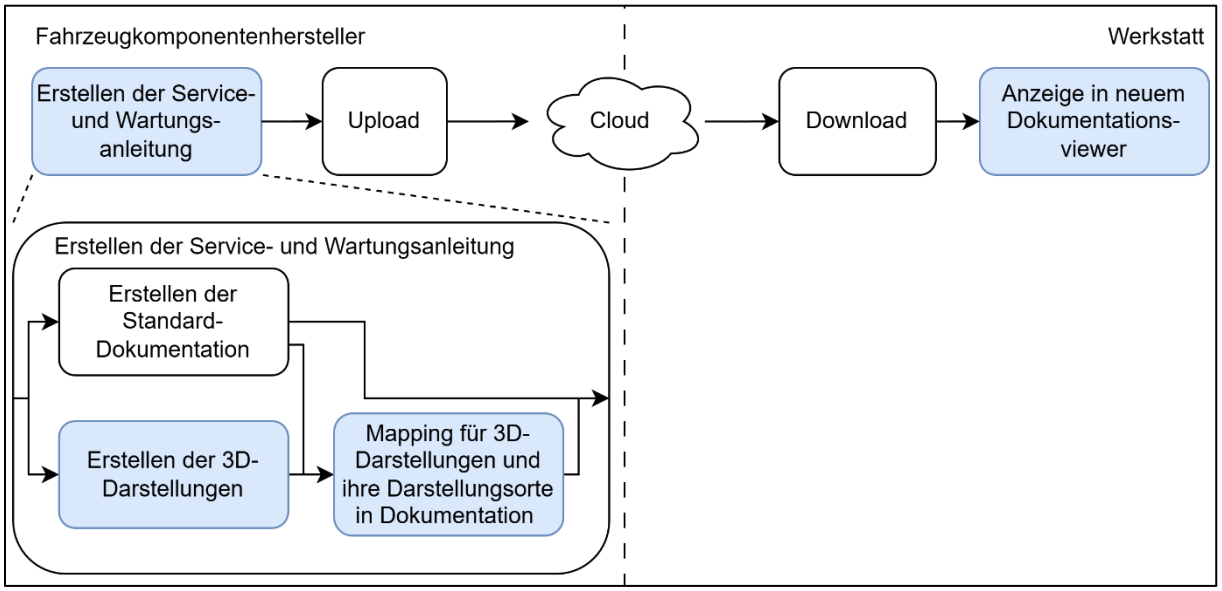


Abbildung 4: Bereitstellung der 3D-Darstellungen als Teil der Service- und Wartungsdokumente (Eigene Darstellung)

Für das Erstellen der 3D-Darstellungen und für die Anzeige der Service- und Wartungsdokumentation zusammen mit den 3D-Darstellungen werden zwei neue Systeme erstellt: den Doc3DCreator und den Dokumentationsviewer. Die Abbildung 5 zeigt den Kontext, ergänzt mit den zwei Systemen.

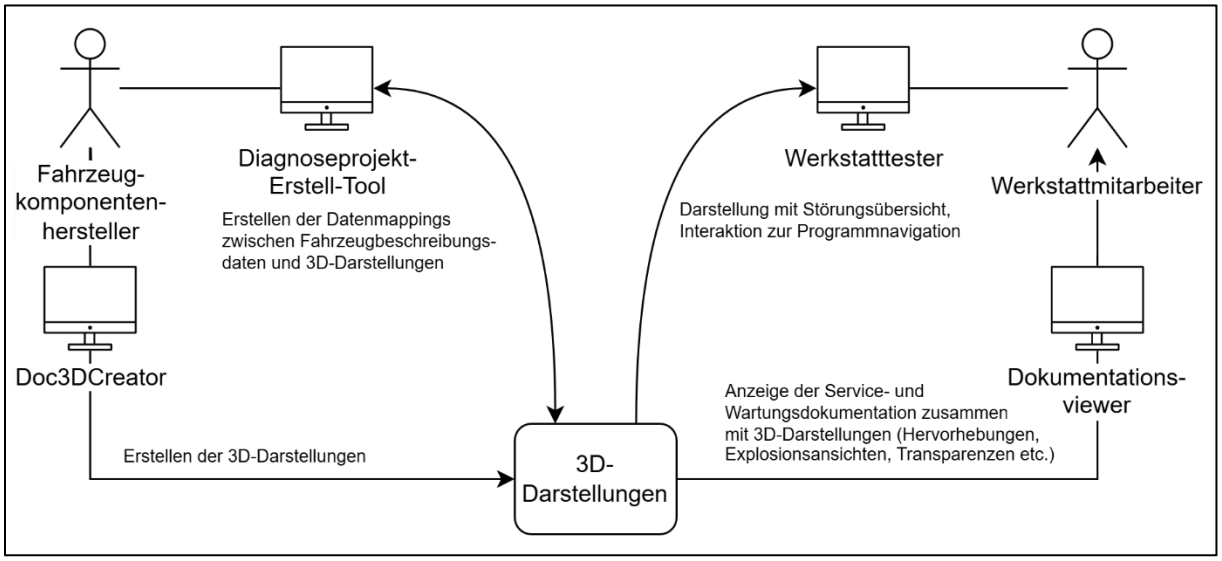


Abbildung 5: Kontext der 3D-Darstellungen ergänzt mit Doc3DCreator und Dokumentationsviewer (Eigene Darstellung)

4.2 Basiskonzepte: Modell, Szene, Dokument

Zur sauberen konzeptionellen Trennung der Systembestandteile werden im Folgenden die zentralen Domänenobjekte des Systems definiert: Modell, Szene und Dokument. Die Basis für die 3D-Darstellungen sind die 3D-Modelle aus der Fertigung. Der Industriestandard ist die Modellierung komplexer Systeme in Form von Bauteil-Hierarchien (Bharadwaj & Starly, 2022). Wiederverwendbarkeit von Standard-Bauteilen reduziert den Entwicklungsaufwand. Verschiedene Ansätze zur Verbesserung der Wiederverwendung von Bauteilen existieren, aber die Grundidee zur Wiederverwendung häufiger auftretender Muster bleibt (Vasantha et al., 2021, S. 1). Für die Verwendung der 3D-Fertigungsdaten ist die Wiederverwendbarkeit integral, um konsistente Ergebnisse mit möglichst wenig Aufwand zu erhalten.

Die 3D-Darstellung ergibt sich aus dem darzustellenden Objekt und der Art und Weise, wie es dargestellt wird. 3D-Daten sind grundsätzlich Beschreibungen von geometrischen Objekten. Die meisten Dateiformate sind in der Lage, zusätzlich zur Geometrie auch Informationen zum Material des Bauteils und damit zur Darstellung anzugeben. Die Basis für eine realitätsnahe Darstellung der 3D-Objekte sind Modelle mit entsprechend realitätsnahen Materialien.

1. Für den Kontext dieses Systems ist ein „Modell“ definiert als eine geometrische Beschreibung eines Bauteils, ergänzt mit Materialien, die das Aussehen des Modells vorgeben, und einer modellspezifischen, hierarchischen Bauteilstruktur. Ein Modell ist in diesem System zustandslos und ausschließlich für die visuelle Darstellung vorbereitet. Es enthält keine szenenspezifischen Interaktionen oder Kontexte.
2. Eine „Szene“ legt die Regeln zur Darstellung und Interaktion für ein oder mehrere Modelle fest. Eine 3D-Darstellung ist eine Szene mit definierten Modellen. Die „Darstellung“ betrifft hier Veränderungen am Modell selbst, beispielsweise für dynamische Hervorhebungen, aber auch Beschreibungen der globalen Umgebung wie Hintergrund und Licht.
3. Ein „Dokument“ ist jede Form von Service- und Wartungsdokumentation, die als einzelnes Paket der Werkstatt zur Verfügung gestellt wird. Industriestandard ist hier PDF.

Wichtige, grundlegende Aspekte der Pipeline sind die Reproduzierbarkeit (REQ_NF_P_01_REPR) und Wiederverwendbarkeit der Konfigurationen für ein sinnvolles Management der 3D-Darstellungen über einen längeren Zeitraum hinweg (REQ_NF_P_06_UPD).

Die Abbildung 6 zeigt eine mögliche Instanziierung der Verhältnisse zwischen Modellen, Szenen und Dokumenten. Verschiedene Szenen können Modelle wiederverwenden (z. B. die Fahrzeugchassis). Analog können Szenen in unterschiedlichen Dokumenten wiederverwendet werden. Ein spezialisiertes Dokument für den Motor integriert ggf. die Gesamtübersicht und eine detaillierte Darstellung des Motors, während ein vereinfachter UserGuide ggf. auf die Detailansicht verzichtet. Die Übersicht der fehlerhaften Steuergeräte nach SC_02_GROUPING als Teil des Werkstatttesters entspricht einer Szene, die dementsprechend die fehlerhaften Steuergeräte als Modelle beinhaltet.

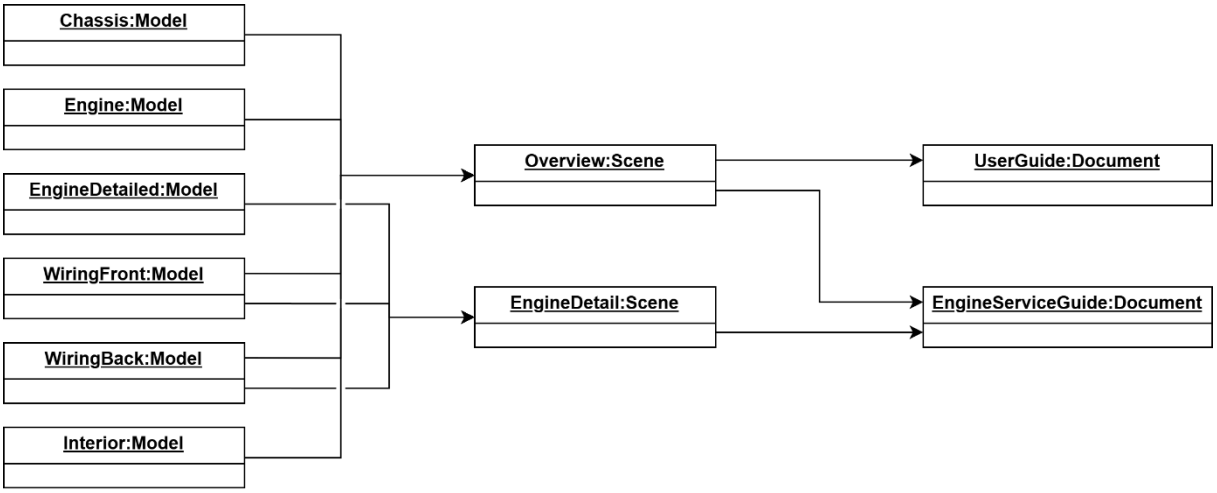


Abbildung 6: Beispiel für mögliche Modell-, Szenen- und Dokument-Instanzen (Eigene Darstellung)

4.3 Systemarchitektur und Realisierungsansicht

Aus den Anforderungen ergibt sich die Systemarchitektur in Abbildung 7. Die zentrale Systemkomponente ist der **Viewer3D**, der entsprechend der Anforderung REQ_F_V_01_SHOW passend aufbereitete Szenen anzeigt. Der **Viewer3D** bietet drei wesentliche Schnittstellen, wie sie auch nach Nandico (2023, S. 20) identifiziert wurden: Die grundsätzliche Verwendung als Teil einer Nutzeroberfläche, die Rückgabe der aktuellen Auswahl (REQ_F_V_07_SELEC) und die Konfiguration der Art und Weise wie die Szene dargestellt wird (REQ_F_V_04_HIGHL und REQ_F_V_08_CONF).

Der **Viewer3D** wird als Nutzeroberfläche zum einen im **Werkstatttester** und zum anderen im **DocumentViewer** genutzt. Ergänzend verwenden ihn auch die Pipeline-Komponenten als Vorschau-Anzeige.

Im Gegensatz zum **DocumentViewer** benötigt der **Werkstatttester** zusätzlich zur UI-Darstellung ein Interface, um die aktuelle Auswahl abzurufen (REQ_F_V_07_SELEC) und Zusatzinformationen beispielsweise zu aktuellen Fehlerzuständen zu ergänzen (REQ_F_V_05_ANNO).

Die Konfiguration der Szenen erfolgt über die **SceneConfiguration**. Sie gibt an, welche Modelle auf welche Art und Weise angezeigt werden (REQ_F_V_04_HIGHL und REQ_F_V_08_CONF) und wie mit ihnen interagiert wird (REQ_F_V_03_TRAV). Zudem enthält sie auch die szenenspezifische Strukturbeschreibung (REQ_F_V_02_STRUC). Die Vorbereitung der angezeigten Modelle erfolgt durch die **ModelPreparation** Komponente.

Der **DocumentViewer** zeigt 3D-Szenen in Kombination mit der Darstellung der Standarddokumentation. Dafür wird als Teil der Aufbereitungspipeline das Mapping zwischen Orten in Dokumenten und Szenen durch die **DocumentPreparation** Komponente erstellt.

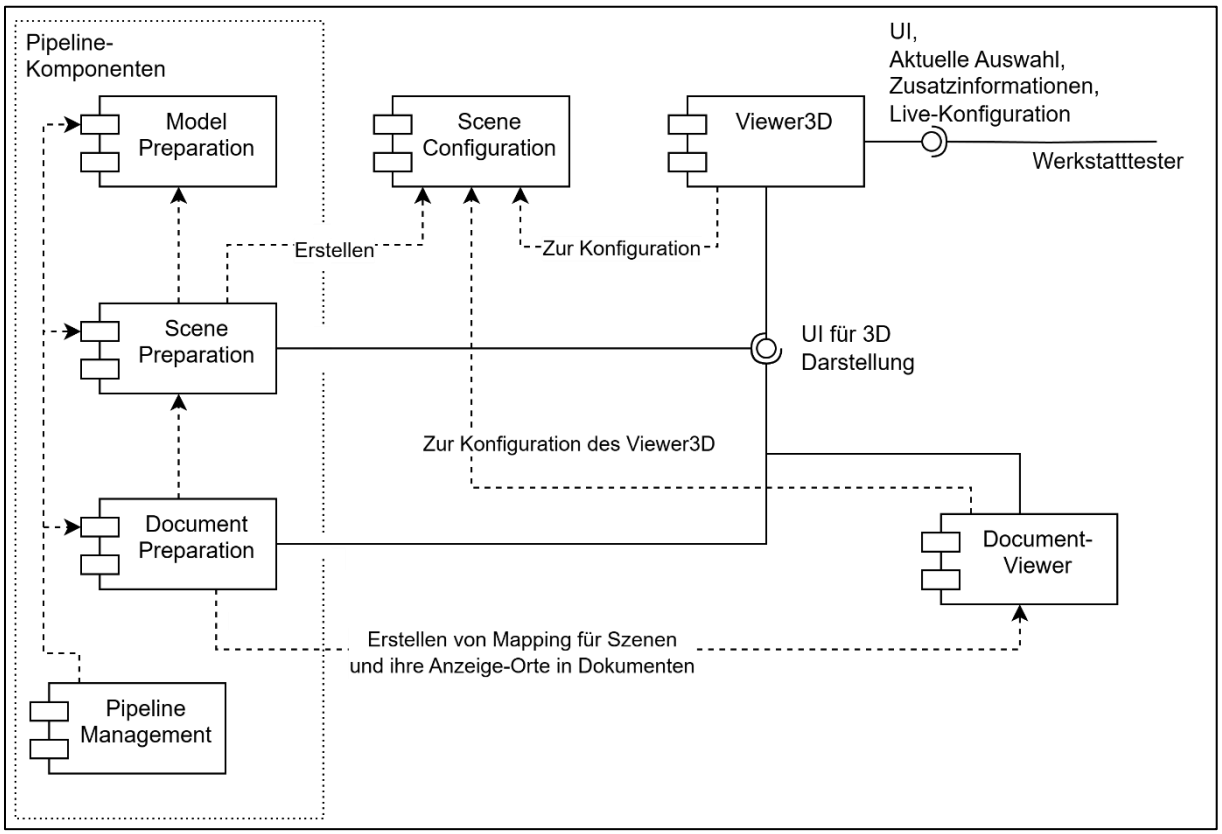


Abbildung 7: Übersicht der Systemkomponenten (Eigene Darstellung)

Die Komponenten werden in drei Anwendungen realisiert: dem **Doc3DCreator** als Rahmenanwendung für die Pipeline, dem **Werkstatttester** und der **DocumentViewer** Applikation. Die Abbildung 8 zeigt die konkrete Zuweisung der Komponenten auf die Applikationen. Der **Doc3DCreator** ist die zentrale Anwendung für die Fahrzeugkomponentenhersteller für die Aufbereitung der CAD-Daten. Sowohl der **Werkstatttester** als auch die **DocumentViewer**-Anwendung verwenden die **SceneConfiguration**, um die Konfiguration für den Viewer3D bereitzustellen.

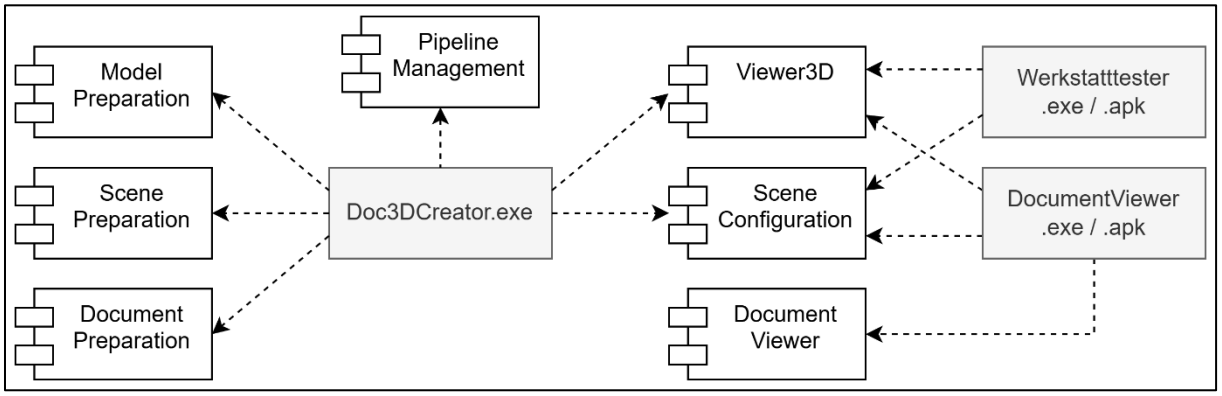


Abbildung 8: Realisierungsansicht für die Systemkomponenten (Eigene Darstellung)

4.4 Architektur der Systemkomponenten

Im Folgenden werden die Systemkomponenten konkretisiert. In 4.4.1 wird der Aufbau der **Datenaufbereitungspipeline** beschrieben. In 4.4.2 folgt eine kurze Beschreibung des **DokumentationsViewers**. Die Komponente **3D-Viewer** wird separat in 4.4.3 behandelt.

4.4.1 Datenaufbereitungspipeline

Das Grundziel der Pipeline ist die Aufbereitung der CAD-Daten und die Vorbereitung auslieferbarer Pakete (REQ_F_P_05_EXP), die dann im Werkstatttester und Dokumentationsviewer angezeigt werden können. Nach Nandico (2023, S. 26) ergibt sich der in Abbildung 9 gezeigte Prozess für die Aufbereitung von CAD-Daten für die Verwendung in 3D-Darstellungen fokussiert für den Werkstatttester. Der Prozess ist zwar konkret auf das verwendete Framework Qt/QML zugeschnitten, lässt sich jedoch grundsätzlich in andere Frameworks übertragen, da auch andere Frameworks von einer spezifischen Formatkonvertierung profitieren können.

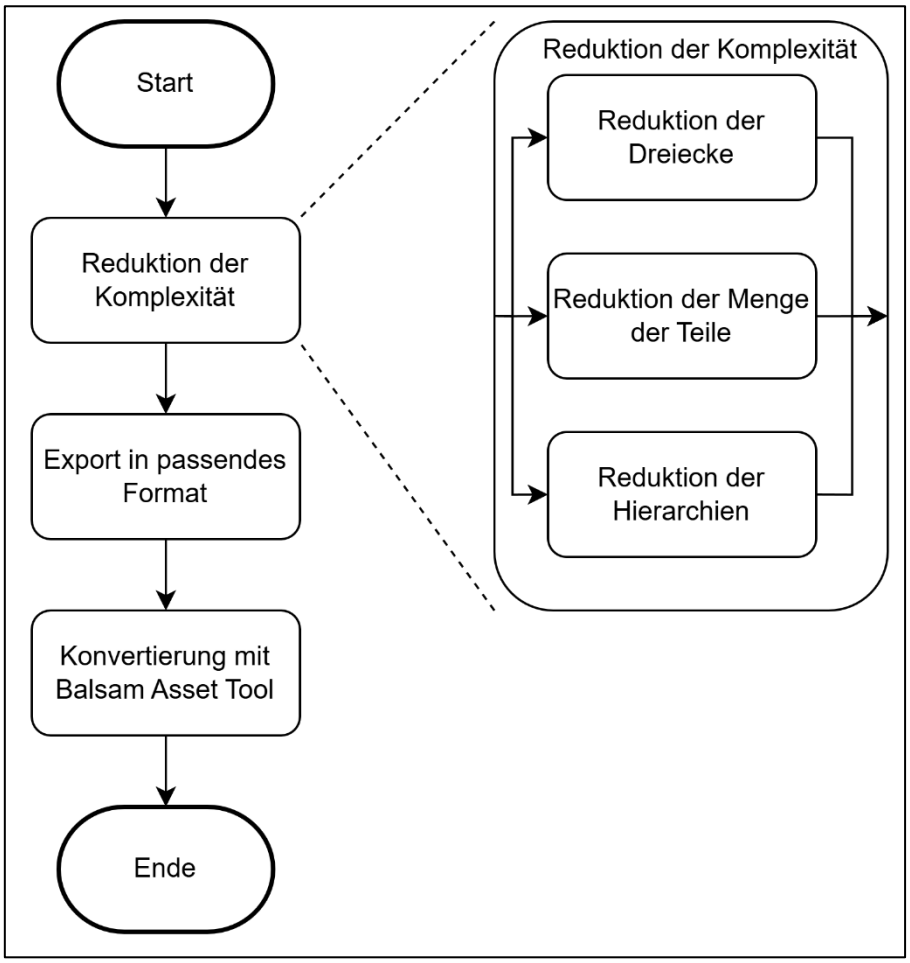


Abbildung 9: Prozess der Datenaufbereitung für Verwendung im Werkstatttester (Darstellung leicht abgewandelt, inhaltlich nach Nandico (2023, S. 26))

Für die Performanz im Ende-zu-Ende-System ist besonders der Schritt der Reduktion der Komplexität von Interesse. Er wird in die Optimierung der angezeigten Teile, die Reduktion der insgesamt angezeigten Teile und die Reduktion der Struktur der Teile aufgeteilt. Diese Schritte finden sich auch im Ende-zu-Ende-System wieder. Die Abbildung 10 zeigt die Schritte des neuen Prozesses:

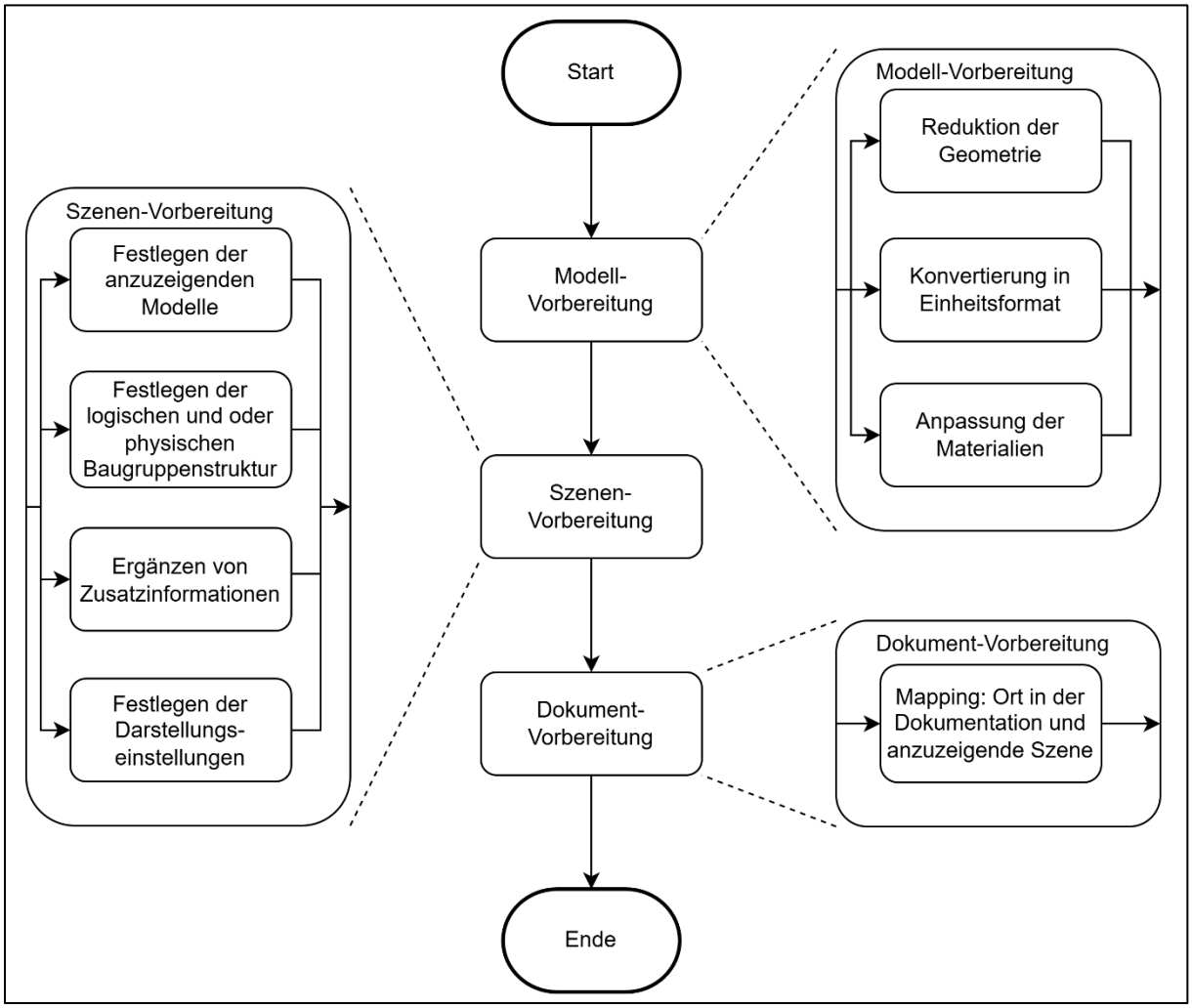


Abbildung 10: Aufbereitung der Daten für die Verwendung als Teil der Wartungsdokumentation (Eigene Darstellung)

Es ist notwendig, aufgrund der unterschiedlichen Ansprüche der CAD-Programme und der Werkstatttester, die 3D-Daten in ihrer Geometrie zu vereinfachen (Nandico, 2023, S. 23–27). Da das Ende-zu-Ende-System auch dazu dient, die Daten für die Verwendung im Werkstatttester vorzubereiten, ist die Reduktion der Geometrie ebenso Teil des Aufbereitungsprozesses.

Für die Pipeline ist eine Konvertierung in ein konsistentes Datenformat grundsätzlich von Vorteil, um die Schritte zu vereinfachen, und ist ebenso Teil der Modellvorbereitung. Die Konvertierung erfolgt dabei nicht in ein allgemeines Standard-Format, sondern in ein konkret für die Verwendung in dem entwickelten 3D-Viewer geeignetes Einheitsformat.

Wie bereits in 4.2 kurz beschrieben, wird die Darstellung von dreidimensionalen Objekten üblicherweise über das Material des Objekts gesteuert. Ein Renderer approximiert dann die reale Darstellung dieses „Materials“ im Kontext der Szene (Licht, Okklusion, Reflexion etc.). Je besser das 3D-Modell die Realität approximiert, desto einfacher wird die Verwendung in der Werkstatt. Für die Fertigung sind diese darstellungsorientierten Materialien von wenig Interesse. Der Fokus liegt auf mathematischer Korrektheit und nicht auf der realitätsnahen Darstellung. Teil der Modellvorbereitung ist es damit, die Möglichkeit zu bieten, die Materialien anzupassen, um eventuell sehr unpassende Materialien zu korrigieren. Ebenso können passendere Materialien zugewiesen werden, wie ein transparentes Material für Autofenster, die möglicherweise in den Daten aus der Fertigung ein einfaches Grau erhalten haben.

Insgesamt ergibt sich die Reduktion der angezeigten 3D-Modelle aus der Trennung in Modelle und Szenen, wie sie in 4.2 beschrieben sind: Eine Szene kann dabei beliebig viele Modelle anzeigen. Das bedeutet, dass durch eine vorherige, sinnvolle Aufteilung die angezeigten Modelle in den Szenen auf das Nötigste reduziert werden können.

Integral ist es, die Möglichkeit zu bieten, die logische und physische Baugruppenstruktur als Nutzer zu definieren. Das entspricht einer Zwischenebene, bei der der Nutzer die gewünschte Struktur anlegt und mit den Verweisen auf die passenden Komponenten in der Szene konfiguriert. Beispielsweise kann ein Hervorheben der neuen, nutzerdefinierten „Tankklappe“ einem Hervorheben der tatsächlichen Komponente „tank_cg_001“ aus der 3D-Datei entsprechen.

Die Vorbereitung der Anzeige in der Dokumentation benötigt nur ein Mapping der konfigurierten Szenen zu festgelegten Orten in der Dokumentation.

Für jeden der drei Hauptvorbereitungsschritte ist es möglich, die konkrete Konfiguration zu persistieren. Jeder der drei Hauptschritte kann „gebaut“ beziehungsweise ausgeführt werden und hat als Ergebnis eine konkrete Instanz entsprechend Abbildung 6. Jeder der Schritte kann über eine Nutzeroberfläche konfiguriert werden, um eine Nicht-Experten-Nutzung zu unterstützen (REQ_NF_P_04_NOEX).

Der folgende Teil beschreibt den konzeptuellen Aufbau der Pipeline-Komponenten. Grundsätzlich unterscheiden wir zwischen einem Step und einem SubStep. Ein Step besteht aus einer beliebigen Anzahl an Substeps. Die Nutzeroberfläche für einen Step setzt sich aus den UI-Komponenten der Substeps zusammen.

Die drei Komponenten ModelPreparation, ScenePreparation und DocumentPreparation sind Steps und entsprechen jeweils in ihrer Grundform der Abbildung 11.

