Fachhochschule Bielefeld
Campus Minden

Informatik, M.Sc.

Evaluierung einer cloud-native Entwicklungsumgebung für das modellbasierte Systems Engineering für Raumfahrtsysteme

Masterarbeit

Wintersemester	2022 / 2023
vorgelegt von:	Eller, Dennis
Matrikelnummer:	1109421
Abgabe am:	08.11.2022
Erstprüfer/in:	Prof. Dr. rer. nat. Jörg Brunsmann
Zweitprüfer/in:	Sascha Müller

Gleichstellungsvermerk

Die Inhalte der vorliegenden Arbeit beziehen sich in gleichem Maße auf Frauen und Männer. Aus Gründen der besseren Lesbarkeit wird jedoch die männliche Form für alle Personenbezeichnungen gewählt. Die weibliche Form wird dabei stets mitgedacht. Eine Ausnahme bilden die Inhalte, die ausdrücklich auf Frauen bezogen werden.

Danksagung

Zu Beginn möchte ich mich bei allen bedanken, die mich während meines Studiums unterstützt und motiviert haben.

Besonders möchte ich mich bei Prof. Dr. rer. nat. Jörg Brunsmann für die Betreuung der Masterarbeit bedanken. Durch die Vorlesung „Spezielle Gebiete zum Software Engineering“ wurde mein Interesse für das Themengebiet der Masterarbeit geweckt.

Ebenfalls möchte ich mich bei meinem Betreuer Sascha Müller bedanken, der mir seit meinem ersten Tag beim DLR bei Fragen mit Rat und Tat zur Seite stand.

Abschließend möchte ich mich bei meiner Familie und meiner Freundin Isabell bedanken, die mich während meines Studiums unterstützt haben.

Zusammenfassung

MBSE muss eine große Vielfalt an Konfigurationen unterstützen, da es mit einer Vielfalt an Software und Schnittstellen arbeitet. Der Ressourcenbedarf der verwendeten Software kann die Kapazitäten eines lokalen Computers überschreiten. Cloud-Computing verspricht, den steigenden Anforderungen gerecht zu werden. Es existieren MBSE-Tools für Raumfahrtsysteme, aber keines dieser Tools nutzt die Vorteile einer Cloud-Umgebung. In den letzten Jahren wurden Open-Source Projekte wie Eclipse Theia, EMF.Cloud und Eclipse Che entwickelt, die es möglich machen, eine cloud-native Entwicklungsumgebung für MBSE zu entwickeln. Es wurde aber noch keine Entwicklungsumgebung entwickelt, die die Vorteile der Cloud für Berechnungen nutzt. Für die Ermittlung der Anforderungen an ein MBSE-Tool in der Cloud wird ein Fragebogen mit zwei Runden durchgeführt. In der ersten Runde werden Anforderungen als User Stories gesammelt. In der zweiten Runde erfolgt eine Kategorisierung der Anforderungen nach dem Kano-Modell. Insgesamt wurden 52 Anforderungen identifiziert, wovon 19 als Begeisterungs-, 20 als Basis-, 3 als Leistungs- und 10 als unerhebliches Merkmal identifiziert wurden. Bei der Performanzanalyse zeigte sich, dass aufwendige Berechnungen ähnlich skalieren wie bei einer lokalen Ausführung. Werden simplere Berechnungen durchgeführt, ist die lokale Berechnung durch den Wegfall des Kommunikationsaufwandes deutlich schneller.

Abstract

MBSE must support a wide variety of configurations because it works with a variety of software and interfaces. The resource requirements of the software used can exceed the capacities of a local computer. Cloud computing promises to meet the increasing demands. MBSE tools exist for space systems, but none of these tools take advantage of a cloud environment. In recent years, open-source projects such as Eclipse Theia, EMF.Cloud and Eclipse Che have been developed which make it possible to develop a cloud-native development environment for MBSE. However, no development environment has yet been implemented that takes advantage of the cloud for computation. To determine the requirements for an MBSE tool in the cloud, a questionnaire with two rounds is conducted. In the first round, requirements are collected as user stories. In the second round, the requirements are categorized according to the Kano model. A total of 52 requirements were identified, of which 19 were identified as enthusiasm, 20 as basic, 3 as performance and 10 as irrelevant. The performance analysis showed that complex computations scale similarly to local execution. If simpler computations are performed, the local computation is significantly faster due to the elimination of the communication overhead.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AOCS	Attitude Determination and Control System
API	Application Programming Interface
ARCADIA	Architecture Analysis & Design Integrated Approach
CDF	Concurrent
CDM	Conceptual Data Model
CDP	Concurrent Design Platform
CE	Concurrent Engineering
CEF	Concurrent Engineering Facility
CNCF	Cloud Native Computing Foundation
ConCORDE	Concurrent Concepts, Options, Requirements and Design Editor
DLR	Deutsches Luft- und Raumfahrtzentrum
DSML	Domain Specific Modeling Language
ECSS	European Cooperation for Space Standardization
EMF	Eclipse Modelling Framework
EMOF	Essential MOF
ESA	European Space Agency
GSEL	Generic System Engineering Language