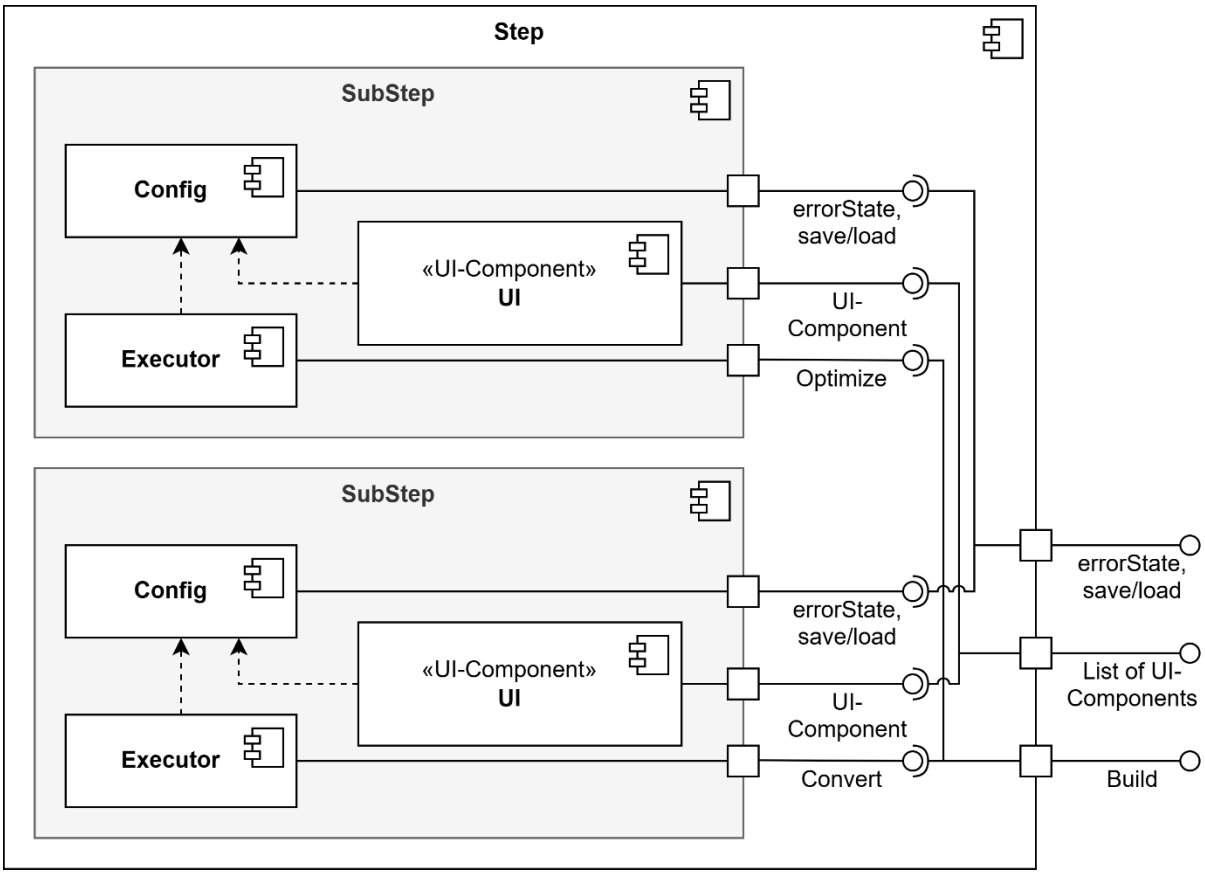


Abbildung 11: Basisaufbau von Pipelinestep und SubStep (Eigene Darstellung)

4.4.1.1 ModelPreparation

Entsprechend dem allgemeinen Pipeline-Step-Aufbau (s. Abbildung 11) zeigt die Abbildung 12 den Aufbau konkret für die Modellvorbereitung. Jeder SubStep stellt Funktionalität für das Speichern und Laden der Konfiguration bereit. Ebenso bietet jeder SubStep eine Funktion für das tatsächliche Durchführen der Aufbereitung. Die UI-Komponenten sowie die Fehlerzustände der einzelnen Schritte werden gesammelt nach außen weitergegeben. Die drei Aufbereitungsschritte Reduktion der Geometrie, Konvertierung in ein Einheitsformat und die Anpassung der Materialien entsprechen den drei Subkomponenten.

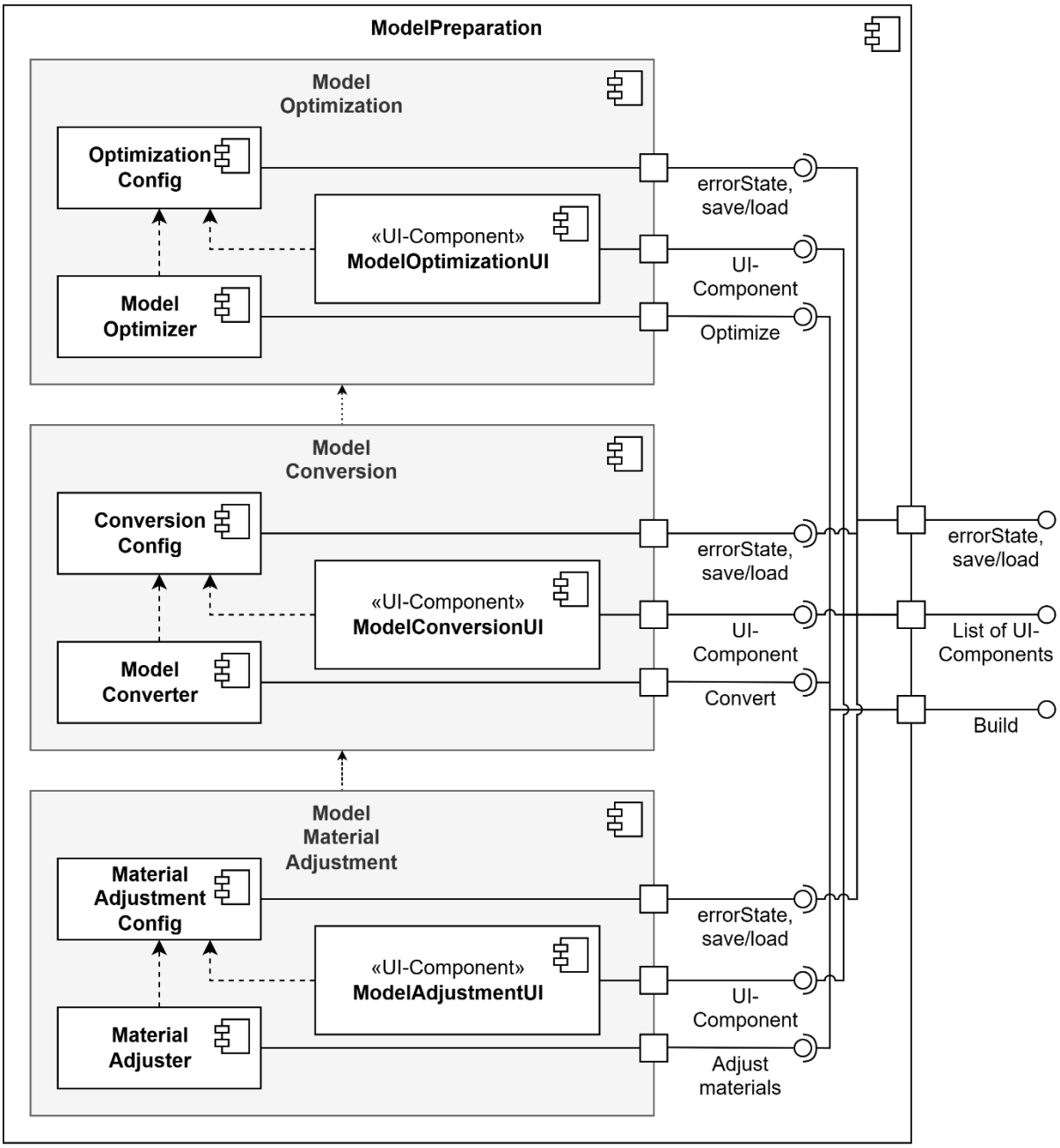


Abbildung 12: Komponenten der Modell-Vorbereitung (Eigene Darstellung)

4.4.1.2 ScenePreparation

Ziel der ScenePreparation ist die Vorbereitung der Szenen. Das umfasst im Wesentlichen die zwei Bereiche:

1. Die Erstellung einer neuen Teilestruktur mit der Ergänzung von teilebezogenen Zusatzinformationen (REQ_F_V_02_STRUC und REQ_F_V_05_ANNO) und der Festlegung, welche Modelle grundsätzlich Teil der Szene sind.
2. Die Konfiguration der Anzeigeeinstellungen der Szene:
    a. Grundsätzliche Konfiguration der Szene mit Hintergrund, Licht, Steuerung, Rendering etc. (REQ_F_V_08_CONF),
    b. Konfiguration der Hervorhebung (REQ_F_V_04_HIGHL),
    c. Konfiguration der Mechanismen zur Strukturtraversierung, wie das Hervorheben von Teilkomponenten beim Anklicken der Beschriftung der Elternkomponente (REQ_F_V_03_TRAV),
    d. Konfiguration der Explosionsdarstellungen (REQ_F_V_06_EXPL)

Die konkrete Umsetzung ist im Kapitel *SceneConfiguration: Struktur und visuelle Konfiguration der Szene* beschrieben. Für die Architektur bedeuten die zwei Bereiche zwei getrennte Komponenten: SceneStructureCreation und SceneUICreation. Die Abbildung 13 zeigt die Komponenten der Szenen-Vorbereitung. Konzeptuell haben wir eine Abhängigkeit der Szenendarstellung von der Szenenstruktur. Wenn bestimmte Objekte zu Beginn hervorgehoben werden sollen, müssen sie zuerst definiert worden sein.

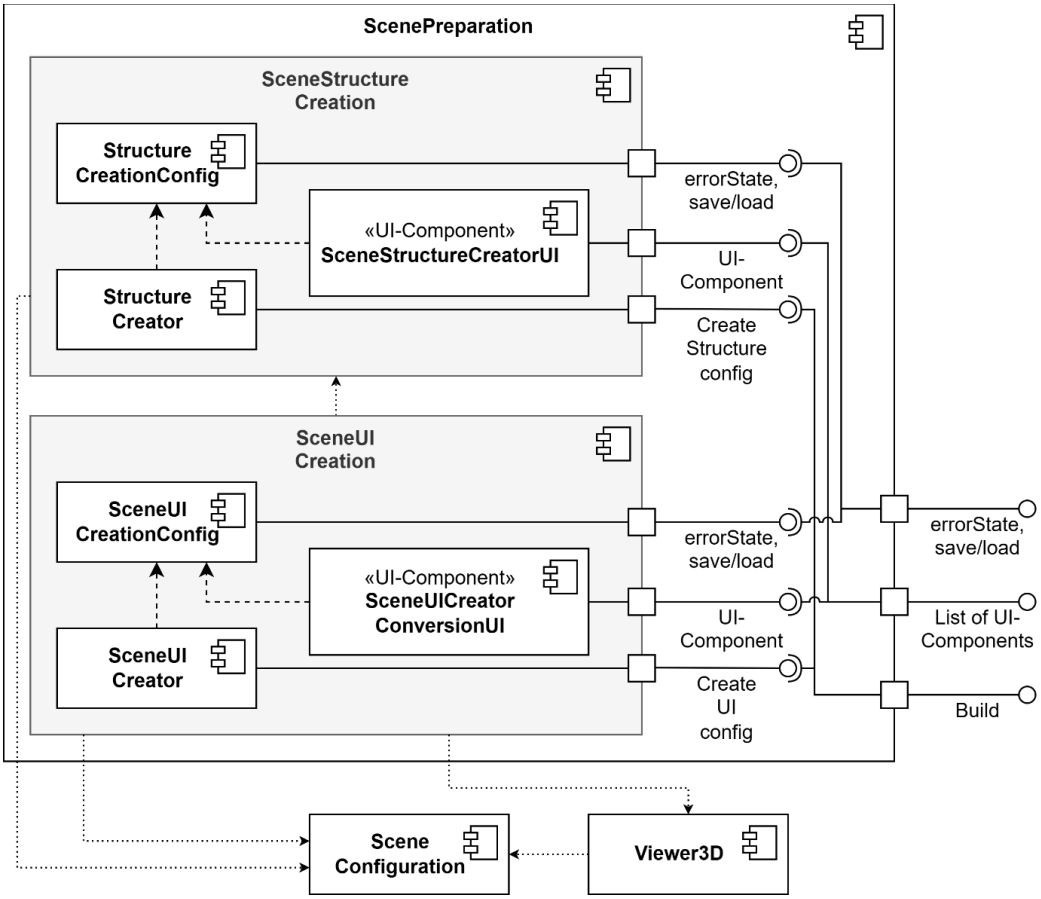


Abbildung 13: Komponenten der Szenen-Vorbereitung (Eigene Darstellung)

Zentrale Komponente für die Definition einer Szene ist die SceneConfiguration Komponente. Diese Komponente enthält die Konfiguration einer kompletten Szene. Die ScenePreparation dient dazu, über ihre zwei Subkomponenten eine spezifische Instanz der SceneConfiguration zu erstellen, die dann von dem Viewer3D genutzt werden kann.

4.4.1.3 DocumentPreparation

Für die Vorbereitung der Dokumentation ist für den aktuellen Anforderungskatalog nur ein Schritt wichtig: das Referenzieren von einem Ort in der Dokumentation mit einer definierten Szene. Die Abbildung 14 zeigt die Komponenten der Dokumenten-Vorbereitung. Wichtig ist, dass der Viewer3D verwendet wird, wenn Szenen gezeigt werden. Ebenso ist die Abhängigkeit von dem DocSceneMappingCreator zum DocumentViewer von besonderer Relevanz: Der DocSceneMappingCreator muss ein Mapping generieren, was dann von dem DocumentViewer genutzt wird, um mithilfe des Viewer3D die Szenen darzustellen.

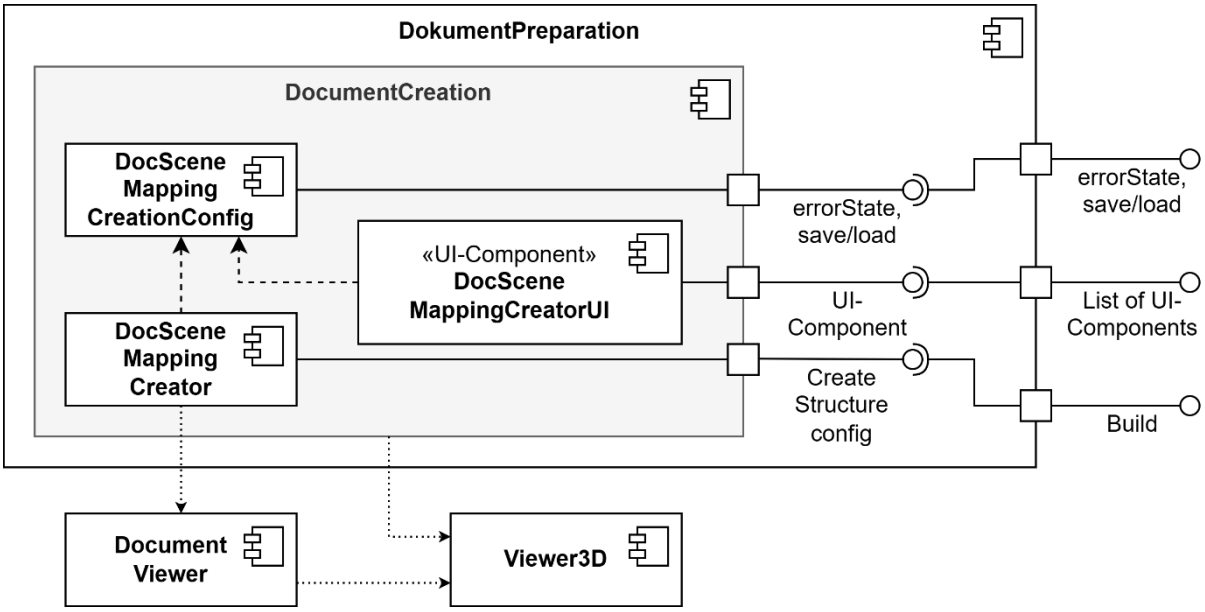


Abbildung 14: Komponenten der Dokumenten-Vorbereitung (Eigene Darstellung)

4.4.2 Dokumentationsviewer

Der Dokumentationsviewer entspricht im Wesentlichen einer Komponente, die die bestehende Dokumentation und die 3D-Darstellungen zusammen anzeigen kann: die Doc3DViewer-Komponente. Für eine Verwendung als Teil des Ende-zu-Ende-Systems ist diese Komponente in die Rahmenanwendung Doc3DViewerApplication eingebunden. Die Abbildung 15 zeigt die Komponenten der Doc3DViewer-Komponente. Für die Darstellung werden die folgenden Daten benötigt:

1. Das Dokument selbst bspw. als PDF
2. Das Mapping von Orten in dem Dokument zu Szenen
3. Der Ort, an dem die Szenendaten vorliegen
4. Der Ort, an dem die Modelle vorliegen

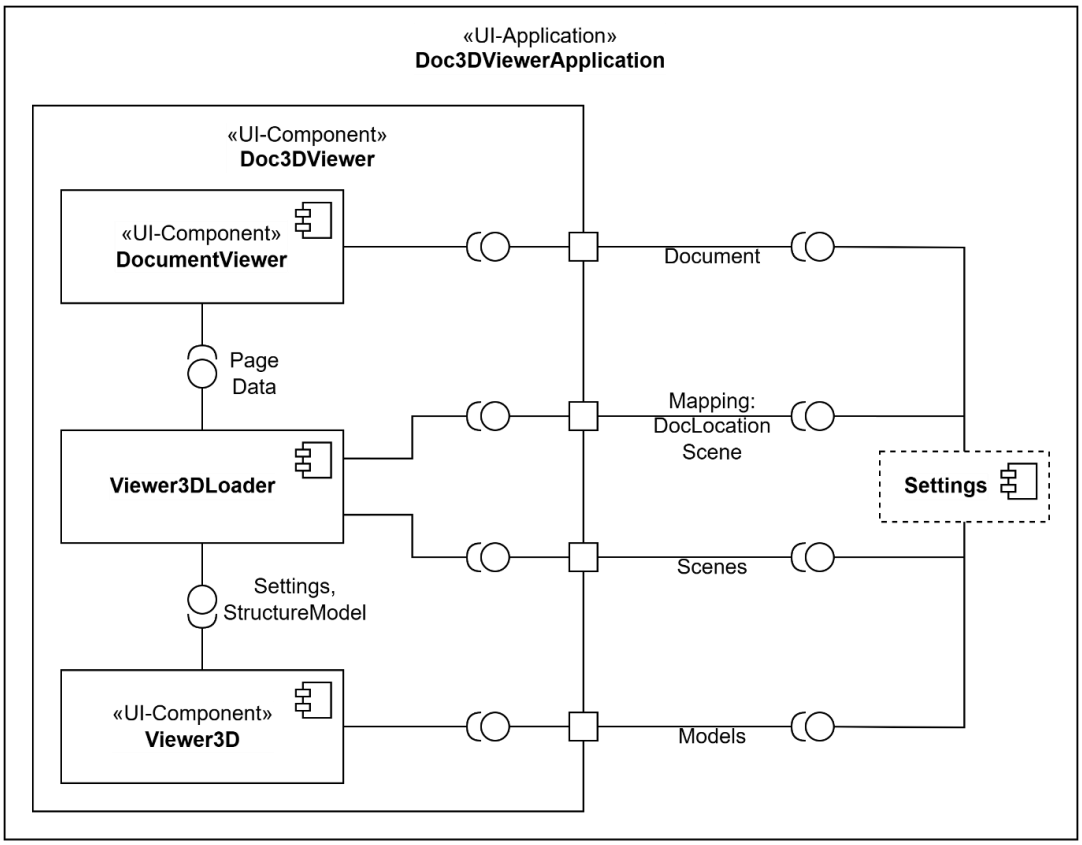


Abbildung 15: Komponenten des Doc3DViewers mit vereinfachter Rahmenanwendung (Eigene Darstellung)

Der **DocumentViewer** zeigt das Dokument an und stellt dem **Viewer3DLoader** die Informationen zur aktuellen Seite wie Seitenzahl und Größe zur Verfügung. Der **Viewer3DLoader** übergibt dann gemäß dem Ort aus dem **Mapping** die angegebene Szeneneinstellungen und das zu ladende **StructureModel** an den **Viewer3D**. Vernachlässigt wird in dieser Darstellung die tatsächliche Bereitstellung der Daten. Grundansatz ist, dass die benötigten Daten entweder dynamisch über einen Server bereitgestellt werden oder initial auf der Festplatte vorliegen. Die konkrete Umsetzung unterscheidet sich nicht von einer generellen Bereitstellung von Daten der Fahrzeughersteller wie Dokumente oder Bildern und muss je nach Anwendungsfall angepasst und umgesetzt werden.

4.4.3 3D-Viewer und Einstellungen

Um die Anforderungen zu erfüllen, muss die 3D-Viewer-Komponente diverse Schnittstellen und Funktionen bereitstellen. In der vorhergehenden Arbeit „Entwicklung eines 3D-Viewers zur dynamischen Visualisierung von Fahrzeug-Komponenten im Bereich der Fahrzeugdiagnose“ (Nandico, 2023) wurde eine Architektur für einen 3D-Viewer als Teil eines Werkstattesters vorgeschlagen. Für die Architektur der 3D-Viewer-Komponente des Ende-zu-Ende-Systems bietet dieser 3D-Viewer die Basis. Die Abbildung 16 zeigt den vereinfachten Aufbau der entwickelten 3D-Viewer-Komponente, der in den folgenden Kapiteln erläutert wird. In dieser Abbildung sind die Schnittstellen zur Konfiguration vernachlässigt, um das Verständnis zu erleichtern. Das **ExtendedStructureModel** repräsentiert die Struktur der Szene. Die aktuelle Auswahl erfolgt über das **SelectionModel**. Auf Basis der Auswahl wird das Aussehen der Komponenten mit dem **MaterialManager** angepasst.

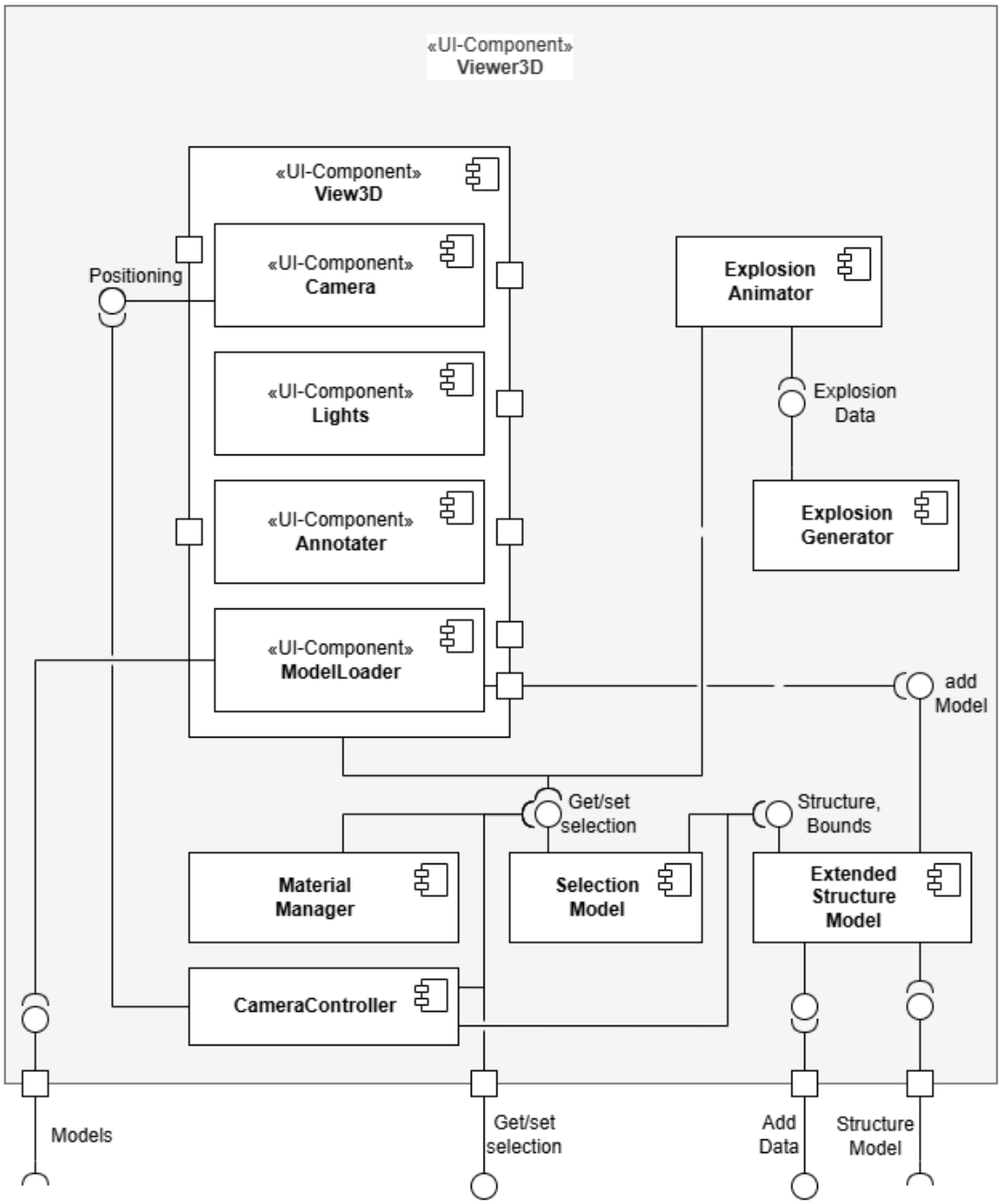


Abbildung 16: Komponentenübersicht des 3D-Viewers ohne Einstellungen (Eigene Darstellung auf Basis von Nandico (2023, S. 29))

Die Szenarien für den 3D-Viewer als Teil des Werkstatttesters nach Nandico (2023) sind in den Anforderungen im Anforderungsteil 3 enthalten. Für die Nutzung als Teil des Ende-zu-Ende-Systems wird die 3D-Viewer Komponente um die grundsätzlichen folgenden Aspekte erweitert (neben Verbesserungen der bereits existierenden Komponenten):

1. Laden mehrerer 3D-Modelle
2. Darstellung und Generation von Explosionsansichten
3. Konfiguration über passende Einstellungen
4. Datenergänzung zur Laufzeit

Im folgenden Kapitel wird die Funktion der Komponenten und die Abgrenzung zur vorhergehenden Implementierung erläutert.

4.4.3.1 Komponentenübersicht und Verantwortlichkeiten

Die **3D-Viewer**-Komponente besteht aus mehreren klar abgegrenzten Subkomponenten mit definierten Verantwortlichkeiten. Die Abbildung 16 zeigt eine reduzierte Darstellung der Komponenten ohne die Einstellungen:

Der **Viewer3D** ist die zentrale und umfassende Komponente, die sich als UI-Element in Anwendungen integrieren lässt. Sie aggregiert die weiteren Teilsysteme und bietet Schnittstellen zur Konfiguration, Interaktion sowie zur Laufzeit-Ergänzung von Daten. Gegenüber der Vorgängerversion wurde der Funktionsumfang insbesondere im Hinblick auf Konfigurierbarkeit und Schnittstellen deutlich erweitert, sowie um neue Darstellungsmechanismen ergänzt.

View3D kapselt die eigentliche visuelle Darstellung der Szene. Sie enthält die 3D-Objekte und stellt die Basisdarstellung bereit. Im Gegensatz zur früheren Architektur ist die Lichtsteuerung nicht mehr Teil dieser Komponente, um deren Verantwortlichkeit auf reine Darstellung zu beschränken.

Die **Lights**-Komponente steuert die Beleuchtung der Szene basierend auf konfigurierbaren Parametern. Sie wurde als eigenständige Einheit ausgegliedert, um gezielt Anpassungen an Lichtverhältnissen und -quellen zu ermöglichen, um so eine realistische Darstellung zu erzeugen, in der sich der Werkstattmitarbeiter schnell zurechtfindet.

Der **ModelLoader** abstrahiert den Ladevorgang der 3D-Modelle. Er erlaubt das dynamische Nachladen beliebig vieler Modelle aus einem Basisverzeichnis.

Der **Annotater** ergänzt die Szene um Annotationen. Dies umfasst Beschriftungen, Pfeile und vergleichbare grafische Hinweise. Die Komponente abstrahiert bewusst von konkreten Textplatzierungen und fasst sämtliche Annotationslogik zusammen. Sie ersetzt die frühere **LabelPlacer**-Komponente funktional und konzeptionell.

Der **CameraController** übernimmt die Steuerung der Kamera im interaktiven Betrieb. Er wurde von einer rein technischen Implementierung (**COrbitCameraController**) auf eine abstraktere Komponente überführt, die sich an funktionalen Anforderungen orientiert.

Das **ExtendedStructureModel** stellt die hierarchische Struktur der Szene bereit, ergänzt um Zusatzinformationen zur Position und Ausdehnung einzelner Komponenten. Sie erweitert damit die reine Hierarchie um Metadaten, die für Selektion und Darstellung notwendig sind.

Das **SelectionModel** verwaltet die aktuell ausgewählten Komponenten und stellt Funktionen zur Modifikation dieser Auswahl bereit. Es arbeitet mit Referenzen zum **ExtendedStructureModel** und ist eine zentrale Schnittstelle für Interaktionen.

Der **MaterialManager** ist verantwortlich für das Anpassen und Wiederherstellen von Materialien der 3D-Modelle. Eine wesentliche Aufgabe besteht in der farblichen Hervorhebung von Komponenten durch gezielte Materialänderungen.

Die Generierung und Darstellung von Explosionsansichten sind in zwei Komponenten getrennt: Der **ExplosionGenerator** berechnet Positionsdaten für die Explosion, während der **Explosions-Animator** die Animation zwischen Normalposition und Explosionsdarstellung steuert.