HTTP(S)	Hypertext Transfer Protocol (Secure)
IDM	Integrated Design Model
INVEST	Independent, Negotiable, Valuable, Estimatable, Small, Testable
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MBSE	Model Based Systems Engineering
MOF	Meta Object Facility
NASA	National Aeronautics and Space Administration
OCDT	Open Concurrent Design Tool
OMG	Object Management Group
RCP	Rich Client Platform
REST	Representational State Transfer
SSDE	Space Systems Design Editor
SSH	Secure Shell
SSRDB	Space System Reference Database
SSVT	Space Systems Visualization Tool
SVN	Subversion
SysML	Systems Modeling Language
UML	Unified Modeling Language
URN	User Requirements Notation

VCS	Version Control System
VDI	Virtual Desktop Infrastructure
VSD	Virtual Spacecraft Design
XMI	XML Metadata Interchange
XML	Extensible Markup Language

1 Einleitung

Die internationale Gesellschaft für Systems Engineering (INCOSE) definiert Systems Engineering als einen transdisziplinären und integrativen Ansatz, bei dem komplexe Systeme[1] entwickelt werden. Dabei beinhaltet das Systems Engineering die Realisierung, Nutzung und Ausmusterung der Systeme. [1] Die Notwendigkeit des Systems Engineerings besteht darin, dass die Systeme durch den wachsenden Konkurrenzdruck und Bedarf an dem Einsatz neuer Technologien immer komplexer werden. Die Spezifikationen der Systeme werden in Dokumenten mit allen Stakeholdern ausgetauscht. Dazu zählen grafische Darstellungen, tabellarische Daten und Diagramme. [2]

Durch immer komplexer werdende Systeme ist das Verwalten der Informationen über Dokumente nicht mehr tragbar. Die Dokumente müssen manuell kontrolliert und aktualisiert werden. Diagramme, die das System beschreiben, werden als Datei angehängt und als Bild in die Dokumente aufgenommen. Die Informationen sind über verschiedene Dateien und Dokumente verteilt. Dies erschwert die Rückverfolgbarkeit und Aussagen über den Einfluss von Änderungen. Das resultiert wiederum in einer mangelhaften Synchronisierung zwischen den Anforderungen, dem Systemdesign und den detaillierten Entwürfen bei Software, Elektronik oder Mechanik. [2]

Eine Möglichkeit, mit der steigenden Komplexität umzugehen, ist der Einsatz vom modellbasiertem Systems Engineering (MBSE) [3]. Modellbasierte Ansätze sind im elektrischen oder mechanischen Design schon lange Standard. Aber auch im Softwarebereich findet Modellierung spätestens mit dem Aufkommen von UML (Unified Modeling Language) immer mehr Verbreitung. Der modellbasierte Ansatz gewinnt auch beim Systems Engineering an Relevanz. Es wird erwartet, dass der modellbasierte Ansatz auch im Systems Engineering zum Standard wird. [2]

Ein Beispiel für komplexe Systeme sind Raumfahrtsysteme, wo hohe Kosten und Risiko vorherrschen. Der Lebenszyklus von Raumfahrtsystem besteht aus mehreren Phasen.

Beginnend mit der Phase 0 (Pre-A bei der NASA) setzt sich der Lebenszyklus mit den Phasen A, B, C, D, E und F fort. Jede Phase beinhaltet Aufgaben, die erfüllt werden müssen, sowie die zu liefernden Ergebnisse. In der Phase 0 werden mögliche Missionen ausgearbeitet. In der

[1] Ein komplexes System besteht aus mehreren Komponenten, die miteinander interagieren

Phase A werden Machbarkeitsstudien und in Phase B detaillierte Entwurfsarbeiten durchgeführt. In der Phase C folgen Entwicklungsarbeiten sowie Montage, Integration und Testen von Prototypen, die in Phase D in dem finalen System enden. Schlussendlich befindet sich das Raumfahrtsystem in Phase E in Betrieb und wird in Phase F entsorgt. [4], [5] Der konzeptionellen Designphase wird eine besondere Wichtigkeit zugesprochen, da diese den Großteil der Kosten für das Raumfahrtsystem bestimmen [6].