Die Tabelle 9 zeigt die Komponenten im Vergleich zur Vorgängerversion und deren jeweilige Aufgabenbereiche:

Komponenten Neu	**Verantwortlichkeit**	**Komponenten Alt**
Viewer3D	Gesamthafte Viewer-Komponente mit einbettbarer UI und erweiterten Schnittstellen zur Konfiguration und Laufzeit-Erweiterung	Viewer3D
View3D	Darstellung der Szene; keine Verantwortung mehr für Beleuchtung	View3D
Lights	Separates Management der Beleuchtung entsprechend den Einstellungen	
Model-Loader	Abstrahierter Ladevorgang für mehrere Modelle aus einem Basisverzeichnis anstatt Laden von einzelnem Modell	Loader3D
Annotater	Generalisierte Szenenergänzung durch Annotationen (Labels, Pfeile etc.); ersetzt konkrete Label-Platzierung	Label-Placer
Camera-Controller	Abstrakte Steuerung der Kamera zur interaktiven Nutzung	COrbit-Camera-Controller
Extended-Structure-Model	Erweiterte Hierarchiestruktur mit Lage- und Ausmaßdaten	Model3D-ObjectManager
Selection-Model	Verwaltung und Manipulation ausgewählter Komponenten auf Basis strukturierter Referenzen zu dem ExtendedStructureModel	
Material-Manager	Verwaltung und Modifikation von Materialien, z. B. für Hervorhebungen	

Explosion-Animator	Animation der Übergänge zwischen Normal- und Explosionsansicht	-
Explosion-Generator	Berechnung der Positionsdaten für Explosionsdarstellungen	-

Tabelle 9: Verantwortlichkeiten der 3D-Viewer Komponenten und ihr Bezug zur Basis-Version (Eigene Darstellung)

Die Konfigurierbarkeit ist eine der zentralen Anforderungen an die 3D-Viewer-Komponente und wird zusammen mit den weiteren Schnittstellen im nächsten Abschnitt **Externe Schnittstellen und Konfigurierbarkeit** behandelt.

4.4.3.2 Externe Schnittstellen und Konfigurierbarkeit

Die 3D-Viewer-Komponente wird durch den Werkstatttester und dem Dokumentationsviewer genutzt. Grundsätzlich müssen für eine Darstellung die folgenden Inputs gegeben sein: das Strukturmodell der Szenen-3D-Modelle, eine Referenz zum Ort der verfügbaren Modelle und die Einstellungen. Wenn diese Inputs gegeben sind, kann die 3D-Viewer-Komponente zuerst das Strukturmodell auslesen, unter Berücksichtigung der Einstellungen die Modelle bestimmen, die geladen werden sollen, und diese dann anzeigen. Die Abbildung 17 zeigt die Komponentenansicht der 3D-Viewer Komponente, ergänzt um die Schnittstellen der Einstellungen.

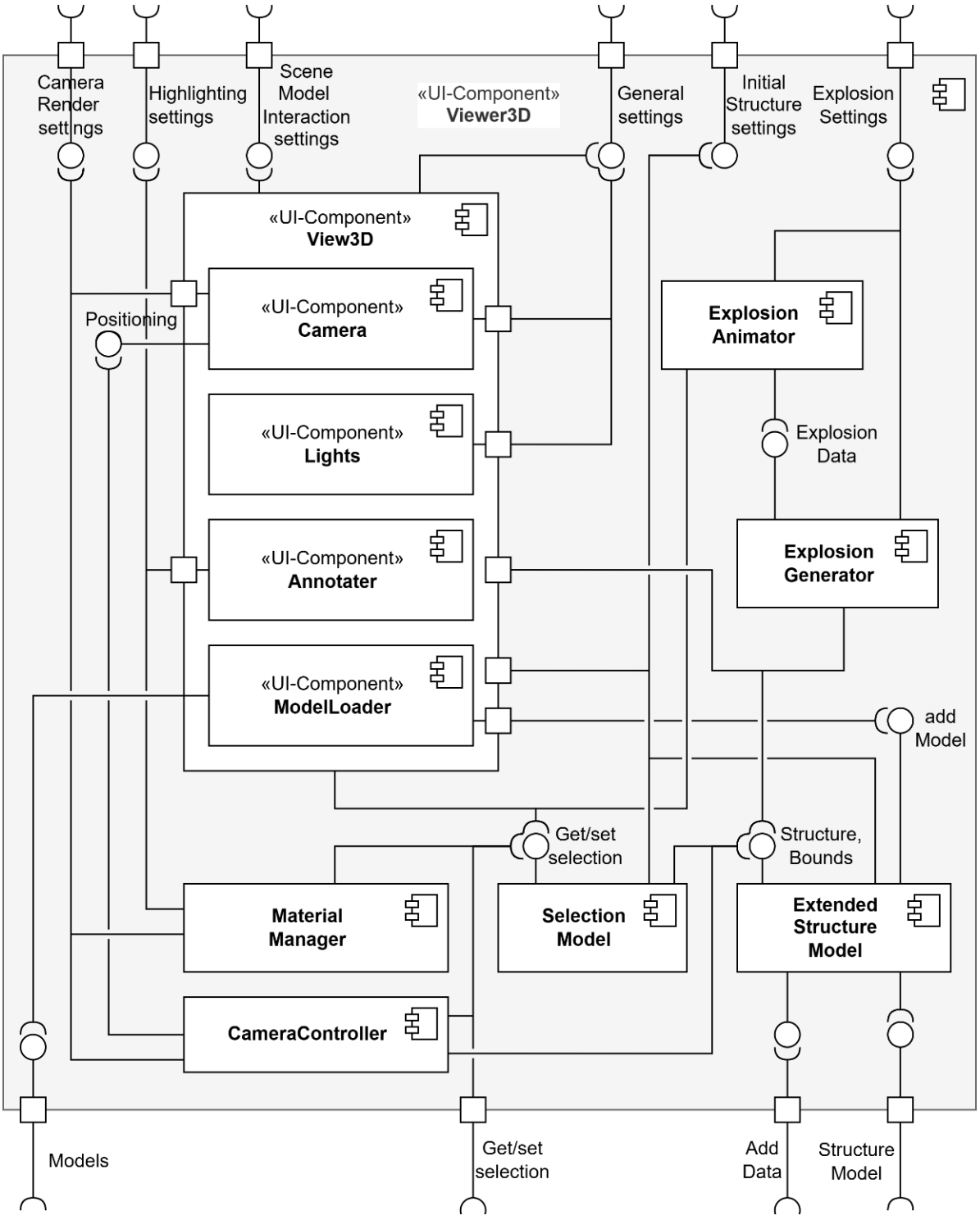


Abbildung 17: Komponentenansicht der 3D-Viewer Komponente (Eigene Darstellung auf Basis von Nandico (2023, S. 29)

Die Einstellungsmöglichkeiten sind in Tabelle 10 gruppiert dargestellt:

Einstellungsgruppe	Beschreibung	Anforderung
General Scene	Allgemeine Szenen-Eigenschaften (Hintergrund, Beleuchtung)	REQ_F_V_08_CONF
Camera Render	Kamerapositionierung und Darstellungsmodus (z. B. Illustration)	REQ_F_V_08_CONF, REQ_F_V_04_HIGHL, REQ_NF_V_04_HIGHL
Scene Model Interaction	Interaktion mit Objekten (z. B. Klickverhalten)	REQ_F_V_03_TRAV
Highlighting	Einstellungen zur Hervorhebung von Modellkomponenten, wie Darstellung der Annotationen, farbliche Hervorhebung, Ausblendung etc.	REQ_F_V_04_HIGHL, REQ_NF_V_04_HIGHL, REQ_F_V_05_ANNO
Explosion	Steuerung der Explosionsanimationen und Benutzerinteraktion	REQ_F_V_06_EXPL, REQ_F_V_02_STRUC
Initial Structure	Initial geladene Modelle, Initiale Vorauswahl innerhalb der Szene	REQ_F_V_08_CONF, REQ_F_V_03_TRAV, REQ_NF_V_01_PERF

Tabelle 10: Einstellungsgruppen mit Beschreibung und Anforderungsreferenz (Eigene Darstellung)

Allgemeine Einstellungen zur Szenendarstellung ergeben sich aus der grundsätzlichen Anforderung zur Konfiguration der Szene.

Die „Initial Structure“ Einstellungen betreffen unter anderem die Fälle, bei denen der Nutzer durch die Struktur traversieren kann. Die „Initial Structure“ Einstellungen erlauben dem Komponentenhersteller, eine initiale Auswahl innerhalb der traversierbaren Struktur der Szene festzulegen. Traversierbarkeit betrifft auch die Interaktion mit den Modellen in der Szene: Eine sehr intuitive und naive Traversierung ist ein schlichtes Anklicken von Komponenten.

Aus der Perspektive der Performanzverbesserung liegt ein verzögertes Laden von Modellen nahe, bis sie benötigt werden. Die Art und Weise welche Modelle initial geladen werden sollen, kann ebenso durch den Fahrzeughersteller als Teil der „Initial Structure“ Einstellungen getroffen werden.

Die Anforderung für die Hervorhebung von Komponenten zeigt sich grundsätzlich in zwei verschiedenen Bereichen der Einstellungen:

1. Die Einstellungen zur Modellhervorhebung selbst mit Veränderungen der Zielkomponenten und der restlichen Szeneninhalte.
2. Einstellungen zur allgemeinen Darstellung der Szene über die Konfiguration der Kamera und Render-Einstellungen. Anleitungen verwenden oft schematische Darstellungen im Stil von Zeichnungen, um die Hervorhebung von einzelnen Bereichen zu erleichtern. Eine solche allgemeine Darstellungsentscheidung, wie dem Renderstil ist deshalb verknüpft mit der Anforderung der Hervorhebung.

Die konkrete Ausarbeitung der Darstellungsmechanismen ist Teil des Design- und Implementierungsteils dieser Arbeit in 5.5.

Die Generation und Darstellung der Explosionsansicht kann durch den Nutzer modifiziert werden, um beispielsweise eine längere oder kürzere Animationsdauer festzulegen.

5 Design und Implementierung

In den folgenden Kapiteln wird die konkrete, technische Umsetzung des Systems beschrieben. Als technologische Grundlage dient das Qt-Framework insbesondere mit Qt Quick 3D. Die Entscheidung für Qt basiert auf mehreren Faktoren: Das Framework bietet eine plattformübergreifende Entwicklungsumgebung für Desktop- und mobile Systeme (z. B. Windows, Linux, macOS, Android). Qt stellt zudem mit seiner Trennung von UI-Definition (QML) und Anwendungslogik (C++) ein leistungsfähiges und einfacher wartbares Architekturmodell bereit. Die vorher entwickelte 3D-Viewer ist aus denselben Gründen bereits mit Qt implementiert worden und dient als Ausgangspunkt für die Weiterentwicklung der 3D-Viewer-Komponente.

Für die in den Kapiteln 5.2 bis 5.5 beschriebene Umsetzung werden in Kapitel 5.1 die relevanten Grundlagen des Qt-Frameworks vorgestellt. Anschließend folgt in 5.2 eine Übersicht der Systemstruktur auf Qt-Modulebene als Basis für die konkrete Implementierung. Die drei Hauptkomponenten des Systems – die Pipeline-Applikation Doc3DCreator (5.3), der Dokumentationsviewer (5.4) und der 3D-Viewer (5.5) werden jeweils detailliert beschrieben. Neben der allgemeinen Struktur werden dabei die zentralen Funktionen und wesentliche technische Herausforderungen erläutert.

5.1 Grundlagen des Qt-Frameworks

Im folgenden Kapitel werden Grundlagen und Eigenschaften des Qt-Frameworks beschrieben, die für die Entwicklung relevant sind. Basis für die Implementierung ist Qt 6.8.3, die zum Zeitpunkt der Entwicklung (01.12.2024) aktuelle Langzeitunterstützungsversion (LTS).

5.1.1 Modularer UI-Aufbau mit QML und C++

Für eine Trennung der Daten und der Darstellung der Daten bietet Qt grundsätzlich die Differenzierung der Applikation in C++-Code und in Code für die Nutzeroberfläche mit der QML-Sprache. QML ist eine deklarative Sprache, die sich an Javascript anlehnt und besonders für das Beschreiben von Oberflächenkomponenten und ihren Interaktionen geeignet ist. Eine einzelne Nutzeroberflächenkomponente entspricht dabei einer QML-Datei mit .qml-Endung. Beispielsweise kann ein neuer, benutzerdefinierter Knopf mit Drehfunktion in der Datei „knob.qml“ definiert werden. Andere Komponenten in demselben Projekt haben Zugriff auf diese Knopf-Komponente, indem sie ihn mit „import knob“ importieren. Es ist auch möglich, mit bestimmten Loader-Komponenten feste Pfade zu QML-Dateien anzugeben, die dann geladen und angezeigt werden.

Zur Unterstützung der Wiederverwendbarkeit bietet Qt die Möglichkeit, ebenso wie Code in Bibliotheken ausgelagert werden kann, QML-Code in sogenannte Module auszulagern. Wird als Build-System cmake verwendet, reicht der entsprechend parametrisierte Befehl „qt_add_qml_module“ mit wenig Zusatzarbeit als Teil der Modul-CMakeLists.txt aus, um ein wiederverwendbares, importierbares Modul zu definieren.

Qt-Komponenten können für die Kommunikation den sogenannten „Signal/Slot“-Mechanismus nutzen. Komponenten können Signale deklarieren, die sie bei Bedarf senden. Andere Komponenten können Slots deklarieren, die bei entsprechendem Signal ausgeführt werden. Der Signal-/Slot-Mechanismus kann als robustere Alternative zu klassischen Callbacks betrachtet werden. Slots sind reguläre Memberfunktionen, die mit dem Slots-Makro deklariert werden können. Ein Objekt muss nicht wissen, ob mit seinen Signalen Slots verbunden sind. Ebenso muss es nicht wissen, ob seine Slots mit Signalen verbunden sind.

Der Signal-Slot-Mechanismus ist standardmäßig Teil der Programmierung mit QML und wird im Hintergrund verwendet, wenn der Programmierer Code schreibt wie „onValueChanged: do xy“. Es ist möglich, QML-Komponenten ausschließlich auf Basis von QML-Code zu definieren. Für Komponenten mit komplexerer Logik ist es allerdings möglich und sinnvoller, sie über C++-Code zu definieren. Insbesondere bei komplexeren Datenmodellen, wie die Projektstruktur in der Datenaufbereitungspipeline, erlaubt das eine klare Trennung zwischen Modell und Darstellung.

Die Projektstruktur mit Vorbereitungsschritten und Teilschritten ist ein klassisches Beispiel einer Baum-Struktur. Basis für nicht-triviale Listen- und Baummodelle ist das QAbstractItemModel von Qt. Es bietet eine Reihe an Signalen und Slots, die von den Qt-gestellten UI-Komponenten integriert sind. Eine Subklasse von QAbstractItemModel muss nur einen Teil der Funktionen implementieren, um als Datenmodell in der Oberfläche genutzt zu werden.

5.1.2 Funktionsweise von 3D in Qt und Shadern

Qt ist ein Framework mit unter anderem dem Anspruch möglichst Plattform-unabhängig zu sein. Mit Qt Quick 3D stellt Qt die Möglichkeit bereit, plattformunabhängige 3D-Darstellungen in Qt Anwendungen zu integrieren (*Qt Quick 3D 6.8.3*, 2025). Qt erreicht das, indem 3D-Render-Aufgaben mit dem Qt Rendering Hardware Interface (RHI) für die Plattform-entsprechende Schicht darunter übersetzt werden (Abbildung 18). Qt-zugehörige Teile sind blau hinterlegt.

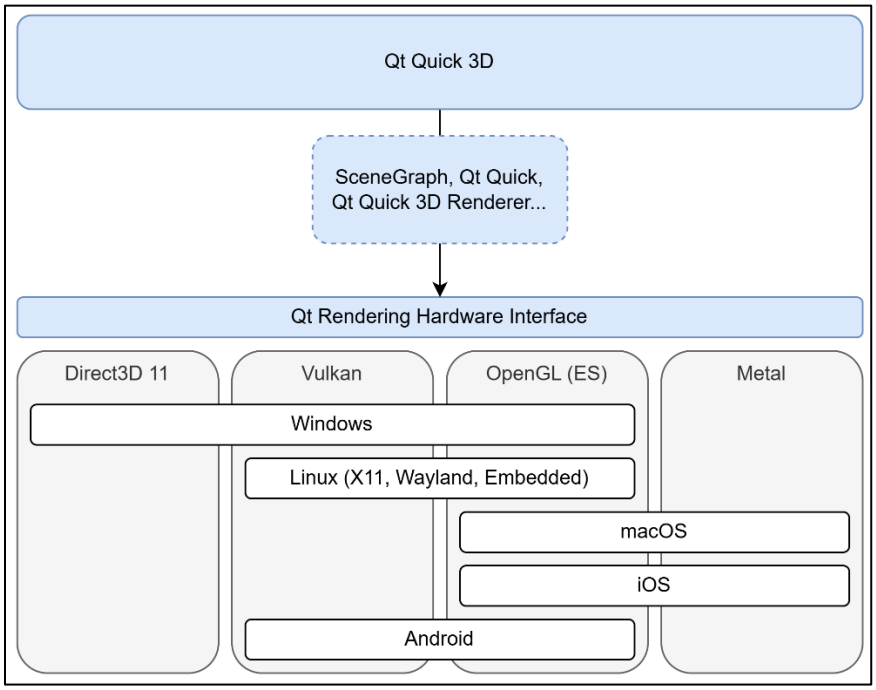


Abbildung 18: Qt Graphics Stack mit Qt Quick 3D und Qt Rendering Hardware Interface (Vereinfachte Darstellung nach *Qt Quick 3D Architecture* | *Qt Quick 3D 6.8.3*, 2025)

Qt Quick 3D ist zwar grundsätzlich eine 3D-Api mit hohem Abstraktionslevel, aber bietet auch die Möglichkeit, über Shader die Darstellungen zu beeinflussen. Shader sind nutzer-definierte Programme, die während des Render-Prozesses ausgeführt werden (*Shader - OpenGL Wiki*, 2025). Qt Quick 3D integriert nutzer-definiere Shader in zwei Bereichen: Shader für die 3D-Objekte selbst und Shader für die Gesamtszenendarstellung als sogenannte Effekte.

Shader für 3D-Objekte können die Form von Objekten und die visuellen Aspekte verändern, um beispielsweise bestimmte reale Materialien besser anzunähern. Shader für die Gesamtszenendarstellung erhalten als Input die Pixel der gerenderten Szene und können beispielsweise extra Effekte wie „Lens-Flare“ einblenden.

5.2 Systemübersicht auf Qt-Modulebene

Die zuvor beschriebenen Qt-Konzepte bilden die technische Grundlage für die Umsetzung der Systemarchitektur. In diesem Abschnitt wird gezeigt, wie die übergeordneten Systemkomponenten aus Abbildung 7 in Qt-Modulen innerhalb der Qt-Infrastruktur organisiert sind. Die Abbildung 19 zeigt die Module und verwendeten Bibliotheken. Blau hinterlegt sind externe Bibliotheken, die nicht im Rahmen dieser Arbeit entwickelt wurden. Blau schraffiert ist der Viewer3D, der wie bereits beschrieben auf einer vorherigen Version aufbaut. Die Pfeile entsprechen einer „benötigt“-Beziehung.

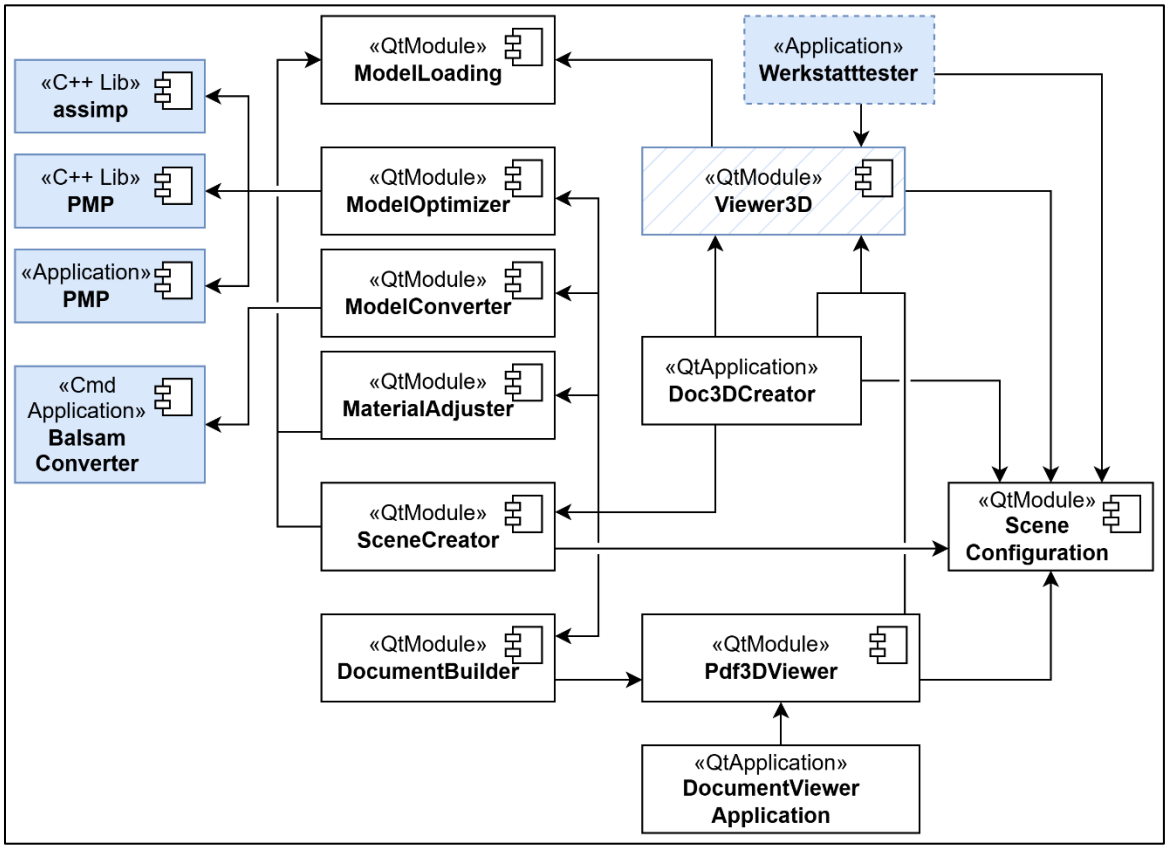


Abbildung 19: Module- und Bibliotheken-Übersicht mit externen Bibliotheken/Anwendungen (Eigene Darstellung)

Da viele der Komponenten 3D-Darstellungen nutzen, sind dafür häufig genutzte und hilfreiche Funktionen in die ModelLoading Komponente ausgelagert. Ebenso sind die Datentypen für die Strukturbeschreibung einer Szene und ihrer Einstellungen ausgelagert in die SceneConfiguration.

5.3 Doc3DCreator als Pipeline-Applikation

Der Doc3DCreator ist die Rahmenanwendung der Pipeline und hat wesentliche Verantwortung für das Management von Dokumentationsprojekten und ihrer Konfigurationen und ist damit verantwortlich für das Management die Abhängigkeiten der verschiedenen Modelle, Szenen und Dokumenten.

Dieses integrale Konzept und die dafür grundsätzlich notwendige Struktur zeigt die vereinfachte Abbildung 20. Eine detailliertere Darstellung ergänzt um die relevantesten Funktionen findet sich im Anhang als Abbildung 43.

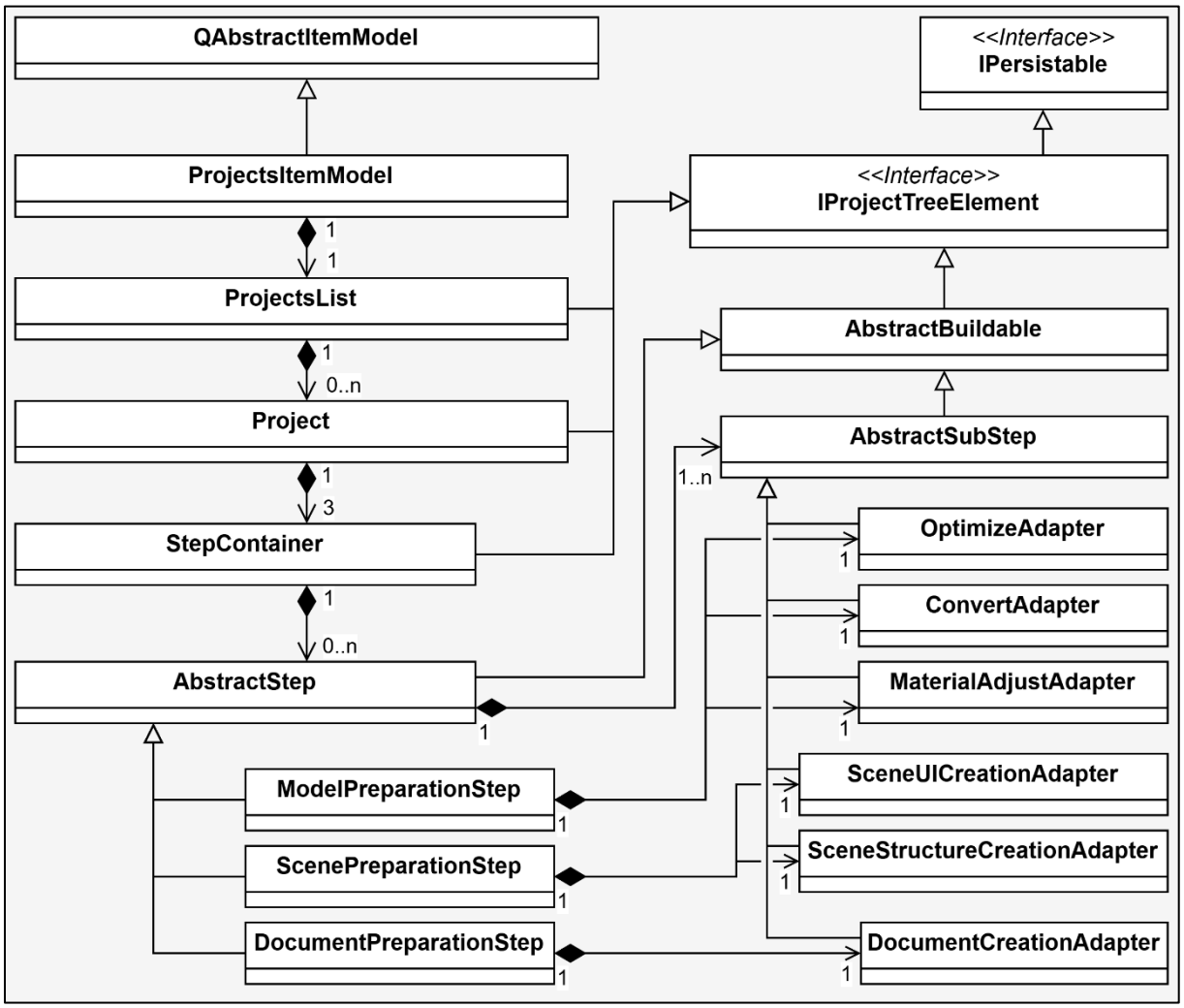


Abbildung 20: Vereinfachtes Klassendiagramm der Projektklassen des Doc3DCreators (Eigene Darstellung)

Ein Dokumentationsprojekt fasst eine Sammlung an Modellen, Szenen und Dokumenten zusammen. Der Doc3DCreator verwaltet die Projekte insgesamt über das ProjectsItemModel, was eine Bindung zur UI ermöglicht. Die einzelnen Projekte sind Teil der ProjectsList. Jedes Projekt hat genau drei differenzierte Bereiche: Modelle, Szenen und Dokumente. Ein „Modell“ entspricht hier der Konfiguration zur Aufbereitung eines Modells. Analog sind „Szenen“ und „Dokumente“ ebenso die entsprechenden Aufbereitungskonfigurationen. Ein StepContainer besteht aus einer beliebigen Anzahl an Modellen, Szenen oder Dokumenten.

Die Klassen ModelPreparationStep, ScenePreparationStep und DocumentPreparationStep sind sehr leichtgewichtige Klassen, die im Wesentlichen nur die Information beinhalten, welche spezifischen SubSteps in welcher Reihenfolge Teil von ihnen sind. Die Substeps entsprechen dann den tatsächlichen Pipeline-Schritten, wie sie bereits in der Architektur genannt wurden.

Mit dieser Form der Aufteilung können leicht weitere Pipeline Schritte ergänzt werden, in dem für sie nur ein Adapter erstellt werden muss und einer der AbstractStep implementierenden Klassen auf sie verweisen.

Im Folgenden werden zentrale Aspekte der Pipeline-Implementierung beschrieben. Kapitel 5.3.1 erläutert die Motivation und Realisierung des dateibasierten Ausführungsmodells und des Abhängigkeitsmanagements. 5.3.2 erläutert die Funktionsweise der modular aufgebauten Nutzeroberfläche.

Die nachfolgenden Kapitel von 5.3.3.1 bis 5.3.3.5 beschäftigen sich mit den Kernfunktionalitäten der jeweiligen Pipeline-Schritte.

5.3.1 Dateibasiertes Ausführungsmodell und Abhängigkeitsmanagement

Die Pipeline unterscheidet zwischen Konfigurationsdateien und den durch deren Anwendung erzeugten Artefakten. Die Pipeline orientiert sich dabei an der klassischen Softwareentwicklung mit Quellcode und Buildsystem.

Die Idee ist, dass die Projektstruktur mit den Modellen, Szenen und Dokumenten mit ihren Konfigurationsdateien analog dazu auf der gleichen Art und Weise auf der Festplatte strukturiert ist. Jeder Vorbereitungsschritt ist definiert über eine Konfigurationsdatei und erzeugt beim Ausführen einen Output-Ordner im Projekt-Build-Verzeichnis. Abhängigkeiten zwischen Schritten erfordern also eine Übergabe der Output-Pfade. Diese Struktur von Config und Build Trennung erlaubt beispielsweise eine einfache Versionierung der Konfigurationen, da sie als schlichte Text-Dateien in dem Projektverzeichnis vorliegen und in keinem Binärformat. Die Referenzierungen der Pfade sind relativ mit dem Projekt-Ordner als Basis aufgebaut, sodass es ohne Weiteres für den Fahrzeugkomponentenhersteller möglich ist, Projektverzeichnisse zu verschieben. Damit wird die Vorraussetzung für die Reproduzierbarkeit (REQ_NF_P_01_REPR) und die Möglichkeit für ein Update- und Änderungsmanagement über Versionierungstools erfüllt.

Zentrale Ausführungskomponente ist die ExecutionManager-Klasse des Doc3DCreators. Der Kernansatz ist, dass der Nutzer einzelne Schritte (Steps) oder Teilschritte (SubSteps) ausführen kann, indem die Steps ihre Ausführung bei dem ExecutionManager beantragen. Der übernimmt die Verantwortung für das Starten der dafür nötigen, vorherigen Schritte. Das basiert auf den Abhängigkeiten zwischen den Steps und dann den Abhängigkeiten der einzelnen SubSteps.

Die Abbildung 21 zeigt ein Beispiel der Prozesse für die Generation eines „EngineService-Guide“.