Die Phase A wird oft während einer Designstudie in speziellen Einrichtungen (Concurrent Design Centers) wie zum Beispiel der Concurrent Engineering Facility (CEF) am Deutschen Zentrum für Luft- und Raumfahrt (DLR) durchgeführt [7]. Das Concurrent Engineering[2] ist ein Vorgehen, bei dem die verschiedenen Aktivitäten beim Systems Engineering integriert und möglichst parallel statt nacheinander durchgeführt werden [8].

Eine Designstudie dauert eine bis drei Wochen, in der alle Stakeholder (z.B. Kunden und Ingenieure) eng zusammenarbeiten. Zum Datenaustausch zwischen den Teilnehmern wird ein Datenmodell verwendet. Das Datenmodell ist die zentrale „source of truth“, über die alle Stakeholder Informationen erhalten und bereitstellen. Durch das Datenmodell können Änderungen an alle Teilnehmer weitergegeben werden. Außerdem können Analysen „on the fly“ durchgeführt werden, wodurch Auswirkungen von Änderungen evaluiert werden können. [7]

Die Arbeit befasst sich mit der Evaluierung einer cloud-native Entwicklungsumgebung für das modellbasierte Systems Engineering für Raumfahrtsysteme. Die zu erstellende Entwicklungsumgebung soll auf Basis von aktuellen Technologien entwickelt werden.

1.1 Zielsetzung

Ziel der Arbeit ist zu evaluieren wie Cloud-Computing bei den genannten Herausforderungen helfen kann. Dafür sollen zunächst Anforderungen an ein Cloud-MBSE-Tool identifiziert werden. Im Anschluss soll evaluiert werden, welche Anforderungen sich mit aktuellen Technologien realisieren lassen. Dabei soll der Fokus der Implementierungen auf den Anforderungen liegen, die mit aktuellen Technologien erfüllt werden können. Nicht-Ziel der Arbeit ist das Implementieren eines vollumfänglichen MBSE-Tools.

[2] Auch Simultaneous oder Integrated Engineering genannt

1.2 Vorgehen

Zu Beginn wird die Problemstellung (Kapitel 2) dargestellt, die aufzeigt, welche Herausforderungen bei dem Umstieg vom klassischen Systems Engineering auf Model-based Systems Engineering und Concurrent Engineering bestehen.

Darauffolgend werden die Grundlagen der Arbeit gelegt (Kapitel 3). Dazu zählen Containerisierung (Kapitel 3.1), Container-Orchestrierung (Kapitel 3.2) sowie Cloud-Computing und cloud-native (Kapitel 3.3).

Danach wird der Stand der Technik dargestellt (Kapitel 4). Dazu zählen Lösungen für MBSE-Tools für Raumfahrzeuge (Kapitel 4.1) sowie web- und cloudbasierte Entwicklungsumgebungen (4.2). Im Anschluss wird dargestellt, was für eine cloud-native Entwicklungsumgebung fehlt (Kapitel 4.3).

Anschließend wird das Konzept vorgestellt (Kapitel 5). Dazu wird zunächst die Methodik der Anforderungsermittlung und Kategorisierung dargestellt (Kapitel 5.1). Nachdem die Anforderungen feststehen, werden die Teilkonzepte zur Erfüllung der Anforderungen vorgestellt. Zu den Teilkonzepten zählen das Cluster (Kapitel 5.2), die Workspaces (Kapitel 5.3), die Entwicklungsumgebung und Modellierung (Kapitel 5.4) sowie das Concurrent Engineering (Kapitel 5.5). Daran anknüpfend wird das Konzept implementiert (Kapitel 6). Die Implementierung teilt sich in die Umsetzung des Clusters (Kapitel 6.1), der Workspaces (Kapitel 6.2), der Entwicklungsumgebung und Modellierung (Kapitel 6.3) sowie dem Concurrent Engineering (Kapitel 6.4) auf.

Im nächsten Schritt wird die Implementierung evaluiert (Kapitel 7). Dafür wird zunächst eine Metadokumentation (Kapitel 7.1) dargestellt, wonach die Ergebnisse zu den einzelnen Forschungsfragen dargestellt werden (Kapitel 7.2). Im Anschluss werden die Ergebnisse diskutiert und mögliche Gefährdungen der Validität werden betrachtet (Kapitel 7.3). Zum Abschluss wird die Arbeit zusammengefasst und ein Ausblick auf weitere Arbeiten gegeben (Kapitel 8).

2 Problemstellung

Systeme werden durch wachsenden Konkurrenzdruck und steigenden Bedarf an neuen Technologien immer komplexer. Gleichzeitig soll die Qualität neuer Systeme erhöht und die Entwicklung beschleunigt werden[9]. Dadurch besteht die Anforderung, dass Systeme nebenläufig entwickelt werden müssen (durch z.B. Concurrent Engineering). Der dokumentenbasierte Ansatz hat sich für komplexe Systeme als nicht geeignet herausgestellt [2].

Ein modellbasierter Ansatz sollte bevorzugt werden, insbesondere, wenn die Software für das Systems Engineering zu groß für einen einzelnen Computer wird oder Sicherheit oder Zugriffbeschränkungen von Relevanz sind [10].

Der Einsatz eines modellbasierten Ansatzes hat zur Folge, dass das Modell des Systems zum Beispiel zentral in einer Datenbank gehalten werden muss. Auf der einen Seite muss das Modell für alle Teilnehmer verfügbar sein, auf der anderen Seite müssen unzulässige Zugriffe oder inkonsistente Zustände verhindert werden. Es ist notwendig, dass Änderungen am Systemmodell zwischen allen Teilnehmern synchronisiert werden. Des Weiteren müssen von dem System verschiedene Versionen und Produktlinien verwaltet werden (Product Lifecycle Management).

Raumfahrtprojekte besitzen eine Lebensspanne von bis zu mehreren Jahrzehnten, wodurch es nahezu unmöglich ist, ein Datenmodell, das alle in Zukunft benötigten Anwendungsfälle abdeckt, im Voraus zu definieren. Daher ist es notwendig, dass die erstellten Modelle im Laufe des Lebenszyklus erweitert werden können. [11] Dabei werden für die Definition von Modellen verschiedene Abstraktionsebenen sowie für die Modellierung verschiedene Sichten benötigt. Sichten stellen dabei zum Beispiel einen domänenspezifischen Ausschnitt des gesamten Systemmodells dar [12].

In der Praxis geschieht die Entwicklung immer noch oft seriell und in Isolation. Das bedeutet, dass jede Abteilung für sich und nicht nebenläufig zu anderen Abteilungen arbeitet. Es werden zwar anstelle von Dokumenten Modelle ausgetauscht, jedoch wird weiterhin nicht an einem gemeinsamen Datenmodell gearbeitet. Ein Grund dafür ist, dass System Engineers Modellierungssprachen verwenden, die nicht alle Fachbereiche verstehen (z.B. SysML). [13] Es deutet darauf hin, dass formalisierte Modellierungssprachen weder ausreichend aussagekräftig noch einfach zu verwenden sind, um beim Concurrent Engineering zum Einsatz zu kommen. Das stellt angesichts der begrenzten Zeit für die Concurrent Engineering Studien und dem heterogenen Wissenshintergrund der Teilnehmer ein Problem dar. [14] Erschwerend kommt hinzu, dass regelmäßig neue Teilnehmer an den Studien teilnehmen und diese noch keine

Erfahrungen mit den verwendeten Werkzeugen besitzen, wodurch eine einfache Verwendung ohne lange Konfiguration und Installation benötigt wird [12], [13]. Ein weiterer Grund für die serielle und isolierte Entwicklung ist, dass Werkzeuge verwendet werden, ohne Möglichkeiten zur Kollaboration zu nutzen [13].

MBSE muss eine große Vielfalt von Konfigurationen unterstützen, da es mit einer Vielfalt an Geräten, Software und Schnittstellen arbeitet [16]. Für MBSE werden verschiedene domänenspezifische Werkzeuge zur Modellierung, Berechnung und Simulation benötigt [17]. Es besteht die Anforderung, dass die Ingenieure möglichst viele ihrer Werkzeuge mit dem Systemmodell verbinden können [16], [17], was die Komplexität erhöhen und die Wartbarkeit verringern kann. Des Weiteren müssen die Werkzeuge bei den Ingenieuren in der passenden Version und der richtigen Lizenz installiert sein.

Im Kontext der CEF bedeutet das vor dem Start einer Designstudie, dass die benötigte Software inklusive aller Laufzeitabhängigkeiten auf den Rechnern installiert sein muss. Manche Werkzeuge benötigen ein bestimmtes Betriebssystem und die Lizenzen der Werkzeuge können das Teilen oder simultane Verwenden der Werkzeuge erschweren.