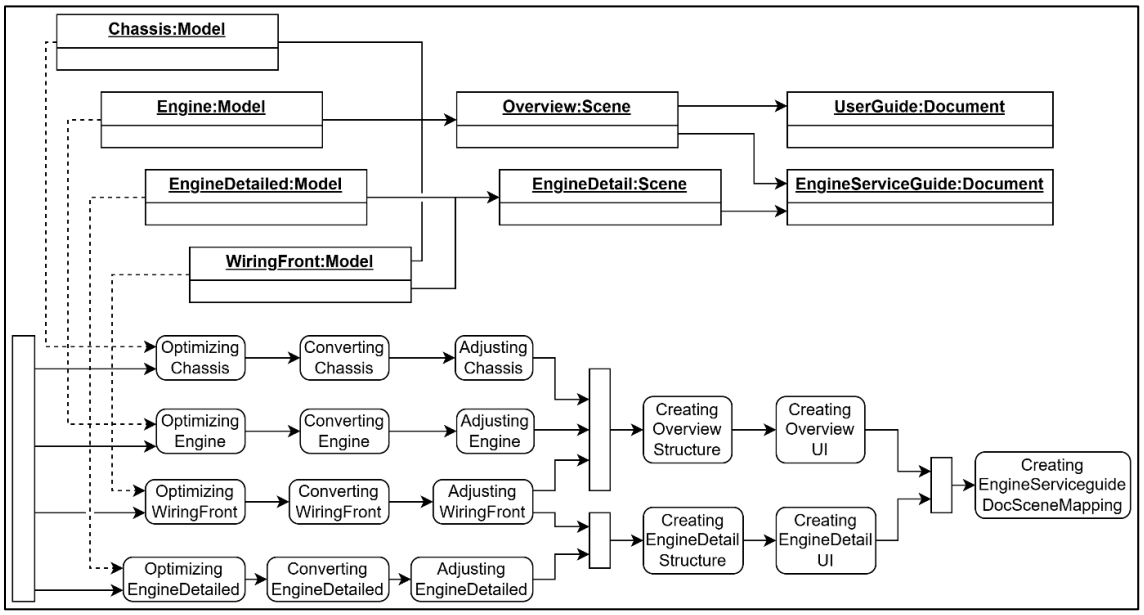


Abbildung 21: Prozessdarstellung der Pipeline für "EngineServiceGuide“ (Eigene Darstellung)

Für die Generation des EngineServiceGuides sind zwei Szenen notwendig: die Overview- und die EngineDetail-Szene. Beide der Szenen zeigen das Modell WiringFront, aber unterscheiden sich sonst in den darzustellenden Szenen. Der ExecutionManager übernimmt das Scheduling der einzelnen Schritte, sodass parallel stattfindende Prozesse auch parallel durchgeführt werden und sodass Schritte nur gestartet werden, wenn ihre vorherigen Schritte bereits ausgeführt wurden. Die tatsächliche Ausführung erfolgt über einen konfigurierbaren ThreadPool.

Die Abbildung 21 zeigt die Prozesse, die notwendig sind für die Generation eines Dokuments. Komponentenhersteller können die Ausführung gezielt bis zu einem bestimmten Verarbeitungsschritt einschränken – etwa, um eine Szene vorzubereiten, ohne das vollständige Dokument zu generieren. Verallgemeinert lässt sich die Ausführung für einen beliebigen AbstractStep oder AbstractSubStep starten, wie sie in der Abbildung 20 als Oberklassen gezeigt sind.

Wenn während der Ausführung weitere Schritte gestartet werden, überprüft der ExecutionManager welche Abhängigkeiten erfüllt werden müssen und startet nur die dafür notwendigen. Wenn ein Schritt fehlschlägt oder vom Nutzer abgebrochen wird, stoppt der ExecutionManager alle davon abhängigen Schritte. Sollte beispielsweise im Prozess aus Abbildung 21 die Aufbereitung des Modells „EngineDetailed“ gestoppt werden, wird im ersten Schritt bestimmt, welche vorher beantragten Ausführungen betroffen werden. Die betroffenen Schritte mit ihren Abhängigkeiten werden gestoppt, sofern sie sonst nicht benötigt werden.

Der ExecutionManager realisiert das durch Zählen der Referenzen von Schritten. Eine Anfrage zum Ausführen eines Steps führt dazu, dass für alle dafür benötigten Steps ein Referenzzähler hochgezählt wird. Zudem werden alle beteiligten Schritte als „wartend“ gesetzt und für den Nutzer ist kein Editieren mehr möglich. Der Executionmanager bestimmt dann auf Basis der gegebenenfalls bereits arbeitenden Schritte, welche weiteren Schritte gestartet werden können.

Wird die Ausführung eines Schritts beendet, wird geprüft, ob es sich um einen der Endschritte gehandelt hat. Wenn ja, wird der Referenzzähler aller vorherigen Schritte um eins verringert und die Schritte werden gegebenenfalls zum Editieren wieder freigegeben, wenn kein anderer Schritt von ihnen abhängt. Wenn nein, wird für die nächsten Schritte geprüft, ob ihre anderen Abhängigkeiten bereits erfüllt sind, und wenn das der Fall ist, wird ihre Ausführung über den ThreadPool gestartet.

5.3.2 UI-Aufbau

Die Abbildung 22 zeigt die Oberfläche des Doc3DCreators. Die Oberfläche setzt sich dabei aus der Projektnavigation auf der linken Seite und dem Konfigurationsfenster auf der rechten Seite zusammen. Im Konfigurationsbereich rechts wird je nach Auswahl in der Projektnavigation die Oberfläche des entsprechenden Steps geladen. Diese setzt sich wiederum aus den Oberflächen der einzelnen Substeps zusammen, wie sie in der Abbildung grün, orange und lila markiert sind. Die Oberfläche wird dynamisch zusammengesetzt: Jeder Einzeloberfläche gibt eine präferierte Höhe und Breite an den Doc3DCreator weiter, der den verfügbaren Bereich kontrolliert und vorgibt.

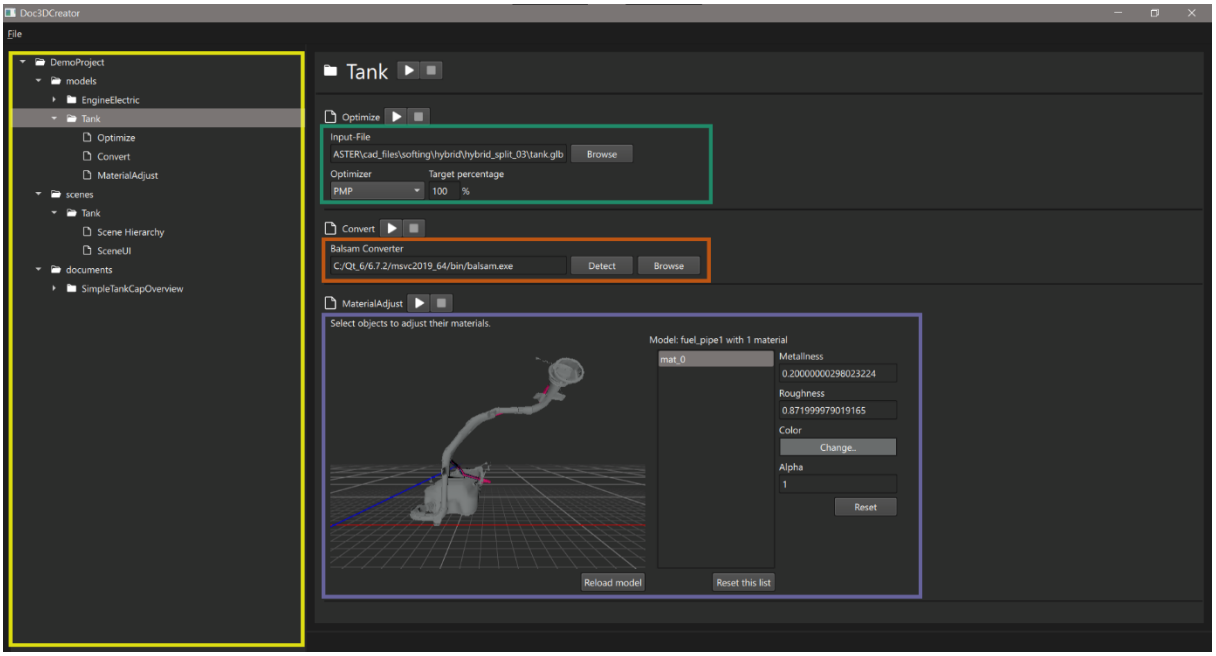


Abbildung 22: Screenshot aus dem Doc3DCreator mit farblicher Hervorhebung der UI-Bereiche: Projektnavigation mit Modellen, Szenen und Dokumenten (gelb), Step-Konfiguration (hier von einem Modell) mit Optimierung (grün), Konvertierung (orange) und Materialanpassung (lila) | (Eigene Darstellung)

5.3.3 Kernaspekte der Pipeline-Schritte

Im Folgenden wird für jeden der Pipelineschritte die wesentlichen Implementierungsherausforderungen und ihre Lösung erläutert.

5.3.3.1 Optimization: Optimieren von 3D-Modellen für Visualisierungszwecke

Das Ziel der Optimierung ist eine Vereinfachung der Geometrie der 3D-Daten mit minimalem Einfluss auf die visuelle Darstellung. Gleichzeitig benötigt der Konvertierungsschritt 3D-Daten, deren Geometrie nur über Dreiecke definiert ist. Eine Optimierung entspricht einer Verringerung der Anzahl der Dreiecke und hat für die weitere Verwendung die folgenden Vorteile:

1. Geringerer Speicher- und Netzwerkbedarf (REQ_F_P_05_EXP)
2. Schnellere Ladezeiten (REQ_NF_V_01_PERF)
3. Bessere Interaktivität (REQ_NF_V_01_PERF)

Auch können Modelle absichtlich stärker vereinfacht werden, um andere Teile der Szene besser zur Geltung kommen zu lassen. Je nach Szene hat der Nutzer gegebenenfalls unterschiedliche Detailanforderungen an die 3D-Daten: In der Motor-Detailansicht könnte die Chassis mit einem sehr reduzierten Detailgrad angezeigt werden, um den Motor zwar insgesamt räumlich einordnen zu können, aber, um gleichzeitig die Darstellung möglichst wenig mit nicht relevanten Details zu füllen.

Wichtige Anforderung ist hier, dass für die Konfiguration kein Expertenwissen notwendig ist (REQ_NF_P_04_NOEX). Das bedeutet, dass Fehlerschwellenwerte oder Ähnliches wenig sinnvoll sind. Diese Anforderung wird damit erfüllt, dass der Nutzer über einen singulären, prozentualen Optimierungsparameter den Grad der Anpassung kontrolliert. Ein Nutzer muss sich allerdings gegebenenfalls dann per Trial-and-Error dem gewünschten Ergebnis annähern.

Nach Xian et al. (2020) lässt sich das Feld der Vereinfachung von 3D-Modellen in grob zwei Bereiche teilen:

1. Geometrisch-zentrierte Algorithmen, die durch Anwenden von Vereinfachungsoperatoren iterativ die Geometrie des Modells vereinfachen, bis entweder eine minimale Anzahl an Dreiecken oder ein Schwellenwert an Abweichung von der ursprünglichen Geometrie erreicht wird.
2. Aussehens-zentrierte Algorithmen, die zusätzlich zu einer rein geometrischen Berechnung auch noch versuchen, das resultierende Aussehen stärker einzubeziehen.

Viele der Aussehens-zentrierten Algorithmen beziehen sich dabei unter anderem auf die Textur der Modelle. Eine Textur ist ein typischerweise 2D-Bild, was um das Modell gelegt wird, und erlaubt Farbe und damit auch Details hinzuzufügen. CAD-Modelle für die Fertigung besitzen solche Texturen in der Industrie nicht, da sie für die Fertigung unerheblich sind und nur das Aussehen modifizieren. Die einfachere Variante ist die Kontrolle des Aussehens über die Materialien.

Für die Implementierung und Integration sind viele Bibliotheken und Programme möglich. Folgende Aspekte sind grundsätzlich für die Verwendung in der Pipeline zu beachten.:

1. Die 3D-Daten besitzen keine Texturen und damit sind Algorithmen, die darauf aufbauen wenig sinnvoll.
2. Die 3D-Daten beinhalten viele Teil-Modelle. Diese müssen auch beibehalten werden. Algorithmen sollten die Gesamt-Struktur optimieren.

Der Algorithmus zur Optimierung eines einzelnen 3D-Objekts ist in seiner Grundform simpel:

1. Berechne Kosten für alle Möglichkeiten der Anwendungen des Vereinfachungsoperators
2. Wiederhole, bis Ziel erreicht:
    a. Wähle Operator mit niedrigsten Fehlerkosten
    b. Wende Operator an und berechne Update der Kosten

Für die Optimierung mehrerer Teilobjekte ist es naheliegend, die Fehlerkosten über alle Teilobjekte hinweg zu betrachten:

1. Wiederhole für jedes Teilobjekt: Berechne Kosten für alle Möglichkeiten der Anwendungen des Vereinfachungsoperators
2. Wiederhole, bis Ziel erreicht:
   a. Wähle Teilobjekt mit Operator mit niedrigsten Fehlerkosten
   b. Wende Operator an und berechne Update der Kosten

Die Entscheidung, welcher Vereinfachungsoperator, welche Kostenfunktion gewählt wird und das Verhalten bei unpassenden Inputdaten führen dann zu den unterschiedlichen Ergebnissen.

Eine vollständige Evaluation mit Prüfung aller möglicher Bibliotheken wurde aufgrund des hohen Aufwands nicht durchgeführt. Die Algorithmen zur Geometrievereinfachung fahren nach demselben, iterativen Muster fort, aber unterscheiden sich in den Eingriff- und Modifikationsmöglichkeiten, sowie der Performanz. Die Tabelle 11 zeigt eine vereinfachte Übersicht möglicher Bibliotheken oder Programme als Entscheidungshilfe.

	Lizenz	**Teile-übergreifende Optimierung**	**Position**	**Kosten**	**Eignung**
MeshLib	Kommerziell / Trial	Ja	Nein	Ja	Ungeeignet, nur eine Trial-Lizenz
CGAL	GPL 3	Ja	Ja	Ja	Geeignet, aber kommerziell nur mit bezahlter Lizenz
Libigl	MPL2	Ja	Ja	Ja	Geeignet, nur aufpassen bei Modifikation
VCGlib	GPL 3	Nein	Nein	Nein	Ungeeignet
MeshLab	GPL 3	-	-	-	Als Basis geeignet (Nutzer-Oberfläche für VCGlib)
Fast Quadric Mesh Simplification	MIT	Nein	Nein	Nein	Ungeeignet (limitierte Eingriffsmöglichkeiten) und höherer Integrationsaufwand
Mesh-Optimizer	MIT	Nein	Nein	Nein	Ungeeignet (limitierte Eingriffsmöglichkeiten) und höherer Integrationsaufwand

VTK	BSD-Stil	Nein	Nein	Nein	Ungeeignet (limitierte Eingriffsmöglichkeiten) und höherer Integrationsaufwand
PMP	MIT	Ja	Nein	Ja	Geeignet als Basis-Variante, leichtgewichtig
Blender	GPL 3	Nein	Nein	Nein	Als Basis geeignet, aber keine Teile-übergreifende Optimierung

Tabelle 11: Übersicht möglicher Bibliotheken und Programme zur Mesh-Vereinfachung (Eigene Darstellung)

Der Optimize-Step dient gleichzeitig als Konvertierung in ein Format, das für den nächsten Schritt der Konvertierung in das Bearbeitungsformat genutzt werden kann. Um möglichst die breite Menge der möglichen 3D-Formate abzugreifen, wird die Bibliothek assimp eingesetzt, die, wenn nötig, die Inputdaten konvertiert. Das kann sowohl bei der Eingabe als auch bei der Ausgabe der Optimierungskomponenten genutzt werden.

Die Implementierung erfolgt exemplarisch für eine Realisierung auf Basis von Fehlermatrizen nach Garland und Heckbert (1997) mit PMP (Sieger & Botsch, 2023) mit einer Einzelteil-übergreifenden Optimierung und einmal als einfache Verwendung eines weiteren Programms: hier die 3D-Bearbeitungssoftware Blender (Blender Online Community, 2018). Die Abbildung 23 zeigt den Aufbau der Optimizer-Komponente entsprechend der Architektur in Abbildung 12. Der PmpOptimizer implementiert entsprechend dem bereits genannten Teile-übergreifenden Algorithmus mithilfe der PMP-Bibliothek eine Vereinfachung der Geometrie. Der BlenderOptimizer führt die Blender-Applikation mit einem Konfigurationsskript aus. Das Skript selbst ist separiert von dem restlichen Code und unterliegt nach den Blender-Lizenzbedingungen auch der GPL-Lizenz.

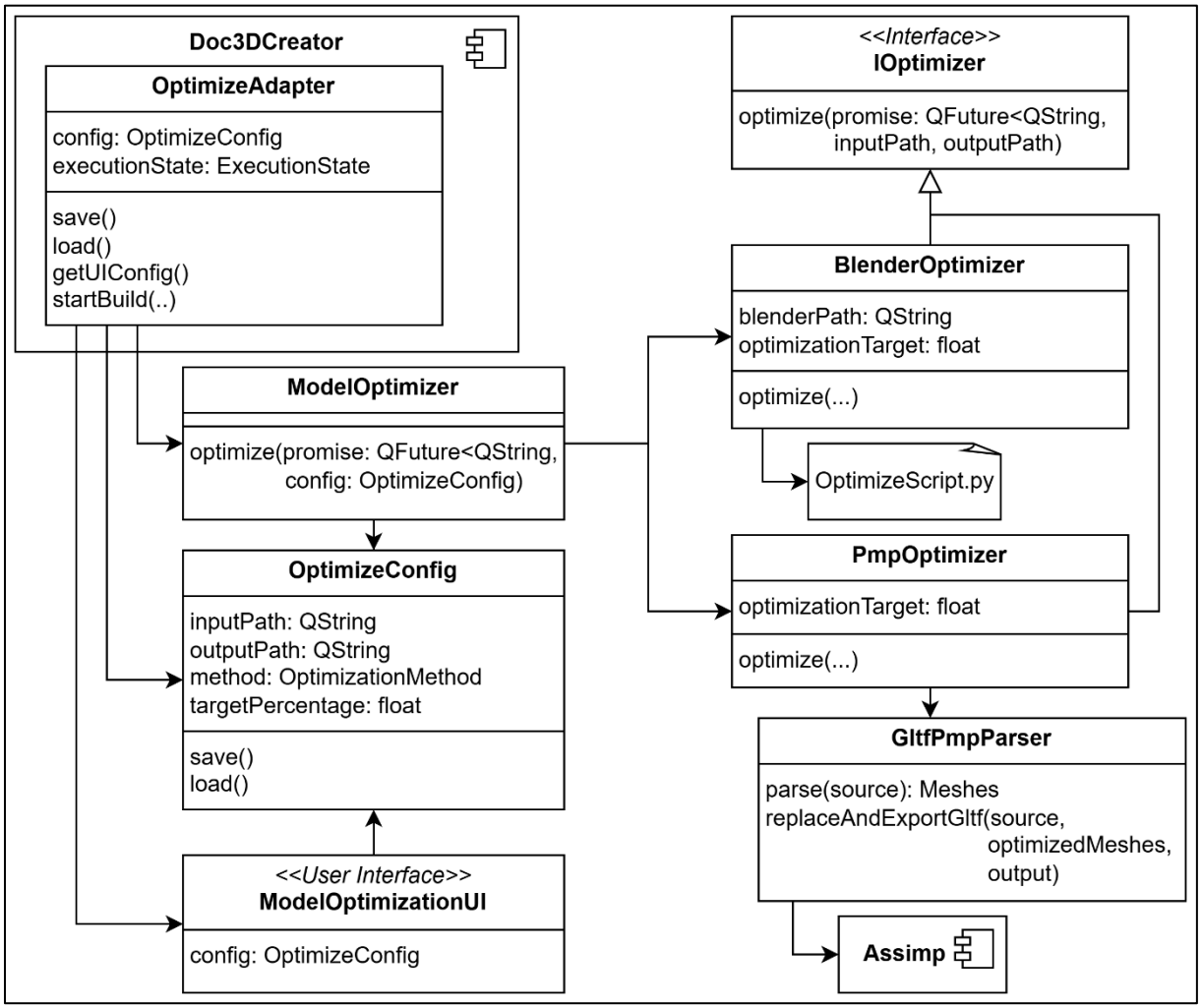


Abbildung 23: Vereinfachtes Klassendiagramm der Optimizer-Komponente (Eigene Darstellung)

5.3.3.2 Convert: Balsam-Converter für effizienteres Laufzeitformat

Ziel des Konvertierungsschritts ist die Konvertierung in ein optimiertes Laufzeitformat für einerseits den 3D-Viewer, andererseits aber auch für die weitere Verarbeitung und Verwendung in der Pipeline. Der Konvertierungsschritt unterscheidet sich von dem vorherigen Schritt der Optimierung, dass er keine direkte Veränderung der Geometrie vornimmt.

Die Entscheidung, Quick 3D zur Darstellung zu verwenden, ermöglicht eine weitgehend plattformunabhängige Entwicklung der 3D-Darstellungen. Gemäß der Qt-Dokumentation ist es allerdings nicht sinnvoll, direkt 3D-Dateiformate zur Laufzeit zu laden, da diese erst für Qt passend umgewandelt werden müssen. Qt bietet speziell für die Verwendung mit Quick 3D die Applikation „Balsam Asset Import Tool“ zur Konvertierung von 3D-Daten in ein Qt-spezifisches Laufzeit Format.

Das Balsam Asset Import Tool konvertiert die Daten in das Qt-eigene Format und separiert dabei Struktur der 3D-Datei und Materialien von der Geometrie. Konkret ergibt sich aus der Konvertierung:

1. Eine .qml-Datei, mit der hierarchischen Struktur und den Materialien
2. Ein Ordner mit binären .mesh-Dateien

Die ausgegebenen Struktur- und Materialdefinitionen sind leicht zu parsen und können in Folgeschritten der Pipeline bearbeitet werden.

Gleichzeitig optimiert das Balsam Asset Import Tool die Geometrie der 3D-Objekte für die Verwendung mit der eigenen Anzeigeschnittstelle RHI. Dabei wird zum Zeitpunkt dieser Arbeit unter anderem die Bibliothek assimp und die Bibliothek MeshOptimizer genutzt.

Das Balsam Asset Import Tool kann nur die Formate in Tabelle 12 lesen. Da später der Nutzer einzelnen Teilen des 3D-Modells Informationen zuweisen können soll, muss das Format zumindest eine grundsätzliche Differenzierung der einzelnen Bestandteile der 3D-Daten erlauben. Der Optimierungsschritt muss unabhängig von dem gewählten Optimierungsverfahren entweder Wavefront, COLLADA, FBX oder GLTF2 ausgeben. In der aktuellen Implementierung gibt sowohl der BlenderOptimizer als auch der PmpOptimizer das Format GLTF2 aus. Dieses Format erlaubt Materialien, hierarchische Teil-Strukturen und ist ein offener Standard.

Lesbare Formate für das Balsam Asset Import Tool	**Differenzierung der Einzelteile**
Wavefront (.obj)	Keine Hierarchie, aber Liste
COLLADA (.dae)	Hierarchie ist möglich
FBX (.fbx)	Hierarchie ist möglich
STL (.stl)	Keine Differenzierung
PLY (.ply)	Keine Differenzierung
GLTF2 (.gltf, .glb)	Hierarchie ist möglich

Tabelle 12: Format-Übersicht für das Balsam Asset Import Tool (Eigene Darstellung auf Basis der Formatspezifikationen)

5.3.3.3 MaterialAdjustment: Live-Anpassung von Materialien

Die Materialanpassung erlaubt die gezielte Umstellung von technisch orientierten Fertigungsmaterialien auf visuell optimierte Varianten zur realitätsnahen Darstellung. Die Definition der Darstellung über Materialien ist ein Merkmal des Physically Based Rendering (PBR). Grundziel des Physically Based Renderings ist es, möglichst realitätsnahe Darstellungen zu erhalten durch Nachbildung physikalischer Prinzipien der Lichtausbreitung und -interaktion.

Qt Quick 3D folgt dem Physically Based Rendering Ansatz und bietet zwei wesentliche Workflows zur Definition von Materialien:

1. Metalness/Roughness: Hauptparameter sind die „Metallheit“ eines Materials und die Rauheit.
2. Specular/Glossiness: Hauptparameter sind die Reflexivität und Farbe in einem Parameter und über den „Glanz“ die grundsätzliche Stärke davon.

Beide Workflows dienen dazu, das Material zu beschreiben, aus dem das 3D-Objekt besteht. Beide Workflows ermöglichen plausible Darstellungen, unterscheiden sich jedoch in Parametrisierung und Benutzerfreundlichkeit. Diverse Programme zur Darstellung von 3D-Daten erlauben die Definition über einen der beiden Workflows: CryEngine folgt Specular/Glossyness (Crytek, 2016), Unreal Engine folgt Metalness/Roughness (Epic Developer Community, 2025) und Blender erlaubt beide.

Nach Burley (2012, S. 12) ist es für die Anforderungen von Disney wichtiger, dass Materialparameter verständlich sind, als dass sie physikalisch exakt modelliert werden. Konkret entschied sich Burley (2012, S. 12) auf Basis der folgenden Prinzipien für einen Metalness/Roughness Workflow:

1. Es sollten eher intuitive als physikalische Parameter verwendet werden.
2. Es sollte so wenige Parameter wie möglich geben.
3. Die Parameter sollten in ihrem plausiblen Bereich zwischen null und eins liegen.
4. Es sollte erlaubt sein, Parameter über ihren plausiblen Bereich hinaus zu verschieben, wenn dies sinnvoll ist.
5. Alle Kombinationen von Parametern sollten so robust und plausibel wie möglich sein.

Für die Nutzung des MaterialAdjusters ist es ebenso wichtig, dass die Parameter intuitiv und leicht zu konfigurieren sind, da die Konfiguration nicht durch Experten erfolgt. Der MaterialAdjuster folgt aus diesem Grund ebenso dem Metalness/Roughness Workflow.

Die Abbildung 24 zeigt die Oberfläche des MaterialAdjustment Schritts. Der Nutzer wählt per Mausklick ein Objekt in der Szene aus. Die aktuell zugewiesenen Materialien werden daraufhin in der Detailansicht angezeigt. Änderungen an den Materialparametern wirken sich unmittelbar auf die Darstellung im Viewer aus.

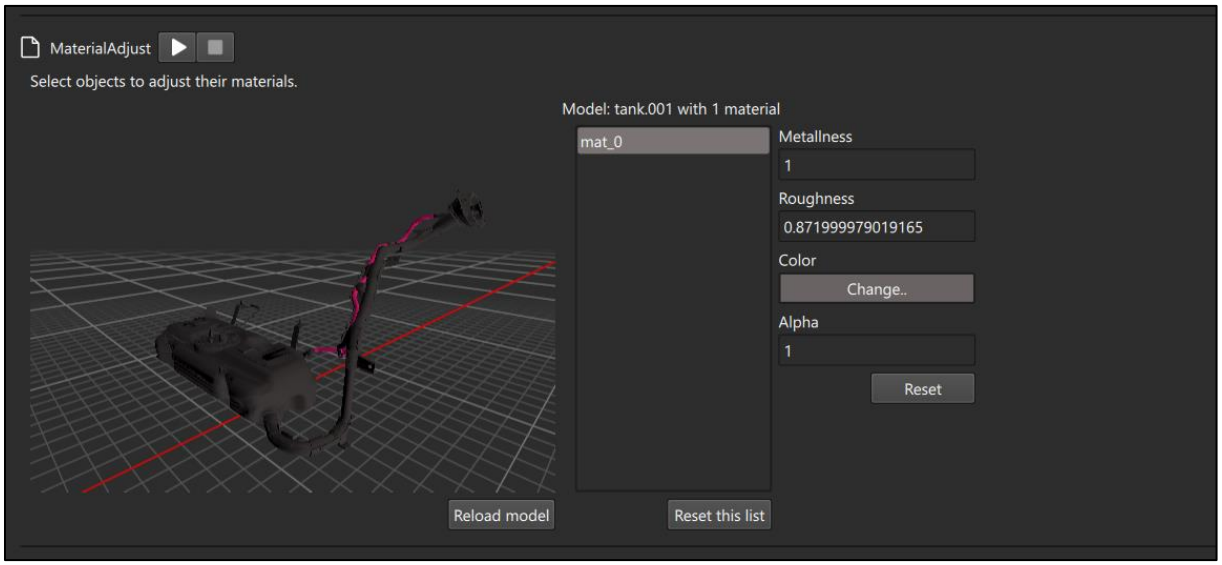


Abbildung 24: MaterialAdjust-Schritt aus Doc3DCreator (Eigene Darstellung)

Jede Material-Anpassung wird in der Step-Konfiguration persistiert. Beim Ausführen des MaterialAdjustment-Schritts werden die Materialien aus dem Input mit den angepassten Materialien ersetzt.

5.3.3.4 SceneConfiguration: Struktur und visuelle Konfiguration der Szene

Bereits als Teil der Architekturentscheidungen wurde die Szenen-Konfiguration in das Modul **SceneConfiguration** ausgelagert (siehe 4.4.1.2). Dieses Modul wird von allen Komponenten verwendet, die eine Szene für den 3D-Viewer konfigurieren wollen. Die SceneConfiguration bietet einerseits das Strukturmodell der Szene und andererseits die Einstellungen zu der Darstellung als vereinheitlichte Datenstruktur.

Die Szenestruktur setzt sich im Wesentlichen aus zwei Typen zusammen:

1. „Mesh“: Ein Mesh ist die Repräsentation von einem eindeutig referenzierten, nicht weiter differenzierbaren Teilobjekt eines 3D-Modells.
2. „Node“: Ein Gruppierungsobjekt, das mehrere Nodes oder Meshes enthält.