Bei einer Befragung von Concurrent Design Centers wurde die Wartung der Systeme als größte Herausforderung für den Betrieb der Infrastruktur für Concurrent Engineering bezeichnet. Verstärkt wird dies durch den Trend, dass Studienteilnehmer vermehrt ihre eigenen Geräte zu den Studien mitbringen. [14] Durch die Corona-Pandemie werden Design-Studien auch virtuell und damit nicht vor Ort durchgeführt [14], [18], was ebenfalls die Wartung erschwert. Die Corona-Pandemie beschleunigt dabei diesen bereits bestehenden Digitalisierungsprozess, der mit verteilter Produktentwicklung einhergeht [18].

Für Simulationen und Berechnungen werden leistungsfähige Computer benötigt, da deren Bedarf an Systemressourcen exponentiell mit den zu verarbeitenden Daten zunimmt [16]. Der Bedarf an Systemressourcen kann die Kapazitäten von lokalen Computern überschreiten. MBSE-Nutzer sind meistens keine IT-Unternehmen und benötigen deswegen eine simple, aber skalierbare Lösung [16].

Trotz der kontinuierlichen Weiterentwicklung von MBSE-Software werden noch erhebliche Verbesserungen im Bereich von simultanem Bearbeiten, Interoperabilität oder Integration mit anderer Software erwartet [14]. Bei der Einführung von MBSE sollte das klassische Vorgehen mit Dokumenten schrittweise abgelöst werden, um die Akzeptanz bei den Anwendern zu erhöhen, weswegen auf Kompatibilität zu aktuellen Praktiken und Werkzeugen geachtet werden muss [16].

Klassische Desktopanwendungen werden diesen steigenden Anforderungen nicht mehr gerecht. Cloud-Computing verspricht, den Anforderungen gerecht zu werden, indem es zum Beispiel für Simulationen oder Berechnungen nahezu unendliche Systemressourcen bereitstellen kann.

Des Weiteren kann über die Cloud Software bereitgestellt werden, wodurch eine lokale Installation von Software nicht mehr notwendig ist. Über die Cloud kann zudem das Systemmodell zentral gehalten und über eine Schnittstelle zur Verfügung gestellt werden. [16]

Wenn das Systemmodell in der Cloud abgelegt wird, sind Punkte wie Datenhoheit und Zugriffsmanagement zu beachten [16], da theoretisch von überall aus auf das Systemmodell zugegriffen werden kann. Bei der Produktentwicklung sollten die Daten vor dem Zugriff Dritter geschützt sein. Insbesondere in der Raumfahrt unterliegen viele Informationen der Exportkontrolle. Des Weiteren sollte ein „Vendor-Lock-In“ vermieden werden, bei dem eine Abhängigkeit vom Cloud-Anbieter besteht, indem Cloud-Anbieter-spezifische Lösungen in Anspruch genommen werden, die einen Wechsel erschweren.

Durch die Bereitstellung einer MBSE-Anwendung in der Cloud kann das Systemmodell über eine Schnittstelle zur Verfügung gestellt werden. Dabei kann die Schnittstelle von beliebigen Anwendungen verwendet werden. Dazu zählen bereits existierende Desktopanwendungen oder Webanwendungen. Bei dem Einsatz eines Webbrowsers benötigen die Anwender keine lokale Installation, was die Einstiegsschwelle vom Einsatz von MBSE-Anwendungen verringert und den Einsatz auf beliebigen Geräten möglich macht. Durch diese Interoperabilität ist auch ein schrittweiser Umstieg möglich.

Ziel der Arbeit ist, zu evaluieren, wie Cloud-Computing bei den genannten Herausforderungen helfen kann. Dafür sollen zunächst Anforderungen an ein Cloud-MBSE-Tool identifiziert werden. Im Anschluss soll evaluiert werden, welche Anforderungen sich mit aktuellen Technologien realisieren lassen. Im Rahmen der Arbeit werden dabei folgende Forschungsfragen untersucht:

- F1: Welche Anforderungen stellt das Concurrent Engineering mit dem modellbasierten Systems Engineering für den Betrieb in der Cloud?
- F2: Wie gut unterstützen aktuell verfügbare Tools die Prozesse für Raumfahrtsysteme bei Produktlinien, Versionierung, Synchronisierung und Privacy sowie das Abbilden von verschiedenen Sichten bzw. Abstraktionsebenen auf das Systemmodell in der Cloud?
- F3: Wie skalieren Simulationen und Berechnungen im MBSE-Kontext in einer Cloud?

3 Grundlagen

In diesem Kapitel werden die Grundlagen dargestellt, die für das Verständnis der Arbeit wichtig sind. Dazu zählt die Definition von Containerisierung, Container-Orchestrierung sowie Cloud-Computing und cloud-native.

3.1 Containerisierung

Ein Software-Container stellt eine Laufzeitumgebung für Anwendungen zur Verfügung, die sich das Host-Betriebssystem, Binärdateien und Bibliotheken mit anderen Containern teilen. Die Container laufen isoliert voneinander und besitzen weniger Overhead als virtuelle Maschinen. [19] Ein Vergleich zwischen virtueller Maschine und Container ist in Abbildung 3.1 dargestellt.

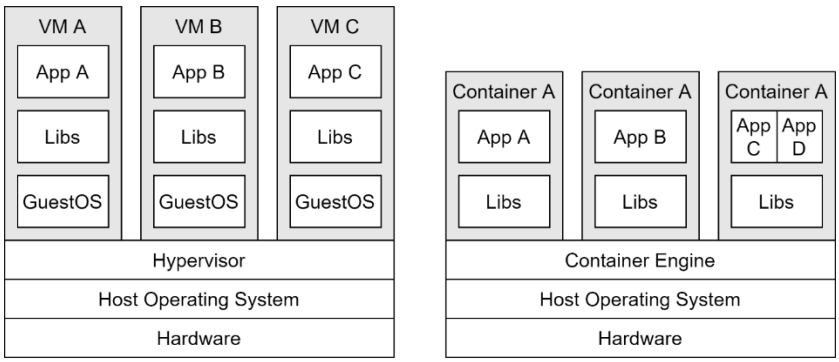


Abbildung 3.1: Vergleich virtuelle Maschine und Container, nach [20]

Während bei einer virtuellen Maschine die Hardware virtualisiert wird, erfolgt die Virtualisierung bei Docker auf Betriebssystemebene. Dadurch sind Docker Container leichtgewichtiger als virtuelle Maschinen. [20] Eine Möglichkeit zur Verwendung einer Containerumgebung ist Docker. Docker basiert auf den Funktionen von Linux-Containern. Docker Container werden aus Docker Images erstellt und stellen eine ausführbare Instanz des Docker Images dar. Docker Images werden über ein Dockerfile beschrieben, die alle benötigten Abhängigkeiten definiert.[21]

3.2 Container-Orchestrierung

Die Container-Orchestrierung ermöglicht es, Cloud- und Anwendungsanbietern zu definieren, welche Konfiguration für Anwendungen aus mehreren Containern ausgewählt, bereitgestellt, überwacht und dynamisch konfiguriert wird. Zu den Aufgaben der Container-Orchestrierung

gehören die Überwachung und Begrenzung des Ressourcenverbrauchs, Auswahl eines Rechenknotens für den Betrieb der Container und die Verteilung der Last auf verschiedene Container-Instanzen. Weitere Aufgaben sind das Neustarten von abgestürzten Containern, das Vorhalten von Containern für erhöhte Fehlertoleranz sowie das Skalieren der Containeranzahl in Abhängigkeit der Last. [22] Container haben es ermöglicht, eine Anwendung in kleine unabhängige Services aufzuteilen. Dieser architekturbezogene Ansatz wird Microservices genannt. [23] In Abbildung 3.2 ist ein Vergleich zwischen einer Monolithischen- und einer Microservices-Architektur dargestellt.

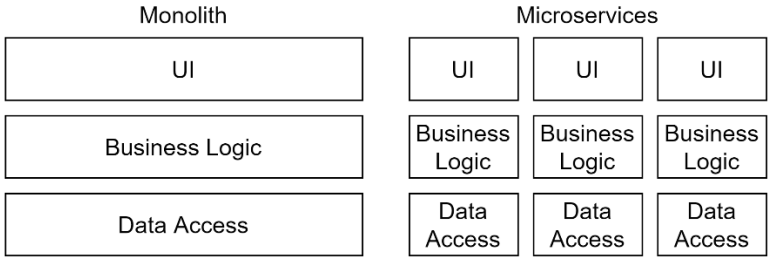


Abbildung 3.2: Vergleich Monolith und Microservices, nach [24]

Eine Möglichkeit zur Container-Orchestrierung ist Kubernetes. Kubernetes ist ein Open-Source System, dass das Bereitstellen, Verwalten und Skalierung von containerisierten Anwendungen in verteilten Clustern automatisiert. [23]