Die Abbildung 25 zeigt das vereinfachte Klassendiagramm der Szenen-Struktur. Im Unterschied zu Nandico (2023, S. 33–34) werden die referenzierten 3D-Modelle explizit als eigene Objekte definiert:

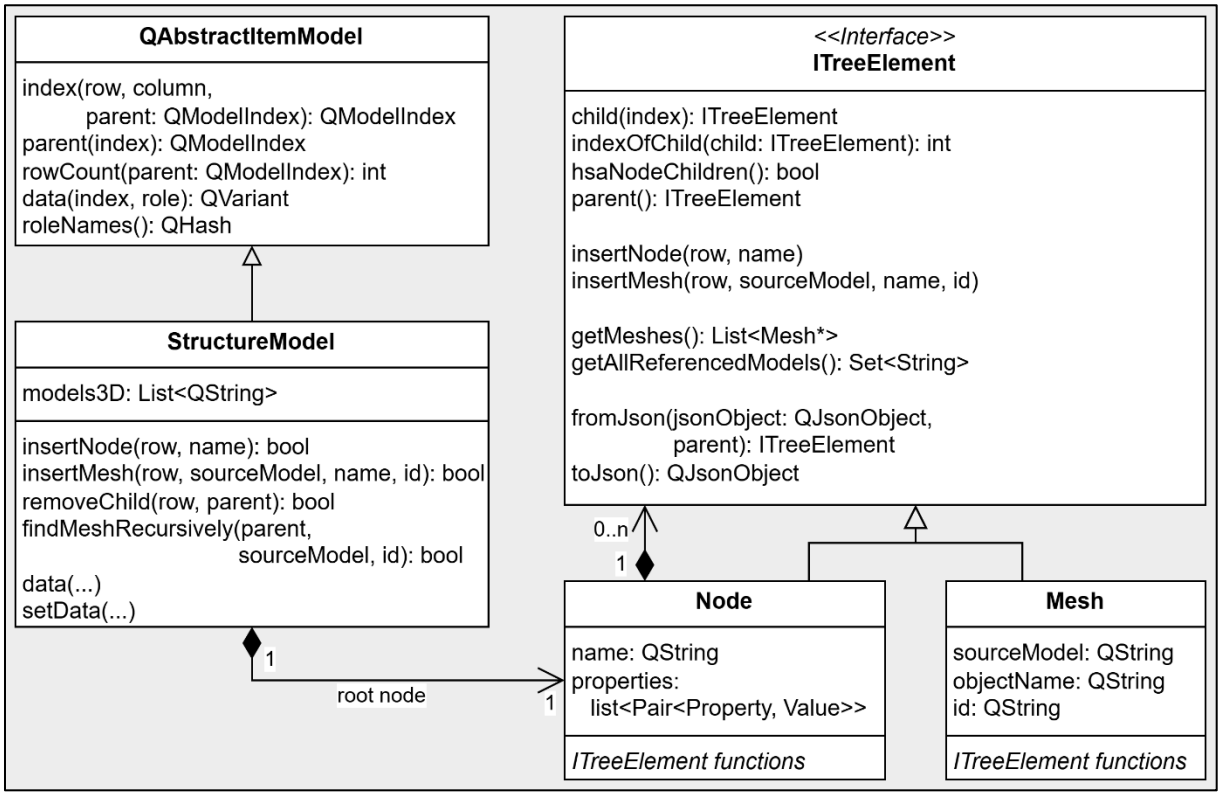


Abbildung 25: Vereinfachtes Klassendiagramm der Szenen-Struktur (Eigene Darstellung)

Mit dieser Struktur ist es möglich, die geforderte logische und/oder physische hierarchische Baugruppenstruktur (REQ_F_V_02_STRUC) umzusetzen. Teilebezogene Zusatzinformationen können als einfache Key-Value-Paare in den Node-Objekten hinterlegt werden (REQ_F_V_05_ANNO).

Die Klasse StructureModel ist die allgemeine Schnittstelle für das Lesen und Schreiben des Modells und stellt damit auch die Schnittstelle zur Oberfläche. Realisiert wird das im Qt-Stil für eine klassische Modell-/View-Architektur, wie sie im Kapitel 5.1 allgemein beschrieben ist.

Die Konfiguration der Darstellung erfolgt über die Klassen mit entsprechenden Attributen und für jede Gruppe an Einstellungen ein UI-Element. Die grundsätzlichen Einstellungsgruppen sind Einstellungen zu: General Scene, Camera Render, Scene Model Interaction, Highlighting, Explosion und zur Initial Structure (siehe Tabelle 10 für Beschreibungen). Für eine einfache Darstellung mit hervorgehobenen Objekten sind die Standardwerte ausreichend.

Eine zukünftige Implementierung für wiederkehrende Musterdarstellungen ist naheliegend, aber nicht im Rahmen dieser Arbeit implementiert. Besondere Einstellungsmöglichkeiten zu den Erweiterungen des 3D-Viewers sind Teil des Kapitels 5.5.

5.3.3.5 DocumentCreation: Dokumentations-Referenzierung mit Fokus auf PDF

Für die Realisierung der Referenzierung von Szenen und Dokumenten sind drei Aspekte von besonderem Interesse, um die Dokumentationsprozesse möglichst wenig abändern zu müssen und einfach zu halten (vgl. REQ_NF_DV_01_MOD und nach Szenario SC_07_DOCU):

1. Änderungsstabilität: Referenzen sollten nicht bei jeder Layout- oder Inhaltsänderung ungültig werden
2. Nicht-invasiv: Dokumentation selbst sollte unverändert bleiben
3. Einfache Handhabung: Referenzierung soll durch Nutzer mit minimalem Aufwand erfolgen können

Da PDF-Dateien einen zentralen Bestandteil der vorhandenen Dokumentationslandschaft darstellen (Gattullo et al., 2019, S. 2), konzentriert sich das entwickelte System auf deren Unterstützung. Eine Referenzierung sollte idealerweise relativ zu änderungsstabileren Elementen definiert werden, um das Kriterium der Änderungsstabilität zu erfüllen. Für die Anforderung REQ_F_DV_03_EXT kann der Nutzer zudem wählen, in welchem Modus die Szene geöffnet werden soll: direkt im PDF oder in einem separaten Fenster.

Zur Umsetzung wurden zwei Referenzierungsmechanismen implementiert:

1. Fix-Position: Der Nutzer legt die Seite, die Position auf der Seite und die Größe/Modus der anzuzeigenden Szene fest.
2. Text-Relative-Position: Der Nutzer legt Anker-Text, Offsets zu Anker und Größe/Modus der anzuzeigenden Szene fest.

Die Fix-Positionierung ist besonders einfach in der Handhabung, da der Nutzer schlicht festlegt, auf welcher Seite die Szene angezeigt werden soll und wie groß die Szene sein soll. Jegliche Änderung, die in dem Dokument auf einer Seite vor der Szene oder auf der Szenen-Seite selbst stattfindet, kann allerdings zu Problemen führen: Soll ein Bild von der 3D-Szene überlagert werden, kann es sein, dass sich die Szene jetzt auf einer Seite vor oder nach dem Bild befindet. Wird zum Beispiel eine Szene zum Motor auf die Seite 10 mittig positioniert, um genau ein statisches Bild des Motors zu überlagern, kann es sein, dass nach einem Update der Dokumentation dieses Bild auf Seite 11 steht. Die Szene wäre allerdings noch auf Seite 10. Der Nutzer bekommt die Szene am falschen Ort angezeigt.

Die Text-Relative-Position arbeitet stattdessen mit einem nutzergegebenen Anker-Text. Das Dokument wird nach diesem Text durchsucht und die Szene wird relativ zu der Text-Position platziert. Der Nutzer kann zusätzlich noch Offsets angeben, um relativ zu dem Anker-Text die Szene zu verschieben. Probleme ergeben sich bei nicht eindeutigen oder sich leicht ändernden Textankern, was zu unerwartetem Verhalten für den Nutzer führen kann.

Beide Referenzierungsmechanismen benötigen keine Veränderung der ursprünglichen Dokumentation. Die textrelative Referenzierung kann vereinfacht werden, indem beispielsweise Bilder mit eindeutigen Textkennungen beschriftet werden.

Grundsätzlich wird beim Ausführen des DocumentCreation-Schritts jede Referenz in eine konkrete Positionsangabe transformiert. Diese Transformation reduziert die Komplexität zur Laufzeit und ermöglicht eine einheitliche Weiterverarbeitung im DocumentViewer, der ausschließlich mit fixen Mappings arbeitet.

Andere Dokumenttypen als PDF können andere spezialisierte Referenzierungsmechanismen nutzen: Wird unter anderem HTML verwendet, könnten Referenzen genutzt werden, die über einen Pfad eine Zielposition definieren. So kann über die Abfragesprache XPath ein spezifisches HTML-Element adressiert werden, was dann mit einer Szene überlagert werden kann.

5.4 Dokumentationsviewer

Der Dokumentationsviewer umfasst zusätzlich zum 3D-Viewer Komponenten für die Anzeige der Dokumentation und für das Laden der 3D-Szenen. Die allgemeine Architektur und Aufteilung sind bereits in 4.4.2 beschrieben und betonen insbesondere die Trennung von der Dokumentationsanzeige und der 3D-Szene.

Qt ermöglicht eine einfache Überlagerung von UI-Elementen, sodass die bisherige Darstellung mit Fokus auf PDF ohne größere Probleme auch für weitere Dokumentationsarten erweitert werden kann, vorausgesetzt, dass diese mit QML-Komponenten gerendert werden können.

Die Positionen der Szenen innerhalb der PDF-Dokumente werden beim Exportieren der Dokument-Szenen Pakete fixiert (vgl. DocumentCreation: Dokumentations-Referenzierung mit Fokus auf PDF). Szenen können wahlweise innerhalb des Dokuments oder in einem separaten Fenster dargestellt werden (siehe REQ_F_DV_03_EXT). Die Standarddarstellung wird durch den Doc3DCreator konfiguriert.

Sollte der 3D-Viewer oder die Konfiguration fehlerhaft sein, bleibt durch die modulare Trennung zumindest die Anzeige der ursprünglichen Dokumentation erhalten.

Abbildung 26 zeigt eine Beispieldarstellung, wie ein einfaches Handout für einen Elektromotor ausschauen könnte mit der geladenen 3D-Darstellung. Ein kompletter Durchlauf der Datenaufbereitungspipeline mit Konfiguration von Modell, Szene und Dokument befindet sich in 5.6.

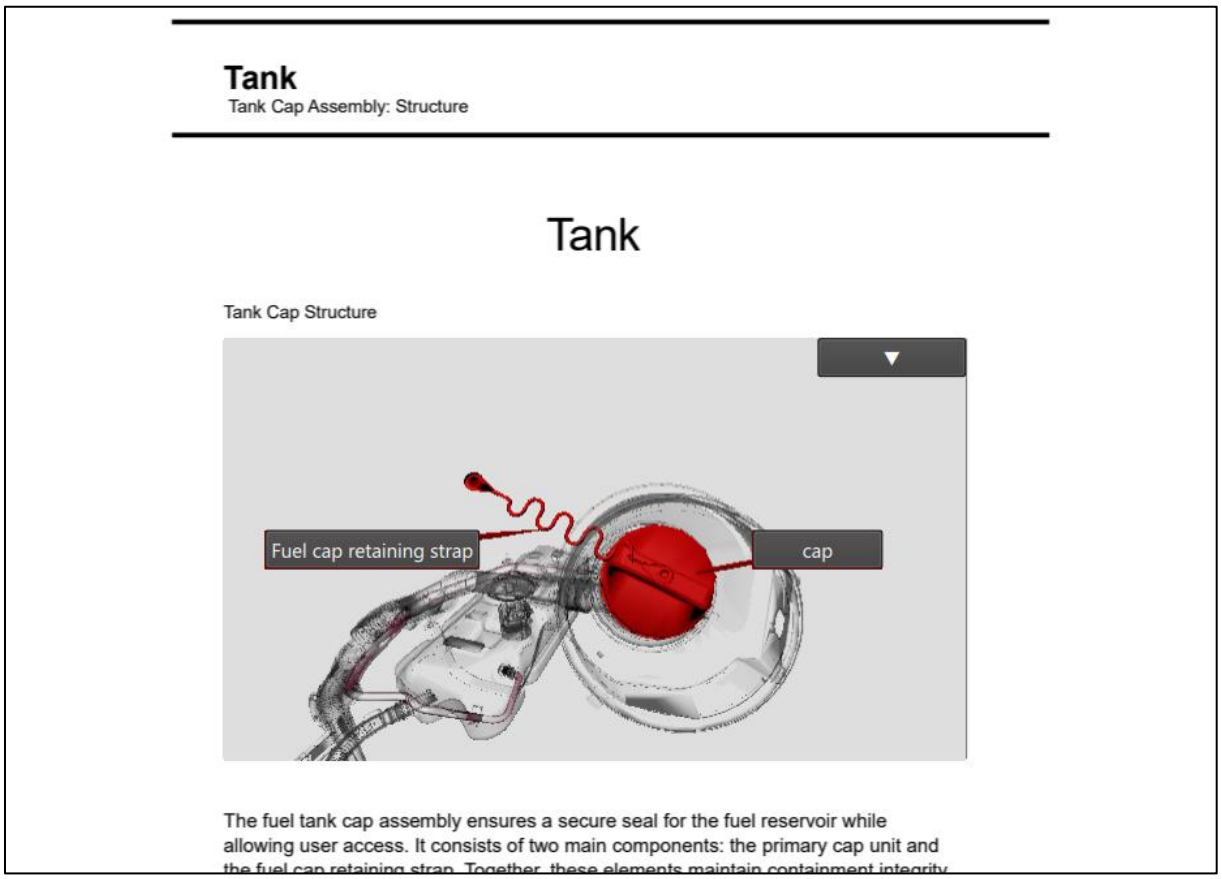


Abbildung 26: Screenshot aus dem Dokumentationsviewer nach dem Laden der 3D-Ansicht (rechts) (Eigene Darstellung)

Beim Seitenwechsel werden geladene 3D-Szenen standardmäßig entladen, um nicht unnötig Kapazitäten in Anspruch zu nehmen, für Szenen, die gegebenenfalls bereits nicht mehr gebraucht werden. Extern geöffnete Szenen bleiben grundsätzlich geöffnet, bis der Nutzer sie manuell schließt.

5.5 3D-Viewer

Der zuvor entwickelte 3D-Viewer (Nandico, 2023) war grundsätzlich bereits in der Lage, Teilkomponenten hervorzuheben. Allerdings sind die Hervorhebungs- und Beschriftungsmechanismen nicht allgemeingültig genug und können verbessert werden. Teil der Entwicklung ist damit eine neue Positionierung der Beschriftungen in der Szene und eine verbesserte Berechnung der Transparenz für ausgeblendete Komponenten.

Um die Wartungs- und Servicedokumentation adäquat zu unterstützen, sollten die bestehenden Darstellungsvarianten als Teil der Bereitstellung zielgerichteter Visualisierungsmechanismen abgedeckt werden (REQ_NF_V_04_HIGHL). Schwarz-weiße Konturzeichnungen stellen dabei einen relevanten Anteil in Aufbau-/Montageanleitungen dar (Mohr et al., 2015, S. 6). Um diese Darstellungen ebenso generieren zu können, ist ein vereinfachter Darstellungsmodus im Stil von Schwarz-Weiß-Konturzeichnungen implementiert.

Ebenso sind Explosionsdarstellungen „populär“ in der Dokumentation (Mohr et al., 2015, S. 3). Dementsprechend wird der Funktionsumfang des 3D-Viewer mit der Möglichkeit ergänzt, automatische Explosionsansichten zu generieren nach der Anforderung REQ_F_V_06_EXPL.

Die Realisierung dieser Erweiterungen wird in den folgenden Kapiteln behandelt.

5.5.1 Label- und Pfeil-Positionierung

Eine wichtige Aufgabe des 3D-Viewers ist die Beschriftung von Komponenten. Die Problematik liegt dabei in der automatischen Bestimmung der Positionen der Beschriftungen. Das Beschriften von Visualisierungen ist ein länger bekanntes Problem mit einer Vielzahl an Ansätzen und Algorithmen mit verschiedenen Vor- und Nachteilen (Bekos et al., 2021). Die Aufgabe kann konkret beschrieben werden als „eine komplexe Aufgabe, die, wenn sie automatisch durchgeführt wird, hoch entwickelte algorithmische Techniken erfordert und eine gute Balance zwischen möglicherweise widersprüchlichen Platzierungskriterien“ (Bekos et al., 2021, S. 9).

5.5.1.1 Klassifikation von Beschriftungsmodellen

Die gesamte, beschriftete Darstellung sollte allgemeine Anforderungen zur Verständlichkeit erfüllen. Das betrifft zwei Bereiche: einerseits die Art und der Verlauf der Verbindungslinien oder Pfeile von Beschriftung bis hin zu dem beschrifteten Zielelement und anderseits die Darstellung und insbesondere die Platzierung der Beschriftungen. Für die Differenzierung verschiedener Beschriftungsmodelle unterscheiden Bekos et al. (2021, S. 4–6) auf Basis dreier wesentlicher Aspekte:

1. Erlaubte Label-Positionen
2. Pfeil-Stil
3. Zeitliche-Dimension

Die erlaubten Label-Positionen entsprechen der Frage, wo grundsätzlich Label platziert werden dürfen. Eine Unterscheidung ist danach möglich, ob Label innerhalb oder außerhalb der visuellen Objekte platziert werden (Hartmann et al., 2005). Diese initiale Unterscheidung ist allerdings nicht allgemeingültig, da es auch Label-Varianten gibt, die zwar ihre Zielkomponenten, die Fokus-Region, nicht überlappen, aber dafür keine besondere Rücksicht auf den Rest des Modells nehmen (Bruckner & Groller, 2005, Madsen et al., 2016). Bekos et al. (2021) betrachten diese Varianten ebenso als eine Positionierung „außerhalb“ einer bestimmten Region, da die Algorithmen dafür oft trotzdem anwendbar sind. Sie unterscheiden grundsätzlich die folgenden Möglichkeiten:

1. Kontur-Platzierung: Die Platzierung erfolgt entlang der Kontur der Darstellung.
2. Platzierung nach Begrenzung: Die Platzierung erfolgt entlang klar definierter Begrenzungen der sichtbaren Szene, bei dem die Label das umschließende Rechteck der Darstellung an festgelegten Kanten berühren muss.
3. Exzentrische / Fokus-Region-Platzierung: Die Platzierung erfolgt nach den gleichen Mustern wie Punkt 1 und 2, nur dass anstatt der gesamten Darstellung lediglich eine Fokus-Region mit den beschrifteten Elementen umgeben wird.

Die verbindenden Elemente zwischen Beschriftung und dem beschrifteten Element können ebenso auf verschiedene Art und Weise umgesetzt werden. Bekos et al. (2021, S. 5–6) unterscheiden als wesentliches Kriterium die unterschiedliche Form der Verbindungslinien: Die Linien sind simple Geraden, gebogen oder auch aus Segmenten zusammengesetzt.

Die dritte wesentliche Eigenschaft ist die Eignung der Platzierungen für Veränderungen der Darstellung über einen Zeitraum hinweg statt nur einer statischen Platzierung. Wird das Modell beispielsweise gedreht, ändern sich die optimalen Positionen der Beschriftungen.

5.5.1.2 Anforderungen und Bewertungskriterien für Beschriftungsmodelle

Bekos et al. (2021, S. 8–9) haben in ihrer Arbeit konkrete Kriterien für statische Darstellungen identifiziert. Speziell für dynamische Darstellungen kommen zusätzlich die ergänzenden Kriterien C10 und C11 hinzu für die Eignung der Platzierung für interaktive Systeme (s. Tabelle 13):

C1	Die Verbindungslinien sind kurz.
C2	Die Anzahl der Überlappungen der Verbindungslinien und die Anzahl der Überlappungen der Label sind niedrig.
C3	Die Label nähern die Form der Darstellung an.
C4	Die Label sind gleichmäßig verteilt.
C5	Die Richtungen einzelner Liniensegmente entsprechen bevorzugten Richtungen.
C6	Die Verbindungslinien haben möglichst wenig Knicke.
C7	Es ist ausreichend Platz zwischen Verbindungslinien.
C8	Label bestehen möglichst aus einer einzelnen Zeile Text.
C9	Label, die semantisch zusammengehören, sind gruppiert.
C10	Flackern und Springen von Labeln ist gering.
C11	Beschriftungen enthalten Tiefeninformation.

Tabelle 13: Kriterien für Beschriftungsmodelle nach Bekos et al. (2021, S. 8–9)

Für den vorherigen 3D-Viewer wurde ein Algorithmus implementiert, der die Label auf einer Kugel um das Fahrzeug als Teil der 3D-Szene positioniert. Diese Art und Weise der Positionierung ist damit nicht Teil der externen Beschriftungsmodelle, da die Beschriftungen sich unter anderem hinter dem Objekt befinden können und sich damit in der 2D-Darstellung am Bildschirm überlagern. Die Verbindungslinien sind einfache, gerade Linien von den Beschriftungen zu den Mittelpunkten der beschrifteten Objekte. Die Darstellungen bleiben über den zeitlichen Verlauf hinweg stabil, da die Positionen feste Bestandteile der 3D-Szene sind.

Dieses Beschriftungsmodell ist aus zwei Gründen problematisch:

1. Die Beschriftungen überlappen sich leicht, insbesondere bei räumlich benachbarten Zielen.
2. Die Beschriftungen sind nicht dynamisch für Zoomen nutzbar.
3. Die Beschriftungen sind schlecht erkennbar, wenn das Fahrzeug oder die Komponente sie verdeckt.

Ziel dieser Arbeit ist die sinnvolle Hervorhebung von Teilen und Anzeige von Zusatzinformationen. Um diesen Anforderungen besser gerecht zu werden, ist es das Ziel, eine neue Form der Beschriftungen zu implementieren. Um den Aufwand in einem vertretbaren Rahmen zu halten, wurde auf besonders komplexe Verfahren verzichtet.

5.5.1.3 Implementierung eines vereinfachten Flush-Layout-Algorithmus

Als Flush-Layout wird eine Anordnung entlang fest definierter Kanten verstanden (Ali et al., 2005, S. 5–6). Der implementierte, vereinfachte Flush-Layout-Algorithmus positioniert die Label auf zwei senkrechten Linien links und rechts in der 3D-Szene. Die Label sind grundsätzlich 2D und passen sich nicht der Tiefe an. Sie werden auf den senkrechten Linien so angeordnet, dass sie überlappungsfrei sind und insbesondere ihre Höhe beachtet wird. Jedes Label kann mit einem Pfeil mit dem beschrifteten Teil verbunden werden. Diese Pfeile sind Teil der 3D-Szene und werden vom Licht und den Objekten in der Szene beeinflusst. Die Abbildung 27 zeigt ein statisches Ergebnis dieses Algorithmus mit Beschriftungen links und rechts auf den senkrechten Linien:

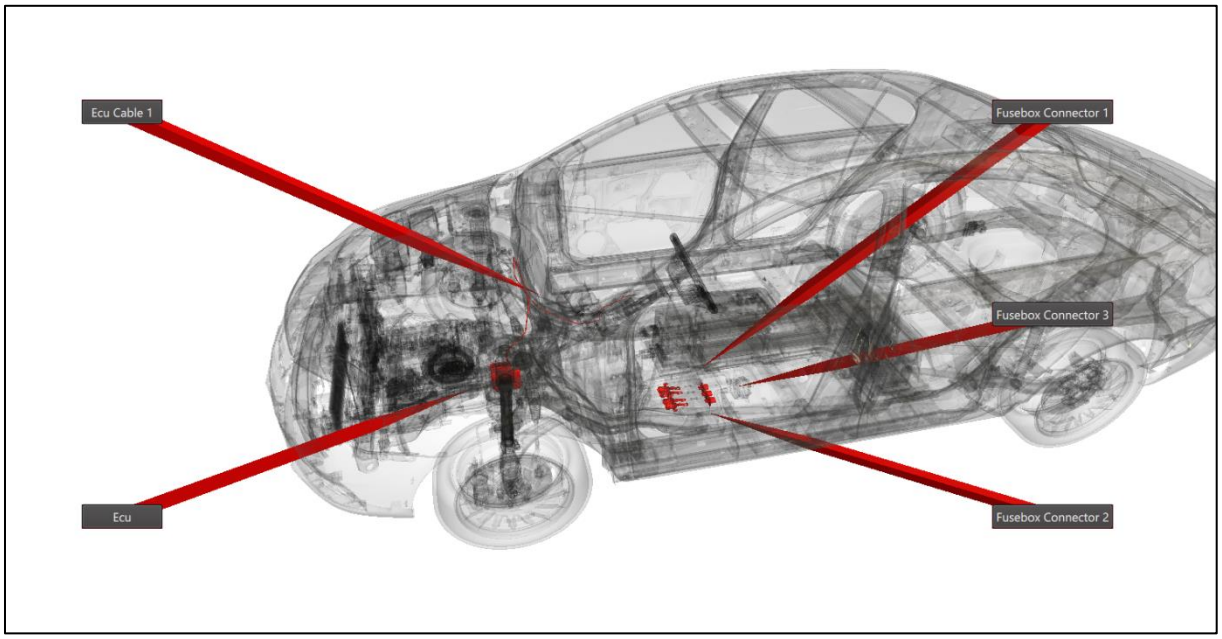


Abbildung 27: Hybrid-Fahrzeug im 3D-Viewer mit Beschriftungen auf senkrechten Linien links und rechts (Eigene Darstellung)

Die Tabelle 14 enthält die Übersicht, inwiefern der vereinfachte Flush-Layout-Algorithmus die Kriterien C1-C11 erfüllt:

	Kriterium	**Erfüllung durch vereinfachten Flush-Layout-Algorithmus**
C1	Kurze Verbindungslinien	Ja, aber nachrangig zu C2 und C4
C2	Wenig Überlappungen (Linien und Label)	Ja (wenn nicht zu viele Label angezeigt werden)
C3	Label folgen der Form der Darstellung	Nein
C4	Gleichmäßige Label-Verteilung	Ja

C5	Segmente folgen präferierten Richtungen	Keine Segmente
C6	Wenig Knicke	Ja
C7	Platz zwischen den Linien	Entsprechend der Verteilung C4
C8	Einzeiler-Label	Nein (aus gegebenen Anforderungen)
C9	Semantische Gruppen sind zusammengehörig	Nein
C10	Geringes Flackern und Springen	Geringes Flackern, aber Springen (Reihenfolge stabil bei gleicher Labelmenge, instabil bei Änderung der Anzahl)
C11	Tiefeninformationen im Beschriftungsmodell	Label selbst sind 2D, aber die Verbindungslinien (Pfeile) sind Teil der 3D-Szene und geben Tiefenhinweise

Tabelle 14: Erfüllung der Platzierungskriterien durch den vereinfachten Flush-Layout-Algorithmus (Eigene Darstellung)

Der Algorithmus orientiert sich an dem Flush-Left-Right-Algorithmus nach Ali et al. (2005, S. 5–6). Er weicht insofern ab, als dass speziell die Optimierung der Positionen der Label zusätzlich auf einer einfachen Heuristik basiert: Wenn die Höhe der Label auf einer Seite weniger als 50 % der verfügbaren Höhe ist, werden die Label gleichmäßig auf den mittleren 50 % der Höhe verteilt. Wenn sie insgesamt höher sind, werden sie direkt übereinander angeordnet. Damit wird auf der einen Seite ein grundsätzlicher Abstand der Label gesichert, mit gleichmäßiger Nutzung des verfügbaren Raums, aber auch andererseits eine nicht zu starke Abweichung von der Wunsch-Label-Position. Der Ablauf des Algorithmus ist wie folgt:

1. Erstelle die Datenstrukturen und die initialen 3D-Positionen der 3D-Zielkomponenten
2. Bestimme die 2D-Positionen der 3D-Zielkomponenten
3. Bestimme die bevorzugten Positionen der 2D-Label
4. Bestimme die Label- und Linien-überlappungsfreien 2D-Positionen der Label
5. Bestimme die 3D-Positionen der Label

Je nachdem, welche Daten sich ändern, können alte Ergebnisse wiederverwendet werden: Ändert sich die grundsätzliche Menge der darzustellenden Objekte, wird alles neu berechnet. Ändert sich die Größe der Anzeige und die Position der Kamera, werden die Ergebnisse von Schritt 1 weiter hergenommen. Ändert sich die Position der Linien, auf denen die Label positioniert werden, werden die Ergebnisse von Schritt 1 und 2 weiterverwendet. Ändert sich die Größe eines Labels, weil zusätzliche Daten bereitgestellt werden, wird erst mit dem 4. Schritt fortgefahren. Damit kann grundsätzlich die Performanz verbessert werden und insbesondere die initialen Datenstrukturen werden seltener allokiert.

Die überlappungsfreie Positionierung in Schritt 4 wird anhand der beschriebenen Heuristik durchgeführt. In einem anschließenden Schritt wird ein einfacher Tauschdurchlauf-Algorithmus zur Auflösung von Überlappungen der Verbindungspfeile nach Ali et al. (2005, S. 7) vorgenommen:

```
Durchlaufe alle Label

    Wenn Pfeile sich überlappen, tausche Label
```

5.5.2 Perspektivabhängige Transparenz

Eine der Kernaufgaben des 3D-Viewers ist das Hervorheben von Teilobjekten in der 3D-Szene. Ein Werkzeug dafür ist die Änderung der Transparenz aller nicht hervorgehobenen Komponenten (Bernhard Preim & Felix Ritter, 2002, S. 8–13). In dem vorhergehenden 3D-Viewer war bereits eine einfache Transparenz-Variante möglich, bei der allen Objekten außer den Zielobjekten ein fester Alpha-Wert zugewiesen wurde. Dieser Ansatz ist allerdings problematisch, wenn sich viele Komponenten überlagern, wie es bei komplexen technischen 3D-Modellen der Fall ist. Für die Verwendung von Transparenz in technischen Zeichnungen definieren Diepstraten et al. (2002, S. 2–4) drei Basis-Regeln:

1. Nur die vordersten Flächen eines transparenten Objekts werden angezeigt.
2. Ein opakes Objekt, das sich hinter zwei transparenten Objekten befindet, wird nicht dargestellt, auch wenn es geometrisch sichtbar wäre.
3. Die Transparenz nimmt an den Rändern transparenter Objekte ab und erhöht sich mit zunehmender Entfernung zum Rand.

Die erste Regel ist implizit bereits in Qt umgesetzt durch das sogenannte BackFaceCulling bei dem die rückwärtigen Flächen von Komponenten nicht gerendert werden.

Die zweite Regel ist sehr restriktiv für die Darstellung und würde schnell dazu führen, dass Bestandteile von größeren, komplexeren 3D-Modellen nicht mehr dargestellt werden. Zudem müsste dafür die Implementierung der Sortierung der Render-Objekte angepasst werden, was tiefere Eingriffe in die Render-Pipeline von Qt erfordert.

Die dritte Basisregel basiert grundsätzlich darauf, dass Silhouetten stärker sichtbar sein sollen als innere Flächen. Silhouetten können auf verschiedene Art und Weise bestimmt werden, sei es im 3D-Bereich oder in der gerenderten 2D-Darstellung.

Für die Darstellung speziell von technischen Zeichnungen verwenden Diepstraten et al. (2002, S. 3–4) einen Algorithmus, der in einem ersten Schritt alle Kanten von Objekten bestimmt und in einem zweiten Durchlauf auf Basis der Entfernung zu den Kanten die Transparenz berechnet. Diese Möglichkeit ist schwieriger in eine Standard-Shader Pipeline zu integrieren und grundsätzlich anspruchsvoller für eine Berechnung als eine Transparenz, die direkt auf lokaler Geometrie basiert.

Für das illustrierte Darstellen von komplexen Flächen und Formen und besonders das interaktive Erforschen dieser Formen definieren Hummel et al. (2010, S. 4–5) zwei Varianten der Transparenzberechnung, die einfacher integriert werden können und performanter sind:

1. Winkel-basierte Transparenz
2. Normalen-Variation-basierte Transparenz

Jeder Pixel der gerenderten 3D-Objekte ist Teil eines kleinen Dreiecks. Für die Berechnungen werden häufig die Normale – also die „Blickrichtung“ dieser Fläche genutzt.