3.3 Cloud-Computing und cloud-native

Cloud-Computing (kurz Cloud) beschreibt ein Modell, das Rechenressourcen (z.B. Netzwerke, Server, Speicher, Anwendungen und Dienste) über das Netzwerk bedarfsgerecht bereitstellt. Dieses Modell besteht aus fünf Charakteristiken, drei Servicemodellen und vier Liefermodellen. Zu den Charakteristiken zählt, dass die Rechenkapazitäten automatisch und ohne menschliche Interaktion bereitgestellt werden können. Des Weiteren werden die Rechenkapazitäten über das Netzwerk zur Verfügung gestellt. Die Rechenkapazitäten werden für mehrere Kunden verwendet, die dynamisch nach Bedarf zugeteilt werden. Dabei können die Rechenkapazitäten elastisch und automatisch provisioniert und freigegeben werden, um die Rechenkapazitäten dem Bedarf anzupassen. Cloudsysteme werden nach Nutzung (Verwendung Speicher, Rechenkapazitäten oder Netzwerkverkehr) abgerechnet. [25]

Das unterste Servicemodell ist Infrastructure as a Service (IaaS), bei der virtuelle Maschinen und Container mit Prozessor, Arbeitsspeicher, Speicher und Netzwerkkomponenten

bereitgestellt werden. Eine Plattform für die Ausführung von beliebigen Anwendungen wird bei Platform as a Service (PaaS) angeboten. Es werden das dafür benötigte Netzwerk, Server, Betriebssystem und Speicher bereitgestellt. Bei Software as a Service (SaaS) werden Anwendungen durch einen Cloud-Provider zur Verfügung gestellt, die zentral verwaltet und über das Internet bereitgestellt werden. [25]

Bei den Liefermodellen von Cloud-Computing wird zwischen Private Cloud, Community Cloud, Public Cloud, und Hybrid Cloud unterschieden. Bei der Private Cloud sind die Rechenressourcen ausschließlich für eine Organisation bestimmt und werden entweder selbst oder von einer dritten Partei oder einer Kombination von beiden vor Ort (On-Premise) oder extern (Off-Premise) betrieben. Community Clouds werden exklusiv für eine Gruppe Organisationen betrieben provisioniert und entweder On-Premise oder Off-Premise betrieben. Der Öffentlichkeit zugänglich sind Public Clouds, die über das Internet zur Verfügung gestellt und beim Cloud-Anbieter betrieben werden. Bei Hybrid Clouds handelt es sich um eine Kombination der vorgestellten Liedermodelle. [25]

Nach der Definition der Cloud Native Computing Foundation (CNCF) ermöglichen cloud-native Technologien den Unternehmen, skalierbare Anwendungen in einer dynamischen Umgebung wie der Cloud zu implementieren und zu betrieben. Diese dynamischen Umgebungen können öffentliche, private und hybride Clouds sein. [26] Cloud-native Anwendungen basieren auf einer Microservice-Architektur, dessen Services in Container verpackt sind und von einem Orchestrator verwaltet werden [23]. Dadurch sind cloud-native Anwendungen lose gekoppelte Systeme, die belastbar, handhabbar und beobachtbar sind. Des Weiteren sind Cloud-native Anwendungen herstellerneutral, wodurch sie von einem Cloudanbieter zu einem anderen portiert werden können [26].

4 Stand der Technik

In diesem Kapitel werden bereits existierende MBSE-Tools vorgestellt und auf ihre Schnittstellen untersucht. Darauffolgend werden bereits existierende Web- und Cloud-Entwicklungsumgebungen dargestellt. Zum Schluss wird dargestellt, was für eine cloud-native Entwicklungsumgebung für das modellbasierte Systems Engineering für Raumfahrtsysteme fehlt.

In einem akademischen Kontext gibt es mehrere Gründe, Open-Source-Software zu verwenden. Zu den Gründen zählen erhöhte Zuverlässigkeit, Anpassbarkeit, Innovation, Zusammenarbeit sowie geringere Kosten. [27] Im Rahmen der Arbeit fokussiert sich dieses Kapitel auf Open-Source-Lösungen, die eine komplett freie Lizenz besitzen. Miteingeschlossen werden Lizenzen, die Lösungen für Personen zugänglich machen, die für ein europäisches Unternehmen arbeiten, das zu den Mitgliedstaaten oder Kooperationsstaaten der europäischen Weltraumorganisation (ESA) gehört.