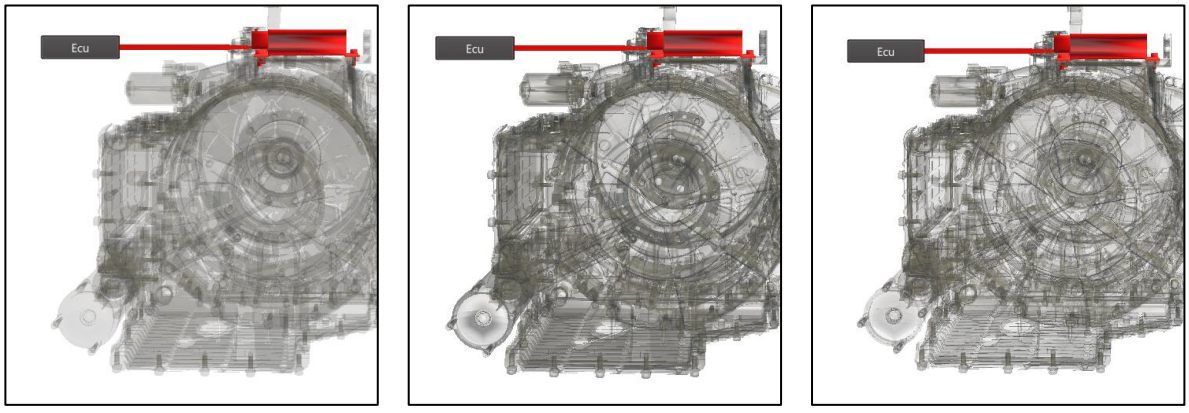
Eine Winkel-basierte Transparenz beschreibt bereits Crow (1978, S. 9) mit der Anpassung der Transparenz auf Basis des Winkels zwischen der Normalen der Fläche und dem Vektor zur Kamera von der Fläche. Allerdings war das Ziel, den Realismus von transparenten Objekten zu verbessern. Die ebenso Winkel-basierte Transparenz nach Hummel et al. (2010, S. 4) ist mit der Normalen n der Fläche und dem Vektor v zur Kamera definiert als $a_{view} := \frac{2}{\pi}\arccos(n \cdot v)$.

Die Normalen-Variations-Transparenz bestimmt die Transparenz der einzelnen Pixel basierend auf der lokalen Variation der Flächennormalen (Hummel et al., 2010, S. 5). Diese Änderung der Normalen wird explizit aus der Perspektive der Kamera bestimmt und ist damit genauso wie die Winkel-basierte Transparenz von der aktuellen Perspektive abhängig. Insbesondere kann diese Änderung über Shader-Funktionen performant approximiert werden.

Beide Varianten wurden im Rahmen der Arbeit für den 3D-Viewer implementiert. Die Wahl der Transparenzberechnung ist eine Abwägung zwischen Verbesserung der Sichtbarkeit von Details versus Überforderung des Nutzers wegen zu vieler Einzelheiten. Der Nutzer kann über Parameter die Transferfunktion der Transparenz beeinflussen. Sowohl Winkel- als auch die Normalen-basierte Transparenzberechnung werden auf einen Wertebereich mit unterer und oberer Schranke gemappt. Für die Normalen-basierte Berechnung kann zusätzlich der Gamma-Wert gesteuert werden, der den Verlauf der Kurve zwischen der unteren und oberen Schranke beeinflusst.

Grundsätzlich ist die Normalen-Variation-basierte Transparenz konzeptionell besser, da die 3D-Modelle in Fahrzeugen oft problematische, ebene Flächen besitzen. Diese ebenen Flächen sind für die Winkel-basierte Transparenz opak, wenn sie aus einem flachen Winkel betrachtet werden und damit schlecht für die Sichtbarkeit (Hummel et al., 2010, S. 4–5).

Im Vergleich zur einheitlichen Transparenz bieten beide Varianten eine deutlich bessere visuelle Differenzierung, wie es die Abbildung 28 zeigt, wobei die winkelbasierte Methode aktuell die praxistauglichere Lösung darstellt für eine einfachere Konfiguration durch den Nutzer.



(a) (b) (c)

Abbildung 28: Verschiedene Transparenzmechanismen in einem technischen 3D-Modell: Einheitlicher Alpha-Wert mit Alpha = 0,3 (a), Winkel-basierte Transparenz mit maximaler Transparenz (b) und Normal-Variations-Transparenz mit Gamma = 0.3 (c). Minimales Alpha = 0,1 und Maximales Alpha = 0.78 in (b) und (c). (Eigene Darstellung)

Die beschriebenen Anpassungen der Transparenz sind global einheitlich. Zusätzliche dynamische Anpassungen der Transparenzen haben hohes Potenzial: Beispielsweise könnte die Transparenz für Komponenten, die die Zielobjekte überlagern, stärker angepasst werden als andere Komponenten. Eine tiefere Ausarbeitung und Analyse entsprechender Mechanismen übersteigen den Umfang dieser Arbeit.

5.5.3 Rendering als Illustration

Linienzeichnungen sind ein zentrales Gestaltungsmittel in Aufbauanleitungen sowie in Wartungs- und Servicedokumentationen. Sie verzichten auf fotorealistische Darstellung zugunsten einer abstrahierten Form, die gezielt visuelle Aufmerksamkeit lenkt und komplexe Sachverhalte effektiv vermittelt (Ivan Herman & David Duke, 2001). Speziell Konturen-Zeichnungen sind eine bevorzugte Darstellungsvariante (Grabli et al., 2010, S. 1–2) und werden typischerweise auch in Anleitungen verwendet (Mohr et al., 2015, S. 2–3).

Zur Unterstützung solcher Darstellungen bietet der 3D-Viewer eine Illustration-Rendering-Option. Schon Saito und Takahashi (1990) beschreiben die Verwendung von Diskontinuitäten in der Tiefe und der Geometrie der 3D-Objekte als Basis für technische Illustrationen: Linien sollen genau dann gezeichnet werden, wenn wir einen ausreichend großen Unterschied in der Tiefe zweier benachbarter Pixel haben oder wenn sich die Richtungen von benachbarten Flächen ausreichend unterscheiden.

Im Qt Quick 3D Framework erfolgt die Implementierung über Effekte (siehe Grundlagen in „Funktionsweise von 3D in Qt und Shadern“): Bildschirmeffekte arbeiten auf einem gerenderten Farb- und Tiefenpuffer. Der Farbbuffer enthält für jeden Bildschirmpixel den vierdimensionalen Farbwert mit dem Rot-, Grün-, Blau- und Transparenzwert. Der Tiefenbuffer enthält die Entfernung der Pixel zur Kamera. Diese Entfernung ist grundsätzlich perspektivisch verzerrt und muss für Vergleiche zwischen Pixeln approximativ linearisiert werden.

Um die Diskontinuitäten der Normalen zu nutzen, wird im ersten Schritt das Material der Objekte so abgeändert, dass die Farbe eines Punktes des 3D-Modells der Normalen an diesem Punkt entspricht. Damit stehen die Normalen als Pixelwerte im Farbbuffer dem späteren Bildschirmeffekt zur Verfügung. Durch Anwendung von Kantendetektionsfiltern auf den Normalen und den Tiefen können die Diskontinuitäten bestimmt werden. In der aktuellen Implementierung wird der auch von Saito und Takahashi (1990, S. 2) empfohlene Kantendetektionsoperator Sobel verwendet, der sich wegen seiner Performanz und guten Ergebnisqualität als allgemein geeignet gilt (Chang et al., 2023, S. 1).

Ein Problem kann bei der Erkennung von Kanten in der Tiefe auftreten: werden flache Flächen aus einem flachen Winkel betrachtet, ist die Differenz der Tiefe zweier Punkte auf dieser Fläche gegebenenfalls tief genug, um als Kante interpretiert zu werden. Um solche Artefakte zu kompensieren, wird zusätzlich der Winkel von den Normalen der Punkte zum Vektor zur Kamera einbezogen: ist der Winkel sehr flach, wird der berechnete Kantenwert abgeschwächt.

Um trotzdem die Hervorhebung zu gewährleisten, werden die hervorgehobenen Komponenten weiterhin in der Hervorhebungsfarbe dargestellt. Der Effekt kann dafür auf Basis des Alpha-Werts im Buffer unterscheiden, ob die Werte die Normalen sind und mit dem Sobel-Operator weiterverwendet werden sollen oder die Werte den Farbwerten der hervorgehobenen Komponenten entsprechen. Die beizubehaltenden Pixel, wie die hervorgehobenen Komponenten, aber auch der Hintergrund, bekommen einen festen Alpha-Wert von 4.0 zugewiesen. Alle anderen Pixel werden als Normalen interpretiert mit dem Alpha-Wert als Skalarprodukt zwischen Normalen der Punkte und dem Vektor zur Kamera.

Im Ergebnis stehen zwei Diskontinuitätsdetektionen zur Verfügung, die für interaktive Darstellung im Qt Quick 3D Framework genutzt werden können. Die Abbildung 29 zeigt Beispiele für die resultierende Darstellung komplexer technischer Bauteile.

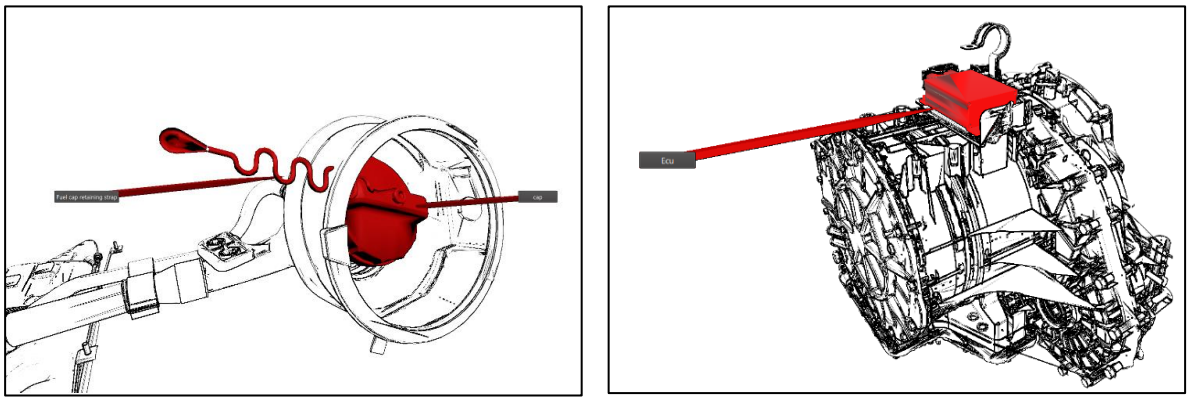


Abbildung 29: Im Illustrationsstil gerenderte Darstellung von einem Tankdeckelaufbau (links) und einem Elektromotor (rechts) mit Hervorhebungen (Eigene Darstellung)

5.5.4 Generation und Animation von Explosionsansichten

Explosionsansichten, wie in Abbildung 30, sind ein etabliertes Mittel zur Darstellung komplexer technischer Baugruppen (Mohr et al., 2015, S. 3). Sie zeigen die interne Struktur und erlauben ein einfacheres Verständnis für die räumlichen Zusammenhänge der Komponenten. Speziell in technischer Dokumentation wie Wartungs- und Reparaturanleitungen werden sie genutzt und unterstützen die Montage oder Demontage visuell (Li et al., 2008, S. 1).

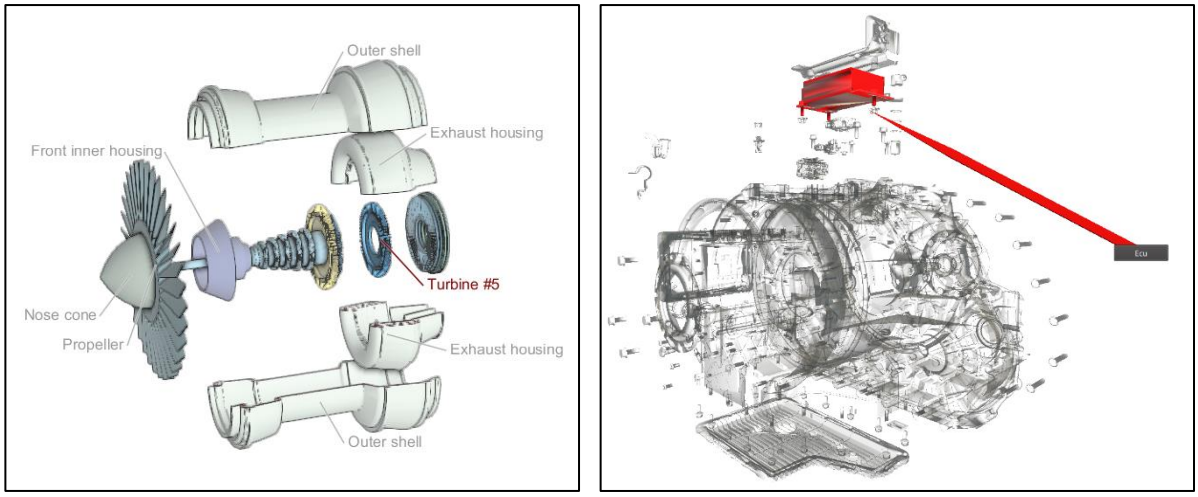


Abbildung 30: Explosionsdarstellungen. Links generiert von Li et al. (2008, S. 1). Rechts voll automatisiert generiert durch 3D-Viewer (Eigene Darstellung)

Für die Unterstützung in der Werkstatt ist die Darstellung von Explosionsansichten ein integraler Bestandteil. Besonders interaktive Explosionsansichten erlauben es, beispielsweise einem Nutzer durch Animation genau zu zeigen, welche Komponenten expandiert werden.

Die Implementierung richtet sich nach der von Li et al. (2008) beschriebenen Aufteilung in die Generation von einem Explosionsgraph und der anschließenden Verwendung des Explosionsgraphen für die Bestimmung der Positionen der Komponenten. Der Explosionsgraph definiert, welche Komponenten von welchen anderen Komponenten blockiert sind und dementsprechend erst im Anschluss nach diesen bewegt werden dürfen. In dem 3D-Viewer erfolgt die Generation des Explosionsgraphen durch den ExplosionGraphGenerator, während die tatsächliche Anpassung der Positionen der 3D-Komponenten durch den ExplosionAnimator durchgeführt wird.

5.5.4.1 Heuristische Bestimmung der Explosionsrichtung

Eine Grundentscheidung für die Entwicklung einer Explosionsansicht ist die Entscheidung, in welche Richtung und in welcher Reihenfolge Komponenten bewegt werden sollen. Dieser Schritt wird oft durch einen Nutzer manuell festgelegt. Für eine einfachere Nutzung ist eine automatische Bestimmung naheliegend. Li et al. (2008, S. 2) definieren die folgenden Konventionen für Explosionsansichten für technische Darstellungen:

1. Blockaden: Komponenten werden in nicht-blockierte Richtungen expandiert.
2. Sichtbarkeit: Komponenten werden so weit verschoben, dass alle relevanten Komponenten sichtbar sind
3. Kompaktheit: Die Entfernungen, die Komponenten bewegt werden, sollten so klein wie möglich sein.
4. Kanonische Explosionsrichtungen: Die meisten 3D-Ansichten beschränken sich auf spezifische Achsen, die sich je nach 3D-Modell unterscheiden können. Gleichzeitig erleichtert eine Beschränkung der Bewegungsachsen die Interpretation der Bewegungen für den Nutzer.
5. Komponentenhierarchie: Die meisten komplexeren Modelle bestehen aus Hierarchien an Komponenten, sodass dementsprechend erst allgemeinere Komponenten voneinander getrennt werden und sich dann erst in ihre Bestandteile aufteilen.

Für die Entwicklung der Explosionsansichten sind damit die folgenden drei Grundregeln Basis für die Richtung, in die sich eine Komponente bewegen sollte:

1. Die Entfernungen, die Komponenten bewegt werden, sollte so klein wie möglich sein relativ zu der Eltern-Komponente in der Hierarchie.
2. Die Richtung sollte einer Standard-Achse der Szene entsprechen.
3. Die Richtung sollte einer kanonischen, natürlichen Bewegungsrichtung des Modells entsprechen.

Eine einfache Heuristik für die kanonische Bewegungsrichtung hat sich als sehr effektiv erwiesen: Es wird der kleinste, achsengleiche, umfassende Quader einer Komponente betrachtet. Betrachtet man den Durchschnitt der Flächeninhalte der drei Achsen können drei Fälle eintreten:

1. Zwei der Flächeninhalte über dem Durchschnitt und einer darunter
2. Zwei der Flächeninhalte unter dem Durchschnitt und einer darüber
3. Alle gleich im Falle eines Würfels

Die Heuristik bestimmt die präferierte Achse als die Achse in Richtung der einzelnen Fläche, wenn möglich. Im Fall 1 die Achse mit den Flächen unter dem Durchschnitt und im Fall 2 die Achse mit den Flächen darüber.

Die Abbildung 31 zeigt eine exemplarisch eine typische Schraubverbindung, die zwei Komponenten über eine Schraube, eine Mutter und eine Unterlegscheibe verbindet.

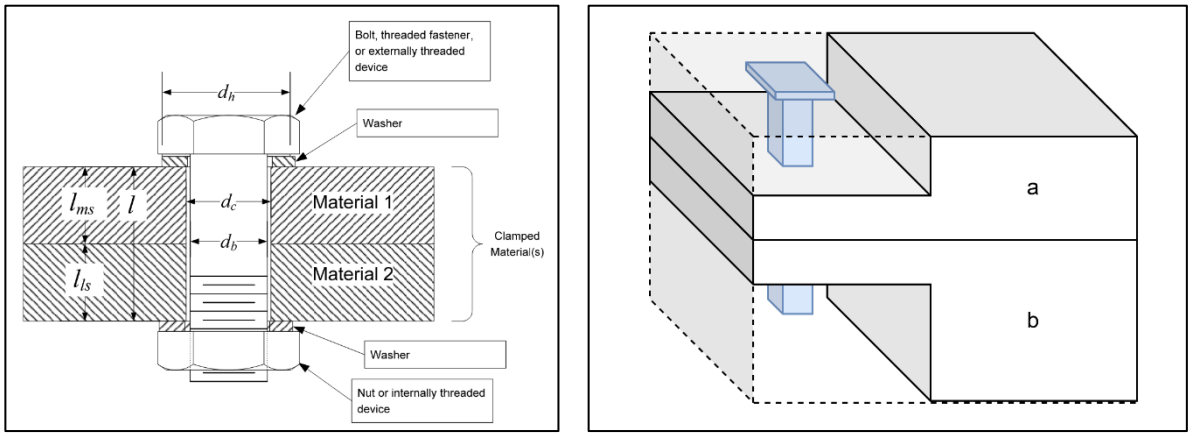


Abbildung 31: Schraubverbindung (links von Brown et al. (2008, S. 10), rechts eigene Darstellung)

Mit der Heuristik werden Schrauben in ihre natürliche Schraubrichtung bewegt. Die Abbildung 32 zeigt die Verwendung von der neuen Heuristik im Vergleich mit einer nur auf Basis der kürzesten Entfernung gewählten Richtung. In der Abbildung 32 links wird die blau gefärbte Schraube in die Richtung geschoben, bei der sie den geringsten Weg zurücklegt. In Abbildung 32 rechts folgt sie der Heuristik und bewegt sich in eine für Schrauben natürlichere Richtung.

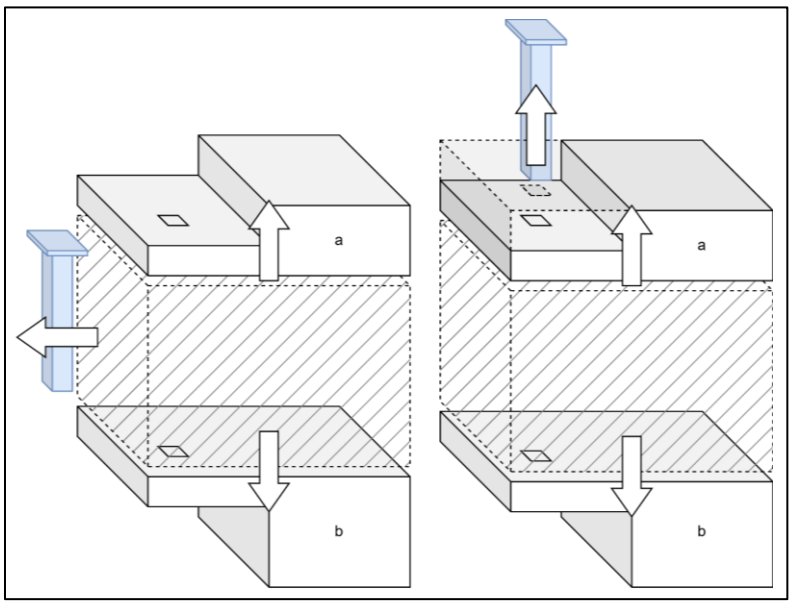


Abbildung 32: Varianten der Explosionsrichtung nach kürzestem Weg (links) und nach der Heuristik (rechts) (Eigene Darstellung)

5.5.4.2 Ermittlung der Komponentenhierarchie und des Explosionsgraphen

Der Explosionsgraph beschreibt die hierarchische Struktur der Komponenten und ergänzt diese um Informationen darüber, welche Komponenten andere entlang bestimmter Bewegungsachsen blockieren. Das bedeutet, dass in einem ersten Schritt diese Komponentenhierarchie ermittelt werden muss. Diese Hierarchie ist im Vergleich zu der Szenenhierarchie explizit auf die physischen, räumlichen Zusammenhänge bezogen. Wenn die Hierarchie definiert ist, kann für die Komponenten bestimmt werden, welche Komponenten durch welche anderen blockiert werden. Das hängt insbesondere von der Explosionsrichtung der Komponenten ab. Diese Abhängigkeiten werden in einem als BlockingGraph bezeichneten Graphen festgehalten.

Basis für die Hierarchiebestimmung und für die Generation der BlockingGraphen ist die Generation von hier als InterferenzGraph bezeichneten Graphen. Ein InterferenzGraph gibt für eine definierte Achse an, inwiefern die Komponenten sich überlagern, wenn sie auf diese Achse projiziert werden. Der Algorithmus richtet sich nach dem Sweep-And-Prune Algorithmus von Cohen et al. (1995, S. 191–193). Im Vergleich zu einem naiven Ansatz von paarweisen Vergleichen kann damit eine Laufzeitverbesserung von $O(n^2)$ zu $O(n \log n)$ erreicht werden, da sich die Vergleiche im Wesentlichen auf die Sortierung der Anfangs- und Endpunkte in den drei Achsen reduziert.

Die Abbildung 33 zeigt für drei Objekte die Generation der InterferenzGraphen. Die Randpunkte der Objekte bezogen auf die aktuelle Achse werden sortiert und daraus bestimmt, wer wen enthält oder zumindest sich überlagert. Für das Beispiel in Abbildung 33 ist das Objekt c zwar bezüglich der y-Achse in a enthalten, aber nicht bezüglich der x-Achse.

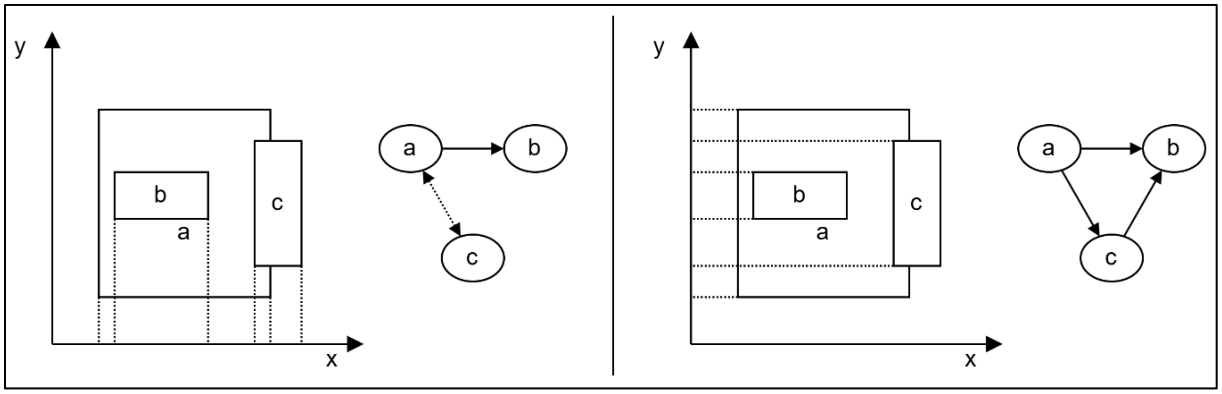


Abbildung 33: Projektion auf x-Achse (links) und auf y-Achse (rechts). Die durchgezogenen Pfeile in den Graphen entsprechen einer „beinhaltet“-Beziehung. Der gestrichelte Pfeil zwischen a und c links bedeutet eine Interferenz bezüglich dieser Achse. (Eigene Darstellung)

Für die Bestimmung einer Komponentenhierarchie werden für die drei Achsen der Szene die Interferenzgraphen bestimmt. Nur wenn eine Komponente in allen drei Projektionen innerhalb derselben übergeordneten Komponente liegt, wird sie als Teilkomponente von dieser festgelegt. Um eine eindeutige Eltern-Kind-Beziehung zwischen Komponenten zu erhalten, wird in einem ersten Schritt eine transitive Reduktion durchgeführt, bei der aus einem Graphen wie in Abbildung 32 rechts redundante Beziehungen entfernt werden, wenn es bereits einen Pfad zu den Knoten gibt. Aus „a zu b“, „a zu c“ und „c zu b“ wird nur noch „a zu c“ und „c zu b“.

Für die Bestimmung des BlockingGraphen werden die Interferenzgraphen für alle Komponenten berechnet, die Teil der gemeinsamen Elternkomponente sind. Um zu entscheiden, von welchen Komponenten eine Komponente blockiert wird, müssen immer zwei der Interferenzgraphen betrachtet werden: Soll sich eine Komponente in x-Richtung bewegen, sind alle Komponenten mögliche Blockaden, mit denen sie bezüglich der y- oder z-Achse interferiert. Vernachlässigt man die Hierarchie und die initiale Überlappung, ist im zweidimensionalen Beispiel von Abbildung 33 leicht erkennbar, dass, wenn Objekt c auf der x-Achse waagrecht bewegt wird, es mit a und b grundsätzlich kollidieren kann. Wird im Vergleich Objekt c vertikal bewegt, kann es nicht mit Objekt b kollidieren. Für die Kandidaten wird zusätzlich geprüft, ob sie allerdings tatsächlich bei der gewünschten Bewegungsrichtung kollidieren. Wenn Objekt c nach rechts bewegt wird, kollidiert es nicht mit Objekt b.

Im Ergebnis besteht der ExplosionsGraph aus hierarchisch strukturierten Komponenten und BlockingGraphen für jede Komponente mit Teilkomponenten. Komponenten sind eindeutig kollidierend, wenn sie sich initial nicht überlagern, aber sich bei der Bewegung in ihre Wunschexplosionsrichtung überlagern. Wenn sich Komponenten initial bereits überlagen, aber nicht komplett gegenseitig enthalten, speichern wir sie als „Kontext“-Komponenten zusätzlich ab. Kontextkomponenten sollten möglichst gemeinsam bewegt werden, um zusammengehörende Komponenten nicht zu stark voneinander zu trennen.

5.5.4.3 Generation von Animationsschritten

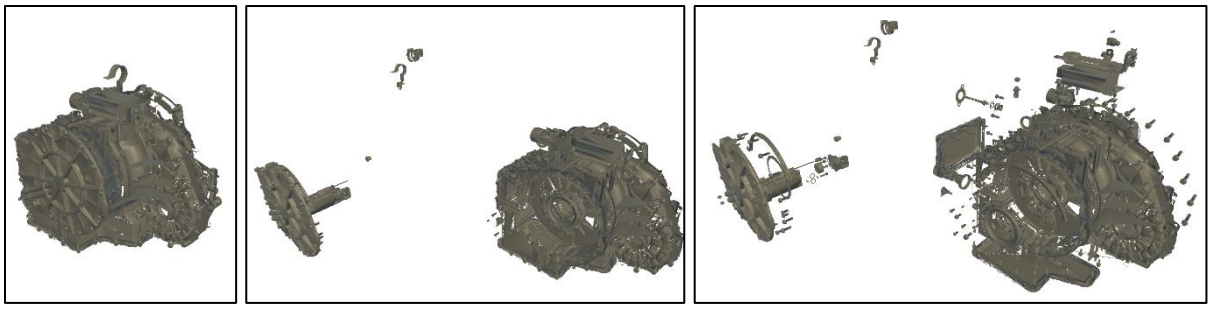
Für die Darstellung der Explosionsansichten müssen aus dem Explosionsgraphen die konkreten Positionsveränderungen der einzelnen Komponenten abgeleitet werden. Es werden aktuell zwei Modi unterstützt: Explosionsansicht des gesamten 3D-Modells und die Extraktion einzelner Komponenten.

Beide Modi basieren darauf, dass in einem ersten Schritt die Offsets-Anpassungen für den aktuellen Vorgang bestimmt werden. Dafür wird rekursiv der Explosionsgraph durchlaufen. Jede Komponente gibt zurück, welchen Bereich und Volumen sie einnimmt, wenn ihre Teilkomponenten vollständig auseinandergezogen sind. Diese Explosionsgröße einer Komponente wird dann verwendet, um zu bestimmen, wie weit sie selbst aus ihrer Elternkomponente bewegt werden muss, um sich vollständig aus dem Elternvolumen zu bewegen. Die berechneten Positionsanpassungen werden in einer Liste gespeichert. Im Gegensatz zu Li et al. (2008) wird die aktuelle Nutzerperspektive aufgrund des hohen Implementierungsaufwands nicht einbezogen. Die Idee wäre dabei, Entfernungen so anzupassen, dass aus einer bestimmten Perspektive die Bildschirmfläche optimal genutzt wird.

Die Animation beginnt – im Gegensatz zur rekursiven Berechnung – nicht bei den untersten Komponenten, sondern nach Beispiel von Li et al. (2008, S. 5) mit den übergeordneten Komponenten und expandiert diese dann in den darauffolgenden Schritten. Ein spezifischer Stand der Animation kann leicht berechnet werden, indem bestimmt wird, welche Positionsveränderungen bis zu diesem Zeitpunkt durchgeführt werden müssen. Besteht die komplette Explosion aus sechs Expansionen von Komponenten, bedeutet ein Aufruf von explode(0.5), dass die ersten drei Schritte der Explosionsliste angewandt werden.

Für die Integration in den 3D-Viewer steht für jeden Schritt das Volumen im 3D-Raum zur Verfügung, in dem sich aktuell etwas ändert. Damit kann der Viewer die Kamera passend verschieben und orientieren.

Die Abbildung 34 zeigt die schrittweise Explosion eines Elektromotors:



(a) (b) (c)

Abbildung 34: Explosionsansichten am Beispiel eines Elektromotors mit der initialen Position (a), der Position nach dem ersten Explosionsschritt (b) und der Endposition (c) (Eigene Darstellungen)

Insbesondere die Richtung der Schrauben und der Dichtungsringe entspricht auf Basis der Heuristik einer natürlichen Richtung. In der Abbildung 35 sind unter anderem Schrauben sichtbar, die sich entsprechend ihrer Schraubrichtung bewegen (gelb). Blau markiert sind die Dichtungsringe, die sich einheitlich in „Flächenrichtung“ bewegen.

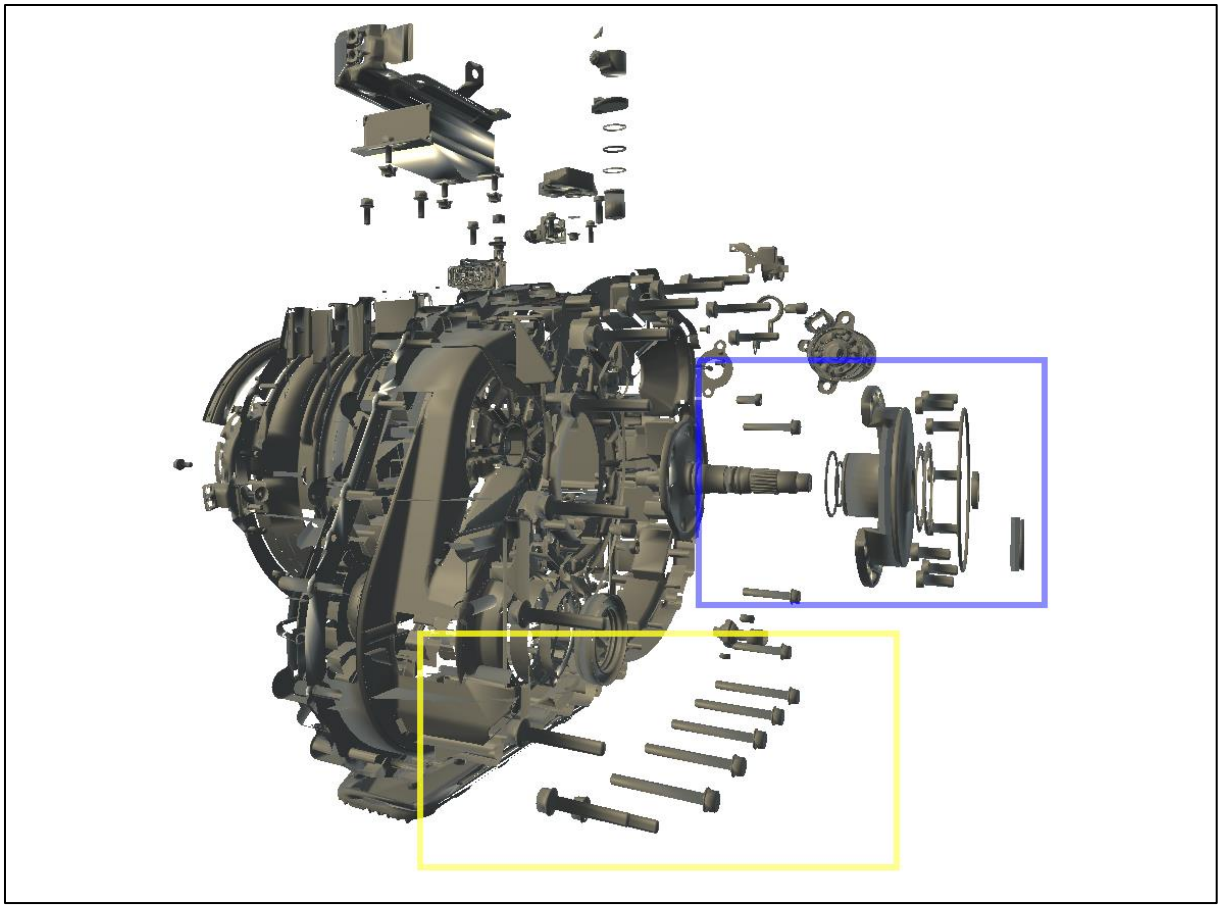


Abbildung 35: Endposition der Explosion: Dichtungsringe bewegen sich in natürliche Richtung (blau), Schrauben bewegen sich nicht nach unten heraus, sondern in Schraubrichtung (gelb) (Eigene Darstellung)

5.6 Gesamtprozess der Datenaufbereitung am Beispiel „Tank-Einfüllklappe“

Es ist folgendes, vereinfachtes Beispielszenario gegeben: „Ein einseitiges DIN-A4-Handout soll die Struktur der Tank-Einfüllklappe beschreiben“. Das Beispiel wurde auf Grundlage von bereitgestellter, praktisch genutzter Werkstattdokumentation erstellt. Ausgangslage ist das einseitige DIN-A4-PDF in seiner Grundform mit einem Bild der Einfüllklappe und zwei Beschriftungen: „Fuel tank retaining strap“ und „cap“. Die Abbildung 36 zeigt einen Ausschnitt des PDF-Dokuments (siehe Abbildung 44 im Anhang für das komplette Dokument).

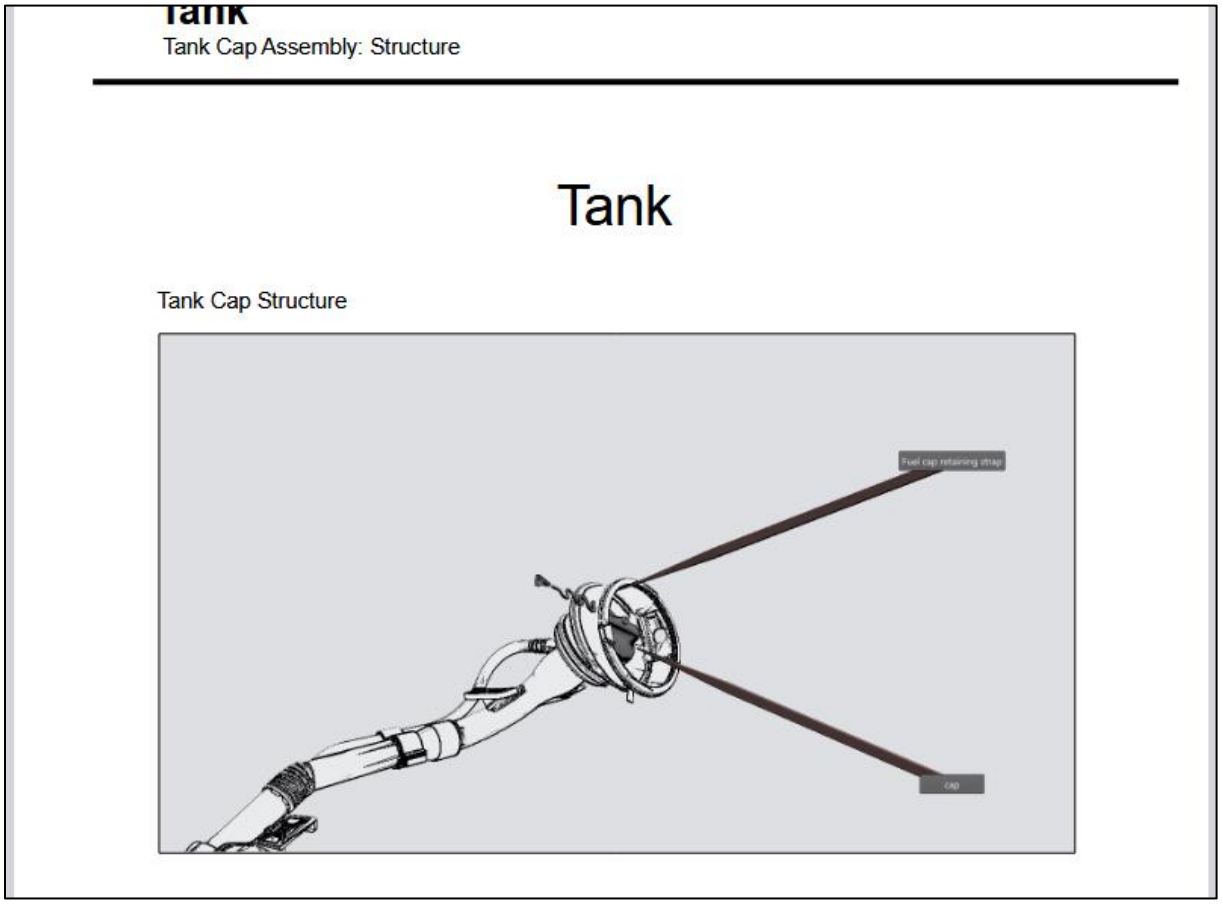


Abbildung 36: Ausschnitt aus dem PDF-Dokument Struktur der Tank-Einfüllklappe (Eigene Darstellung)

Ziel ist es, diese Darstellung mit einer 3D-Szene zu ersetzen. In einem ersten Schritt muss dafür von der Konstruktionsabteilung dafür ein passender Ausschnitt exportiert werden. In diesem Beispiel wurde die Konstruktionsdatei als tank.glb exportiert. Eine .glb-Datei folgt dem Format gLTF 2.0 und behält die interne Struktur der Teile aus der Fertigung bei.

Für dieses Tank-Beispiel wird in einem ersten Schritt ein neues Projekt mit dem Namen „DemoProject“ in einem gewünschten Dateiverzeichnis über einen entsprechenden Dialog im Doc3DCreator angelegt. Dann werden ein neues Modell, eine neue Szene und ein neues Dokument angelegt. Einfachheitshalber wird auch die Szene als Basis-Szene mit „Tank“ bezeichnet. Im Anschluss kann jeweils die Konfiguration erfolgen. Die Abbildung 37 zeigt links die Projektstruktur und rechts die Konfiguration des Modells.

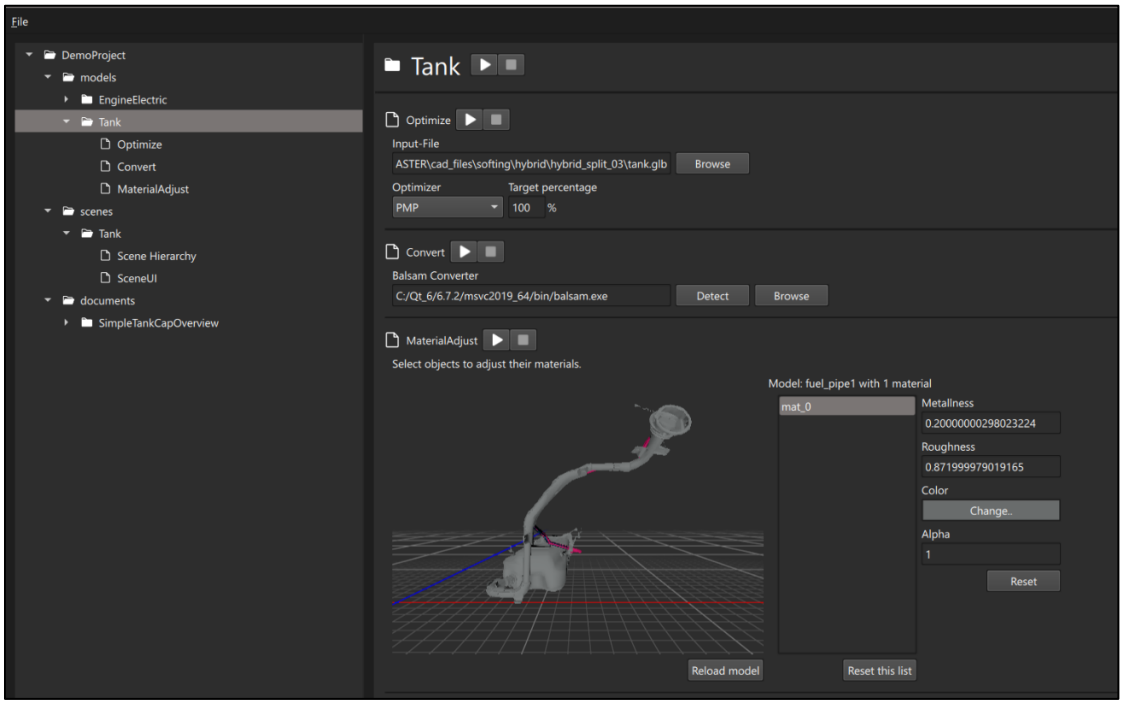


Abbildung 37: Screenshot aus dem Doc3DCreator mit Demoprojekt und ausgewähltem Modell (Eigene Darstellung)

Da die Datei nur sehr klein ist, wird keine Komprimierung durchgeführt. Die Materialien sind für einen ersten Ansatz ausreichend, sodass keine weiteren Anpassungen des Modells nötig sind. Die Abbildung 38 zeigt die anschließende Konfiguration der Szene.

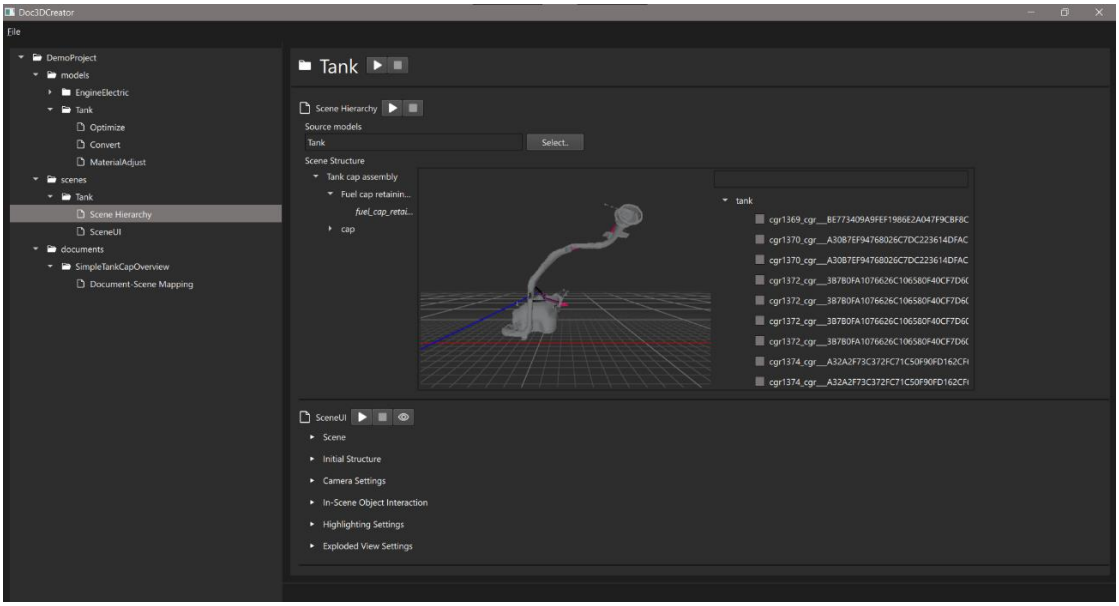


Abbildung 38: Screenshot aus dem Doc3DCreator mit Demoprojekt und ausgewählter Szene (Eigene Darstellung)

Im ersten Schritt der Szenenkonfiguration wird das „Tank“-Modell als Startmodell festlegt. Hier können theoretisch auch weitere Modelle hinzugefügt werden. Über die „SceneStructure“ kann die Struktur editiert werden. Auf der rechten Seite sind die Modellnamen aus der Datei – wie leicht zu erkennen ist, sind diese Namen für eine praktische Nutzung ungeeignet. Der Nutzer kann in der 3D-Szene Komponenten anklicken, die daraufhin der aktuell ausgewählten Nutzer-definierten Szene links zugeordnet werden.

Über den Knopf mit dem Augensymbol wird eine Live-Vorschau der Szene mit den aktuellen Einstellungen angezeigt. Die Abbildung 39 zeigt einen zweigeteilten Bildschirm mit der Konfiguration der Szene links und der Live-Vorschau rechts. In der Abbildung wird die „Technical Drawing“-Rendervariante verwendet, anstatt einer Standarddarstellung.

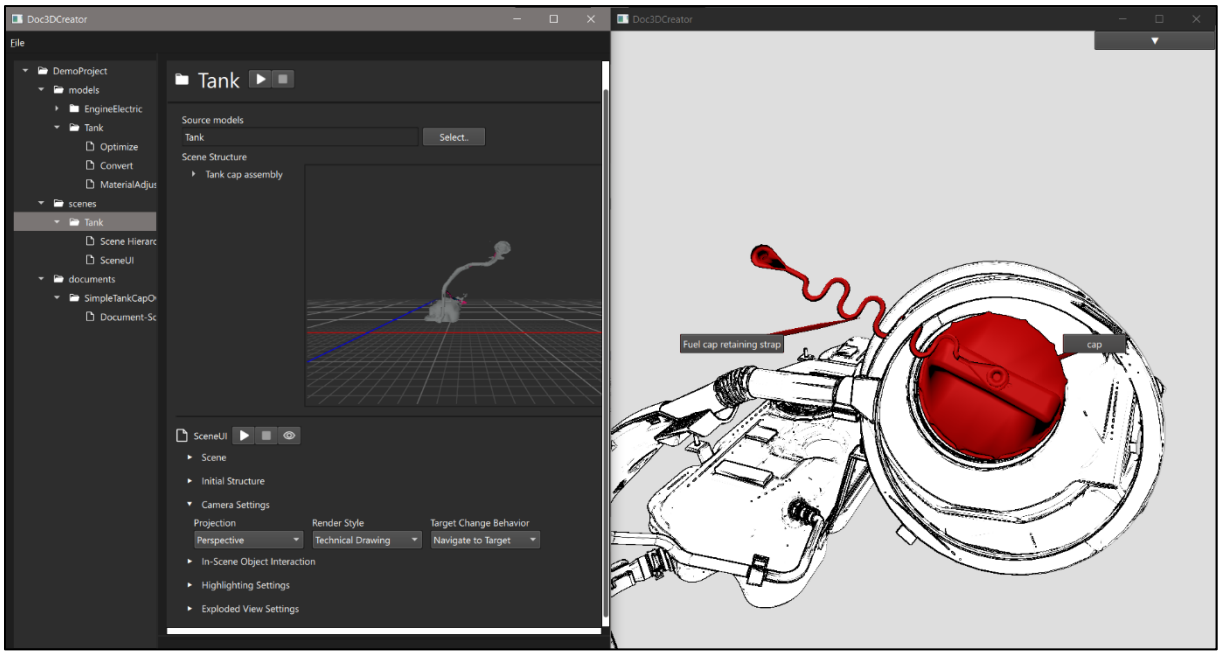


Abbildung 39: Screenshot aus dem Doc3DCreator mit Konturzeichnung in der Live-Vorschau der Szene (Eigene Darstellung)

Der Nutzer kann die Szene den Ansprüchen der jeweiligen Verwendung anpassen. Der Effekt von Hervorhebungen ist grundsätzlich kontextabhängig, sodass ggf. eine Farbanpassung, wie in Abbildung 40 erforderlich ist. Eine Hervorhebung in Rot würde sich unter anderem schlecht von rotem Lack absetzen.

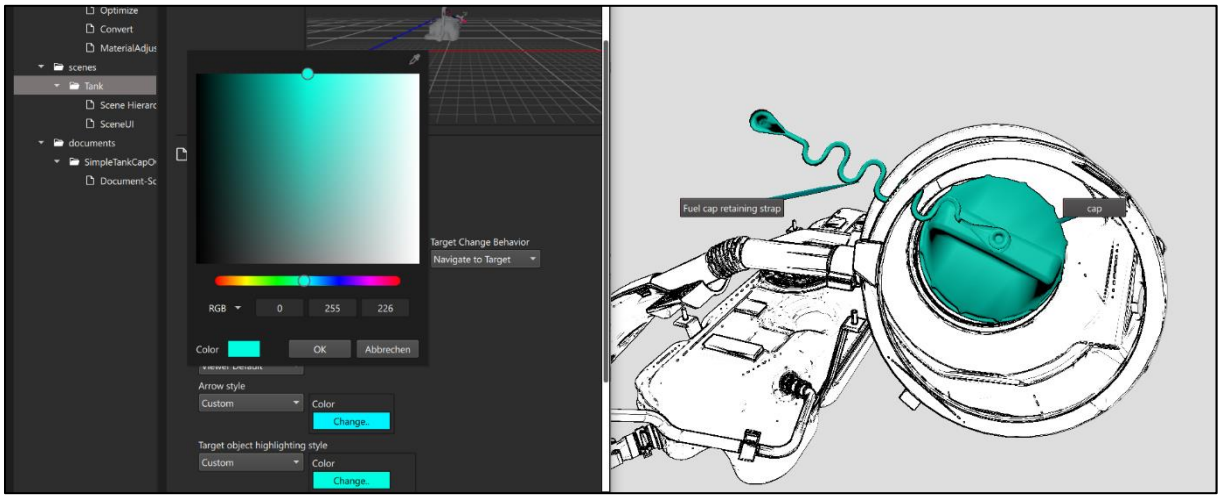


Abbildung 40: Screenshot aus dem Doc3DCreator mit Farbanpassung in der Live-Vorschau der Szene (Eigene Darstellung)

Ist die Szene für die Anwendung konfiguriert, muss der Ort festgelegt werden, an dem sie im Dokument angezeigt werden soll. In diesem einfachen Beispiel ist eine fixe Position einfach umzusetzen. Die Abbildung 41 zeigt das geladene Dokument mit der Positionierungskonfiguration auf der rechten Seite. Die Positionierungskonfiguration legt die Szene, die geladen werden soll, die Position, die Größe und den Anzeige-Modus fest. Damit ist die Konfiguration des Dokuments abgeschlossen.

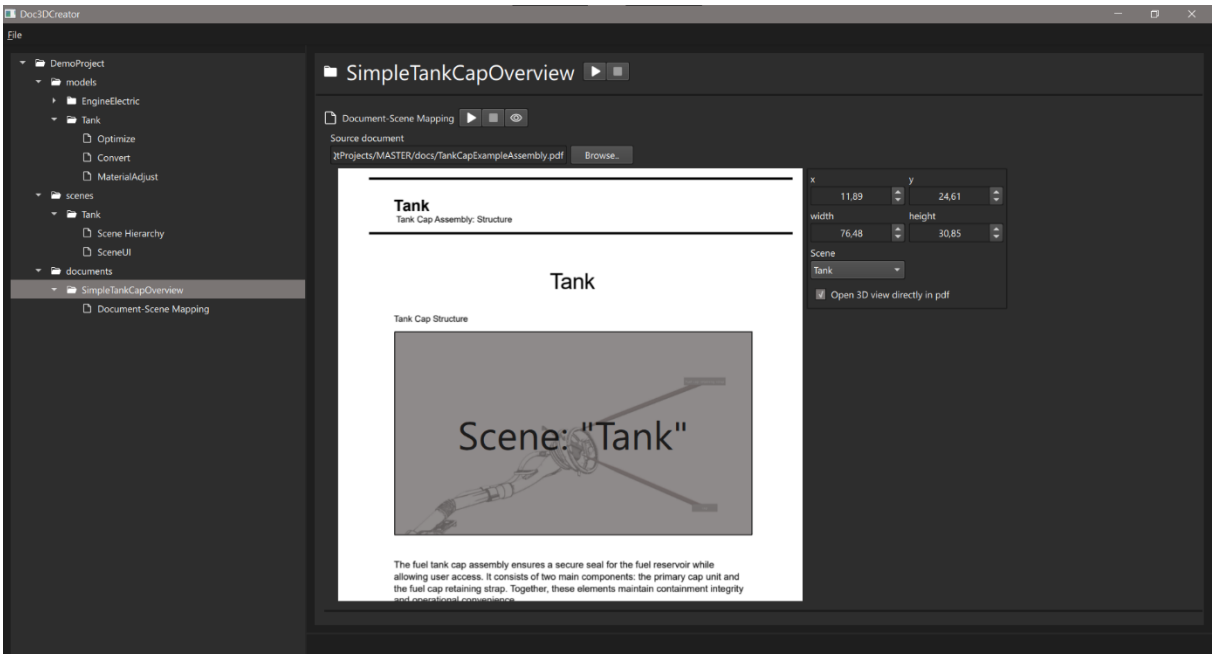


Abbildung 41: Screenshot aus Doc3DCreator: Das Bild in dem geladenen Dokument wird von der 3D-Szene überlagert (Eigene Darstellung)

Im Dokumentationsviewer wird die Darstellung der Dokumentation um die 3D-Darstellung erweitert. Initial wird ein transparentes Fenster mit Lade-Möglichkeit eingeblendet. Nach dem Klicken des Nutzers auf „Load Tank“ wird die interaktive 3D-Szene nahtlos in das Dokument eingebunden und als Teil des Dokuments behandelt.

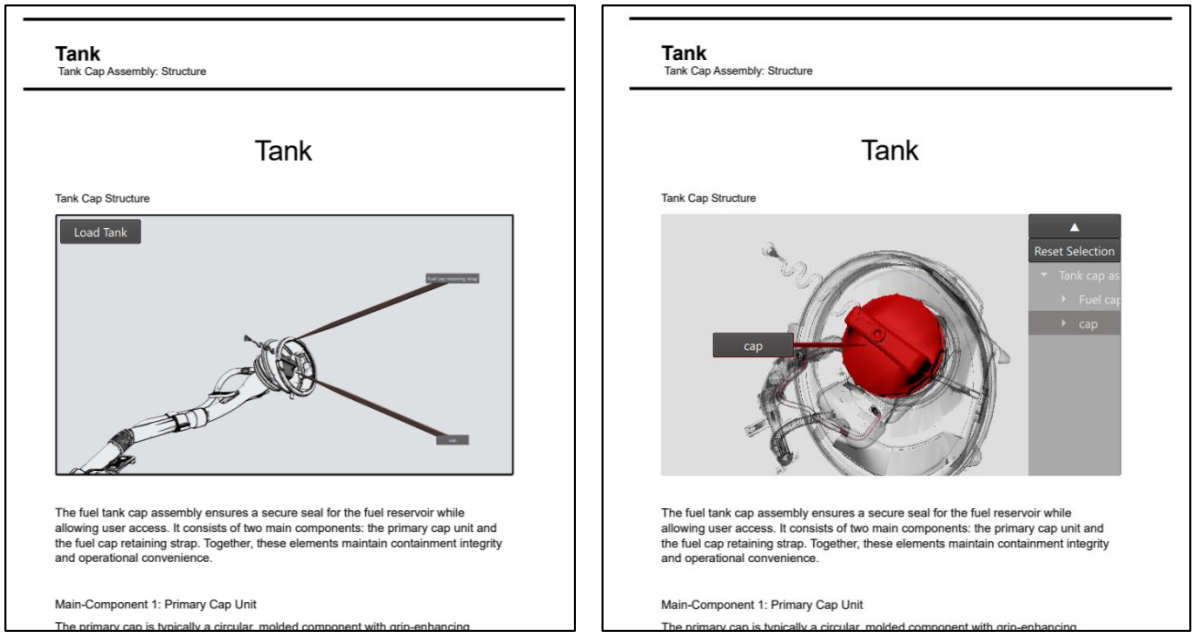


Abbildung 42: Darstellung des Tank-PDF-Dokuments mit dem ursprünglichen statischen Bild links, ergänzt mit einem "Load Tank"-Button. Rechts die Darstellung nach dem Laden der interaktiven 3D-Szene (Eigene Darstellung)

6 Evaluation

Das entwickelte System erfüllt mit seinem Funktionsumfang 24 der 28 expliziten Anforderungen. Vier Anforderungen sind nicht vollständig praxistauglich erfüllt:

1. Die Anzeige der teilebezogenen Zusatzinformationen wird zwar grundsätzlich von den Datenmodellen unterstützt, kann aber aktuell nicht über die Oberfläche des Doc3DCreators konfiguriert werden. Im aktuellen Zustand kann nur eine einzelne Beschriftung für jede definierte Komponente angezeigt werden. Das ist üblicherweise die Bezeichnung.
2. Es werden prinzipiell nur Qt-Komponenten verwendet, die plattformunabhängig bezüglich Touch- und Desktop-Nutzung sein sollten, aber der 3D-Viewer und der Dokumentationsviewer wurden nur für die Plattform Windows gebaut und getestet.
3. Der Dokumentationsviewer bietet die Möglichkeit, PDF-Dokumente ergänzt mit 3D-Darstellungen anzuzeigen. Derzeit fehlen die üblichen Steuerungsmechanismen, wie das Inhaltsverzeichnis oder eine Anzeige der aktuellen Seitenzahl. Für eine praktische Nutzung ist der Dokumentationsviewer nicht ausgereift genug.
4. Die Konfiguration der 3D-Darstellungen über den Doc3DCreator ist ohne Expertenwissen möglich. Eine praktische Nutzung sollte allerdings durch Komfortfunktionen unterstützt werden.

Für eine Nutzung in der Praxis sollten sowohl die Oberflächen der Pipelineschritte als auch die Oberfläche des Dokumentationsviewer mit zusätzlichen Komfortfunktionen ausgestattet werden, um eine intuitive und damit effektive Nutzung zu ermöglichen. Ebenso wurde im Rahmen der Entwicklung das Testen der Nutzeroberflächen vernachlässigt und sollte selbstverständlich Teil der weiteren Entwicklung sein.

Die Code-Qualität wurde durch das Programm Cppcheck (Version 2.17.1) mit dem höchsten Prüflevel *exhaustive* statisch analysiert. Abgesehen von nichtzutreffenden Hinweisen zu const-Parametern und Warnungen für die angepasste PMP-Logik werden keine Mängel erkannt. Die einzelnen Datenmodelle wurden mit dem Qt gestellten QAbstractItemModelTester auf die Grundfunktionalitäten getestet. Der QAbstractItemModelTester untersucht das Modell auf Inkonsistenzen, beispielsweise bezüglich in der Modellstruktur. Zusätzlich wurden die Datenmodelle mithilfe der Data-Driven-Testing-Unterstützung von Qt auf die über die Basis-Funktionen hinausgehenden Funktionalitäten überprüft.

Weitere Tests wurden für die mathematischen Basis-Funktionen geschrieben, die die Algorithmen beispielsweise nutzen, um die optimalen Labelpositionen zu finden.

In den folgenden Kapiteln werden die Anforderungen und ihre Umsetzung skizziert. Jeder Abschnitt enthält eine Übersichtstabelle der jeweiligen Anforderungen und ihrem Stand. Die Spalte „Stand“ gibt mit „ja“ an, dass diese Anforderung erfüllt wird.

6.1 Stand des 3D-Viewers

Der Nutzer konfiguriert in der Pipeline-Applikation die Struktur (REQ_F_V_02_STRUC), die Art und Weise der Darstellung (REQ_F_V_04_HIGHL, REQ_F_V_08_CONF, REQ_NF_V_04_HIGHL). Der 3D-Viewer zeigt die konfigurierten Szenen an (REQ_F_V_01_SHOW) und erlaubt Interaktionen, wie Anklicken der Beschriftungen zur Traversierung der Komponentenstruktur (REQ_F_V_03_TRAV).

Die Darstellung in der 3D-Szene wurde mit einem bereitgestellten Elektromotor-Modell getestet. Der Motor besitzt 500.000 Einzelflächen und 215 Teilkomponenten getestet und bleibt auf einem Laptop mit 60 Bildern pro Sekunde stabil (REQ_NF_V_01_PERF). Ein größeres Modell mit ca. 9 Millionen Einzelflächen und ca. 3000 Teilkomponenten konnte noch flüssig mit 20 Bildern pro Sekunde angezeigt werden, aber der Ladevorgang betrug ca. 20 Sekunden. Das Laptop besitzt einen i7-6700HQ @ 2.60GHz Prozessor, 16 GB Arbeitsspeicher und einer NVIDIA Quadro M100M Grafikkarte mit 2 GB Grafikspeicher.

Der 3D-Viewer passt sich seiner verfügbaren Fenstergröße responsiv an (REQ_NF_V_03_SIZE) und enthält grundsätzlich nur Komponenten, die nach Angaben von Qt plattformunabhängig sind (REQ_NF_V_02_PLTF).

Das Datenmodell kann prinzipiell Zusatzinformationen für Teile der Struktur enthalten, aber es gibt zum aktuellen Zeitpunkt keine Konfigurationsmöglichkeit in der Pipeline und keine Zusatzanzeigefunktion im 3D-Viewer. Für eine tatsächliche Nutzung sollte die grafische Oberfläche des 3D-Viewers ebenso einfacher gestaltet werden oder der Konfigurator die Möglichkeit bekommen, spezifisch festzulegen, welche Oberflächenelemente gewünscht sind: Beispielsweise, ob eine Baumstruktur des Modells eingeblendet werden soll oder nicht.

Die Tabelle 15 enthält die Übersicht der Anforderungen:

ID	Anforderung	Stand
Funktionale Anforderungen		
REQ_F_V_01_SHOW	Anzeige von interaktiven 3D-Darstellungen auf Basis passend aufbereiteter Konstruktionsdaten	Ja
REQ_F_V_02_STRUC	Unterstützung einer logischen und physischen (Bau-)Gruppenstruktur zur Strukturierung der Komponenten	Ja
REQ_F_V_03_TRAV	Bereitstellung von UI- und programmatischen Mechanismen zur Traversierung der Gruppenstruktur	Ja
REQ_F_V_04_HIGHL	Konfigurierbare Hervorhebung (z. B. Farbe, Transparenz, Fokus) von Komponenten	Ja
REQ_F_V_05_ANNO	Anzeige von teilebezogenen Zusatzinformationen (z. B. Drehmoment, Einbauhinweise, Diagnosezustände)	Nein
REQ_F_V_06_EXPL	Bereitstellung von Explosionsdarstellungen für z. B. (De-)Montageprozesse	Ja

REQ_F_V_07_SELEC	Rückgabe der aktuellen Auswahl aus der 3D-Darstellung an andere Systemkomponenten (z. B. Werkstatttester)	Ja
REQ_F_V_08_CONF	Parametrierbare Konfiguration der Darstellung (Kamera, Sichtbarkeit, Interaktionsregeln)	Ja
Nicht-Funktionale Anforderungen		
REQ_NF_V_01_PERF	Stabile Echtzeitperformanz bei Nutzung realer Konstruktionsdaten (stabile Anzahl der Bilder pro Sekunde bei Modellen mit der Größenordnung von 500.000 Einzelflächen und 200 Teilkomponenten)	Ja
REQ_NF_V_02_PLTF	Grundsätzliche Plattformunabhängigkeit bezüglich Touch- und Desktop-Nutzung	Ja, aber ungetestet
REQ_NF_V_03_SIZE	Anpassung an variable Fenstergrößen (responsives Verhalten)	Ja
REQ_NF_V_04_HIGHL	Bereitstellung anwendungsbezogener, zielführender Hervorhebungsmechanismen	Ja

Tabelle 15: Anforderungen an den 3D-Viewer mit Stand der Anforderungen (Eigene Darstellung)

6.2 Stand des Dokumentationsviewers

Der Dokumentationsviewers zeigt die bestehende PDF-Dokumentation (REQ_F_DV_01_DOC) ergänzt mit den 3D-Darstellungen (REQ_F_DV_02_INT) an. Die Nutzeroberfläche des Dokumentationsviewer bietet in seinem aktuellen Entwicklungsstand allerdings nicht die üblichen Kontrollmechanismen, wie sie sonst in PDF-Viewern üblich sind. Damit ist zwar die grundsätzliche Anforderung der Darstellung erfüllt, aber eine Nutzung in der Praxis ist damit erschwert.

Abhängig von der Konfiguration können 3D-Szenen auch extern in extra Fenstern geöffnet werden (REQ_F_DV_03_EXT). 3D-Darstellungen sind eigene, für sich stehende Oberflächenkomponenten, die das Dokument überlagern. Sie müssen nicht geladen werden – die Anzeige ist für den Nutzer optional. Sollte sie nicht funktionieren, kann der Nutzer weiterhin die bestehende Dokumentation nutzen (REQ_NF_DV_01_MOD).

Die Referenzierung der 3D-Szenen mit spezifischen Orten in der PDF-Dokumentation erfolgt, ohne dass die PDF-Dokumente modifiziert werden müssen, auf Basis von festen Seiten und Positionen oder relativ zu definierten Texten (REQ_NF_DV_01_MOD).

Tabelle 16 enthält die Übersicht der Anforderungen:

ID	Anforderung	Stand
Funktionale Anforderungen		
REQ_F_DV_01_DOC	Darstellung bestehender Dokumentationsinhalte	Ja, aber für die praktische Nutzung nicht ausreichend
REQ_F_DV_02_INT	Integrierbare Anzeige der interaktiven 3D-Ansichten als Bestandteil der Dokumentation	Ja
REQ_F_DV_03_EXT	Möglichkeit zur parallelen, externen Darstellung von 3D-Inhalten außerhalb der eingebetteten Dokumentationsansicht	Ja
Nicht-Funktionale Anforderungen		
REQ_NF_DV_01_MOD	Integration der 3D-Darstellungen ohne tiefgreifende Modifikationen bestehender Dokumentationsprozesse	Ja
REQ_NF_DV_02_ROB	Fehlerrobustes Verhalten bei nicht ladbaren 3D-Inhalten oder beschädigten Konfigurationen	Ja

Tabelle 16: Anforderungen an den Dokumentationsviewer mit Stand der Anforderungen (Eigene Darstellung)

6.3 Stand der Datenaufbereitungspipeline

Die Datenaufbereitungspipeline transformiert die CAD-Daten, sodass sie für die Darstellung im 3D-Viewer geeignet sind (REQ_F_P_01_CAD, REQ_NF_P_02_SUPP). Der Input ist limitiert auf 3D-Daten ohne Texturen, wie es grundsätzlich auch der Fall ist für 3D-Daten der Fertigung. Der Nutzer kann über das Definieren von Modellen, Szenen und Dokumenten die Ergebnisse konfigurieren. Szenen haben definierbare Strukturen (REQ_F_P_02_STRUC) und Darstellungen (REQ_F_P_03_SCENE). Dokumente enthalten die Referenzierung zwischen den bestehenden Fahrzeugdokumenten und den 3D-Szenen (REQ_F_P_04_DOC). Ergebnis der Pipeline sind grundsätzlich Ordner und Dateien. Modelle, Szenen und Dokumente entsprechen jeweils einem Ordner, der auf einem Server zur Verfügung gestellt werden kann. Ein 3D-Dokumentationsdatenpaket setzt sich aus den Modellen, den Szenen und der Dokumentation mit Referenzierung zusammen (REQ_F_P_05_EXP).

Die Konfigurationen der Modelle, Szenen und Dokumente werden auf Dateiebene persistiert (REQ_NF_P_01_REPR) und können dementsprechend ohne größeren Aufwand mit Datei-Versionsverwaltungen wie Git integriert werden (REQ_NF_P_06_UPD).

Die Pipeline ist modular aus Schritten aufgebaut, die sich zu der Endpipeline zusammensetzen. Ein Hinzufügen von weiteren Schritten beispielsweise zur Internationalisierung wird von der Architektur unterstützt (REQ_NF_P_05_MOD). Jeder Schritt bietet eine Nutzeroberfläche zur Konfiguration sofern erforderlich. Die Anpassungen betreffen im Wesentlichen Präferenzen des Nutzers und sind nicht zwingen erforderlich.

Zwar ist im aktuellen Zustand kein Expertenwissen nötig, aber die aktuelle Oberfläche könnte für den Nutzer nicht so einfach zu bedienen sein, wie es gewünscht ist. Das betrifft konkret die folgenden Punkte:

Optimize-Schritt: Der Nutzer könnte unterstützt mit groben Angaben werden, wie viel Zeit das Laden der 3D-Daten mit der aktuell gewählten Vereinfachungsstufe benötigt. Ebenso könnte die Suche nach dem besten Grad der Optimierung der 3D-Daten für den Nutzer vereinfacht werden: Dafür könnten für den Nutzer mehrere unterschiedliche Stufen der Vereinfachung generiert werden, sodass die Wahl des Grads der Optimierung leichter fällt.

1. Material-Anpassung: Eine grundsätzliche Materialanpassung möglich, aber ein Beispiel-Repertoire könnte einen unerfahrenen Nutzer unterstützen.
2. Szenenstruktur-Erstellung: Der Nutzer kann über einen Klick in der 3D-Szene Komponenten zum aktuell ausgewählten Strukturknoten hinzufügen. Eine Unterstützung könnten hier umfangreichere Auswahl-Tools wie eine Auswahl über Drag-And-Drop-Rechteck oder einen frei nutzerdefinierten Pfad sein. Ebenso könnte eine Auswahl noch Kriterien wie „Komponenten in x Entfernung“ die Definition der Struktur beschleunigen.
3. Szenendarstellung-Konfiguration: Der Nutzer kann zwischen inaktiv, Standard und eigener Konfiguration für die meisten Darstellungen wählen. Verschiedene Standardszenen für häufiger auftretende Muster könnten dann als Vorlagen genutzt werden. Auf Basis der Vorlage können dann weitere Modifikationen durchgeführt werden. Das erspart eine komplett neue Auswahl der Kriterien bei einer sehr ähnlichen Szene.

Die Tabelle 17 enthält die Übersicht der Anforderungen für die Datenaufbereitungspipeline:

ID	**Anforderung**	**Stand**
Funktionale Anforderungen		
REQ_F_P_01_CAD	Aufbereitung von CAD-Daten zu visualisierungsorientiertem Format	Ja
REQ_F_P_02_STRUC	Möglichkeit für nutzergesteuerte Neustrukturierung der Daten gemäß logischer oder physischer Gruppierung (z. B. Baugruppen)	Ja
REQ_F_P_03_SCENE	Generierung konfigurierbarer Darstellungsbeschreibungen	Ja
REQ_F_P_04_DOC	Konfiguration der Verknüpfung von 3D-Darstellungen mit Dokumentationsinhalten	Ja
REQ_F_P_05_EXP	Erstellung auslieferungsgeeigneter Datenpakete	Ja
Nicht-Funktionale Anforderungen		
REQ_NF_P_01_REPR	Reproduzierbarkeit der Pipeline-Ergebnisse bei identischen Eingabedaten	Ja
REQ_NF_P_02_SUPP	Unterstützung gängiger CAD-Datenformate (z. B. STEP, JT etc.)	Ja

REQ_NF_P_03_AUTO	Hoher Automatisierungsgrad mit minimal manuellen Schritten	Ja
REQ_NF_P_04_NOEX	Keine Notwendigkeit von Expertenwissen zur Konfiguration der Pipeline	Ja, aber mehr Unterstützung nötig
REQ_NF_P_05_MOD	Unterstützung modularer Erweiterung (z. B. zusätzliche Verarbeitungsschritte)	Ja
REQ_NF_P_06_UPD	Unterstützung für Update- und Änderungsmanagement zur Nachverfolgbarkeit von Input-, Output- und Konfigurationsänderungen	Ja

Tabelle 17: Anforderungen an die Datenaufbereitungspipeline mit Stand der Anforderungen (Eigene Darstellung)

7 Fazit

Im Rahmen dieser Arbeit wurde ein durchgängiges System zur Integration interaktiver 3D-Visualisierungen in bestehende Fahrzeugdokumentation und Werkstattprogramme entwickelt. Es deckt den vollständigen Prozess von der Übernahme konstruktiver 3D-Daten bis hin zur angereicherten Darstellung in einem Dokumentationsviewer ab. Das System besteht aus drei Hauptkomponenten: einer modularen, nutzergeführten Datenaufbereitungspipeline, einer vielseitig konfigurierbaren 3D-Viewer-Komponente und einem Dokumentationsviewer zur Anzeige von PDF-Dokumenten mit integrierten 3D-Szenen.

Die Datenaufbereitungspipeline erlaubt eine weitgehend automatisierte Transformation der CAD-Daten zu visualisierungsgeeigneten Szenen, die sich in bestehende Dokumente oder Werkstatttester-Programme integrieren lassen. Die Platzierung der Szenen in Dokumenten erfolgt über einen stabilen, relativ zu Text oder Seiten definierten Referenzierungsmechanismus. Der Dokumentationsviewer integriert auf Basis dieser Referenzierung die interaktiven 3D-Darstellungen nahtlos in PDF-Dokumente – ohne Änderungen am ursprünglichen Dokument. Das System ermöglicht es, mit minimalem Expertenwissen vollständige 3D-Dokumentationspakete zu erstellen, ohne bestehende Softwarestrukturen tiefgreifend zu ändern.

Der 3D-Viewer selbst visualisiert die aufbereiteten Konstruktionsdaten flexibel konfigurierbar. Er bietet zusätzlich zu den einfachen Hervorhebungsmechanismen des 3D-Viewers nach Nandico (2023) weitere Darstellungsmechanismen, wie sie für die technische Service- und Wartungsdokumentation benötigt und verwendet werden. Dazu gehören automatisch generierte Konturzeichnungen, sichtbarkeitsoptimierte Transparenzberechnungen und automatische Explosionszeichnungen.

Die Evaluation ergab, dass nahezu alle definierten Anforderungen vollständig erfüllt werden. Die Ergebnisse belegen eine grundsätzliche Praxistauglichkeit des Systems, insbesondere im Hinblick auf Visualisierungsqualität und Flexibilität. Verbesserungspotenzial besteht bei den Nutzeroberflächen, primär im Dokumentationsviewer und in Teilen der Pipeline. Die Komponenten bieten aktuell nur eingeschränkte Komfortfunktionen und sind für eine echte Praxisnutzung noch nicht ausreichend getestet. Zudem ist die plattformunabhängige Ausführung bisher nur konzeptionell abgedeckt, wurde aber nicht überprüft.

Zukünftige Arbeiten sollten die Usability stärker in den Fokus nehmen. Mögliche nächste Schritte wären: eine erleichterte Konfiguration von Szenenstrukturen, Auswahlhilfen bei der Konfiguration der Szenendarstellung, erweiterte Material- und Geometrieoptimierungsoptionen sowie ein systematischer Ausbau der Nutzerführung. Ebenso wäre eine tiefere Integration mit Optimierungsbibliotheken zur effizienteren Geometrieverarbeitung wünschenswert, um die Qualität noch zu verbessern und die Ausführung zu beschleunigen.

Langfristig bietet das System ein hohes Potenzial für die Integration in reale Abläufe von Fahrzeugkomponentenherstellern und Werkstätten. Nach Einschätzung eines verantwortlichen Produktmanagers für die Aftersales-Diagnose gilt:
„Wenn neben den beschriebenen Usabilityverbesserungen ein Konzept entwickelt wird, wie die Modell-, Szenen- und Dokumentpakete ausgeliefert werden können, steht aus aktueller Sicht einem produktiven Einsatz nichts entgegen.“

Literaturverzeichnis

Ali, K., Hartmann, K. & Strothotte, T. (2005). Label Layout for Interactive 3D Illustrations. *Journal of WSCG*, *13*, 1–8.

Bekos, M., Niedermann, B. & Nöllenburg, M. (2021). External Labeling: Fundamental Concepts and Algorithmic Techniques. *Synthesis Lectures on Visualization*, *8*, 1–130. https://doi.org/10.2200/S01115ED1V01Y202107VIS014

Bernhard Preim & Felix Ritter (2002). Techniken zur interaktiven Hervorhebung von Objekten in medizinischen 3d-Visualisierungen. In *Simulation und Visualisierung 2002 (SimVis 2002), 28. Februar - 1. März 2002, Magdeburg*. https://www.researchgate.net/publication/221510621_Techniken_zur_interaktiven_Hervorhebung_von_Objekten_in_medizinischen_3d-Visualisierungen

Bharadwaj, A. G. & Starly, B. (2022). Knowledge graph construction for product designs from large CAD model repositories. *Advanced Engineering Informatics*, *53*, 101680. https://doi.org/10.1016/j.aei.2022.101680

Blender Online Community. (2018). *Blender - a 3D modelling and rendering package*. http://www.blender.org

Brown, K., Morrow, C., Durbin, S. & Baca, A. (2008). *Guideline for bolted joint design and analysis : version 1.0.* https://doi.org/10.2172/929124

Bruckner, S. & Groller, M. E. (2005). VolumeShop: an interactive system for direct volume illustration. In *VIS 05. IEEE Visualization, 2005*.

Burley, B. (2012). Physically Based Shading at Disney. In *ACM SIGGRAPH 2012 Courses.* ACM.

Chang, Q., Li, X., Li, Y. & Miyazaki, J. (2023). Multi-directional Sobel operator kernel on GPUs. *J. Parallel Distrib. Comput.*, *177*(C), 160–170. https://doi.org/10.1016/j.jpdc.2023.03.004

Cohen, J. D., Lin, M. C., Manocha, D. & Ponamgi, M. (1995). I-COLLIDE. In M. Zyda (Hrsg.), *ACM Conferences, Proceedings of the 1995 symposium on Interactive 3D graphics* (189-ff). ACM. https://doi.org/10.1145/199404.199437

Crow, F. C. (1978). Shaded Computer Graphics in the Entertainment Industry. *Computer*, *11*(3), 11–22. https://doi.org/10.1109/C-M.1978.218090

Crytek. (2016). *CryEngine 5 Documentation*. https://www.cryengine.com/docs/static/engines/cryengine-5/

Diepstraten, J., Weiskopf, D. & Ertl, T. (2002). Transparency in Interactive Technical Illustrations. *Computer Graphics Forum.* Vorab-Onlinepublikation. https://doi.org/10.1111/1467-8659.t01-1-00591

Engelke, T., Keil, J., Rojtberg, P., Wientapper, F., Schmitt, M. & Bockholt, U. (2015). Content first. In W. T. Ooi, W. Feng & F. Liu (Hrsg.), *ACM Digital Library, Proceedings of the 6th ACM Multimedia Systems Conference* (S. 105–111). ACM. https://doi.org/10.1145/2713168.2713169

Epic Developer Community. (2025, 12. Mai). *Unreal Engine 5.5 Dokumentation*. https://dev.epicgames.com/documentation/de-de/unreal-engine/unreal-engine-5-5-documentation

Garland, M. & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In G. S. Owen, T. Whitted & B. Mones-Hattal (Hrsg.), *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97* (S. 209–216). ACM Press. https://doi.org/10.1145/258734.258849

Gattullo, M., Scurati, G. W., Fiorentino, M., Uva, A. E., Ferrise, F. & Bordegoni, M. (2019). Towards augmented reality manuals for industry 4.0: A methodology. *Robotics and Computer-Integrated Manufacturing*, *56*, 276–286. https://doi.org/10.1016/j.rcim.2018.10.001

Grabli, S., Turquin, E., Durand, F. & Sillion, F. X. (2010). Programmable rendering of line drawing from 3D scenes. *ACM Transactions on Graphics*, *29*(2). https://doi.org/10.1145/1731047.1731056

Hartmann, K., Götzelmann, T., Ali, K. & Strothotte, T. (2005). Metrics for functional and aesthetic label layouts. In *SG'05, Proceedings of the 5th International Conference on Smart Graphics* (S. 115–126). Springer-Verlag. https://doi.org/10.1007/11536482_10

Hummel, M., Garth, C., Hamann, B., Hagen, H. & Joy, K. I. (2010). IRIS: Illustrative Rendering for Integral Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, *16*(6), 1319–1328. https://doi.org/10.1109/TVCG.2010.173

Ivan Herman & David Duke (2001). Minimal Graphics. *IEEE Comput. Graph. Appl.*, *21*(6), 18–21. https://doi.org/10.1109/38.963456

Li, W., Agrawala, M., Curless, B. & Salesin, D. (2008). Automated generation of interactive 3D exploded view diagrams. *ACM Transactions on Graphics*, *27*(3), 1–7. https://doi.org/10.1145/1360612.1360700

Madsen, J. B., Tatzgern, M., Madsen, C. B., Schmalstieg, D. & Kalkofen, D. (2016). Temporal Coherence Strategies for Augmented Reality Labeling. *IEEE Transactions on Visualization and Computer Graphics*, *22*(4), 1415–1423. https://doi.org/10.1109/TVCG.2016.2518318

Mohr, P., Kerbl, B., Donoser, M., Schmalstieg, D. & Kalkofen, D. (2015). Retargeting Technical Documentation to Augmented Reality. In B. Begole (Hrsg.), *ACM Digital Library, Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (S. 3337–3346). ACM. https://doi.org/10.1145/2702123.2702490

Nandico, L. (2023). *Entwicklung eines 3D-Viewers zur dynamischen Visualisierung von Fahrzeugkomponenten im Bereich der Fahrzeug-Diagnose* [Bachelorarbeit]. Technische Hochschule Rosenheim, Rosenheim.

Qt Quick 3D 6.8.3. (2025, 1. April). https://doc.qt.io/qt-6.8/qtquick3d-index.html

Qt Quick 3D Architecture | Qt Quick 3D 6.8.3. (2025, 1. April). https://doc.qt.io/qt-6.8/qtquick3d-architecture.html

Rauner, F., Hitz, H., Spöttl, G. & Becker, M. (2002). *Neuordnung Kfz-Berufe - Berufsbildungsinstitut Arbeit und Technik - Europa-Universität Flensburg (EUF).* Institut Technik und Bildung (ITB) - Universität Bremen, Berufsbildungsinstitut Arbeit und Technik (biat) - Universität Flensburg. https://www.uni-flensburg.de/biat/forschung/forschungsprojekte/fahrzeug-metalltechnik/neuordnung-kfz-berufe

Reif, K. (2014). *Automobilelektronik: Eine Einführung für Ingenieure* (5., überarb. Aufl. 2014). *SpringerLink Bücher*. Springer Vieweg. https://doi.org/10.1007/978-3-658-05048-1

Reif, K. (2020). *Dieselmotor-Management Systeme, Komponenten, Steuerung und Regelung* (6., überarb. u. erw. Auflage 2020). *Bosch Fachinformation Automobil*. Springer Fachmedien Wiesbaden.

Rumpe, B. & Schiffers, J. (2006). Herausforderungen an die Diagnose Integration der Diagnose in die Steuergeräteentwicklung. *Zeitschrift für die gesamte Wertschöpfungskette Automobilwirtschaft*, *1*, 65–69.

Saito, T. & Takahashi, T. (1990). Comprehensible rendering of 3-D shapes. *ACM SIGGRAPH Computer Graphics*, *24*(4), 197–206. https://doi.org/10.1145/97880.97901

Shader - OpenGL Wiki. (2025, 14. April). https://www.khronos.org/opengl/wiki/Shader

Sieger, D. & Botsch, M. (2023). *The Polygon Mesh Processing Library*. https://github.com/pmp-library/pmp-library

Vasantha, G., Purves, D., Quigley, J., Corney, J., Sherlock, A. & Randika, G. (2021). Common design structures and substitutable feature discovery in CAD databases. *Advanced Engineering Informatics*, *48*, 101261. https://doi.org/10.1016/j.aei.2021.101261

Xian, Y., Fan, Y., Huang, Y., Wang, G., Tu, C., Meng, X. & Peng, J. (2020). Mesh Simplification With Appearance-Driven Optimizations. *IEEE Access*, *8*, 165769–165778. https://doi.org/10.1109/ACCESS.2020.2987939

Abbildungsverzeichnis

Tabellenverzeichnis

Appendix A Klassendiagramm der Projekt-Klassen im Doc3DCreator

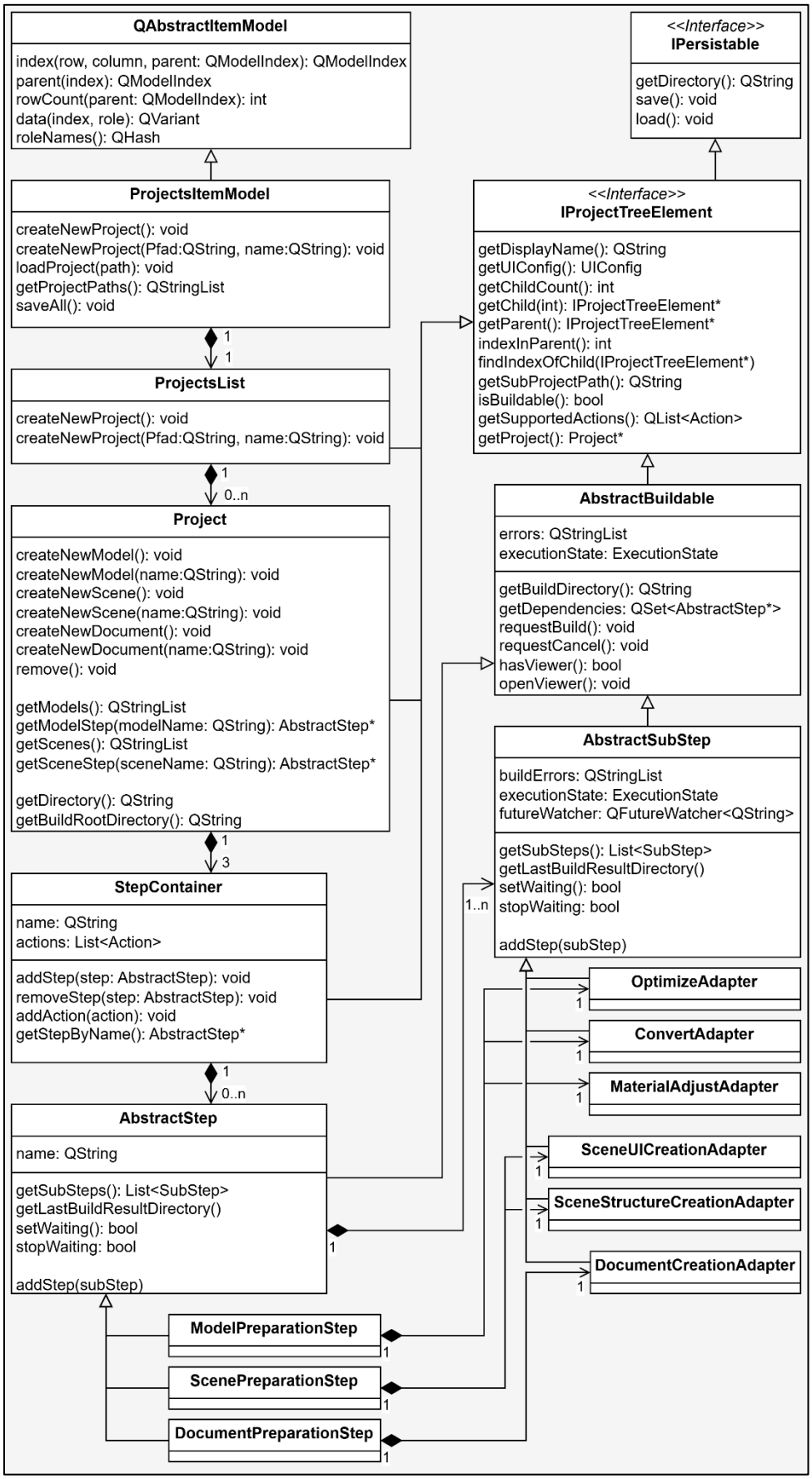


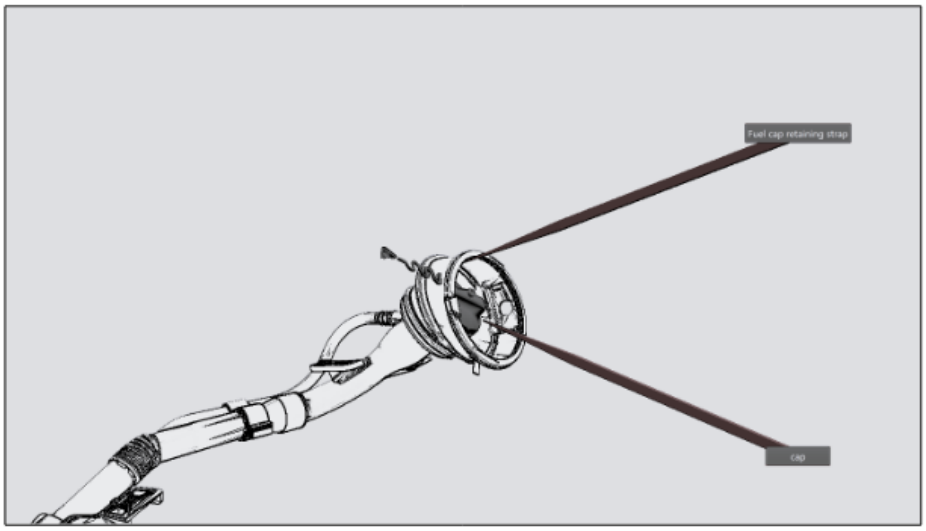
Abbildung 43: Klassendiagramm der Projekt-Klassen im Doc3DCreator

Appendix B Screenshot des Tank-Einfüllklappen-Dokuments

Tank
Tank Cap Assembly: Structure

Tank

Tank Cap Structure



The fuel tank cap assembly ensures a secure seal for the fuel reservoir while allowing user access. It consists of two main components: the primary cap unit and the fuel cap retaining strap. Together, these elements maintain containment integrity and operational convenience.

Main-Component 1: Primary Cap Unit

The primary cap is typically a circular, molded component with grip-enhancing features and a sealing interface. It connects to the filler neck via threads or bayonet coupling, often including a vapor relief valve in vented models. It may also include mounting points for a retaining strap.

Main-Component 2: Retaining Strap

The retaining strap is a flexible tether made of durable elastomer or plastic, securing the cap to the vehicle during refueling. It prevents loss and allows full range of motion without detachment, enduring repeated mechanical stress

Abbildung 44: Screenshot des einseitigen Dokuments mit der Tank-Einfüllklappen-Beschreibung