4.1 MBSE-Tools

Im Rahmen der Arbeit wird sich auf MBSE-Tools für Raumfahrtsysteme fokussiert. MBSE ist kein neues Thema innerhalb der europäischen Raumfahrtgemeinschaft. Verschiedene Projekte bieten Lösungen zur Unterstützung der Entwicklung eines Raumfahrtsystems für die verschiedenen Phasen des Lebenszyklus an. Diese werden oft als Datenbanken bezeichnet und stellen Software zur Verfügung, mit der ein Systemmodell eines zu entwickelnden Raumfahrtsystems erstellt werden kann. Die meisten Datenbanken ermöglichen die gemeinsame Nutzung des Modells mit mehreren Beteiligten über ein zentrales Repository, das normalerweise mit einem Versionskontrollsystem (VCS) verbunden ist. [11]

4.1.1 Virtual Spacecraft Design (VSD)

Virtual Spacecraft Design (VSD) ist ein von der ESA geleitetes Projekt, dessen Ziel es war, die Organisation von technischen Daten auf Systemebene zu verbessern und einen reibungslosen Austausch von wichtigen technischen Parametern zwischen verschiedenen technischen Bereichen und ihrem jeweiligen Modellen zu ermöglichen [28]. Dabei fokussiert sich VSD auf die Phasen B, C und D des Raumfahrtlebenszyklus [11]. Lizenziert wird VSD über die ESA Software Community License Type 2 und Type 3 [29]. Damit ist der Quellcode sowie die Software nur für Personen zugänglich, die für ein europäisches Unternehmen arbeiten, das zu den Mitgliedstaaten oder Kooperationsstaaten der ESA gehören [30].

VSD sollte den Mehrwert und die Machbarkeit von MBSE in der Weltraumindustrie aufzeigen. Dabei lässt es sich mit anderen Werkzeugen und Datenquellen integrieren und bietet verschiedene Sichten wie Diagramme sowie Baum- und Tabelleneditoren. [31]

Im Rahmen des VSD-Projekts wurden prototypisch eine Reihe von Software-Werkzeugen für den Einsatz von MBSE bei Raumfahrzeugen entwickelt [32]. Das Virtual Spacecraft Engineering Environment (VSEE)-Toolset basiert auf der Eclipse IDE, wodurch es durch Plugins erweitert werden kann. Das VSEE-Toolset besteht aus folgenden Komponenten [33]:

- Space Systems Design Editor (SSDE): Ein Modell-Editor zum Bearbeiten des Systemmodells
- Space Systems Visualization Tool (SSVT): Ein Werkzeug zu Visualisierung der Modelle
- Space System Reference Database (SSRDB): Eine Datenbank für das Management von Repositories, die zeitgleichen Zugriff auf das Datenmodell ermöglichen

SSDE basiert auf EMF sowie Eclipse und beinhaltet unter anderem Baum- sowie Diagramm-Editoren. Im Kontext von Concurrent Design können Unterschiede von verschiedenen Modellen dargestellt und zusammengefügt werden. Des Weiteren ist es möglich, in der Oberfläche Anmerkungen an das Modell für manuelle Prüfungen hinzuzufügen und Konsistenzprüfungen durchzuführen. SSDE kann eigenständig oder mit SSRDB integriert verwendet werden, wenn mehrere Nutzer auf das gleiche Model zugreifen sollen. [34]

SSRDB dient als Drehscheibe für das Model-Management und ermöglicht den Austausch von Daten zwischen den VSEE-Komponenten und anderen Datenquellen. Auch ein Import und Export der Daten ist möglich. Dabei erlaubt es den Zugriff von mehreren Stellen durch Editierungssessions und Ownership Tracking. Die Persistierung erfolgt dabei über ein Repository und über XMI. [35] Das Metamodell von VSEE wurde von der European Cooperation for Space Standardization (ECSS) unter dem Namen ECSS ETM 10-23 standardisiert. Implementiert wurde es mit dem Eclipse Modelling Framework (EMF). [36]

VSD ermöglicht es, das System hierarchisch aufzugliedern und zu strukturieren. Informationen werden in vordefinierten Objekten wie Zustandsautomaten, Schnittstellen oder elektrischen Verbindungen gespeichert. Benutzerdefinierte Daten können in Engineering Categories gespeichert werden. Dabei verfügt VSD über Konfigurationskontrolle-Mechanismen zum Wiederverwenden von Informationen in späteren Lebensphasen (z.B. Masse eines Sensors). [37] Im Modellierungsprozess wird das System von einer generischen Sicht zu einem immer konkreter werdenden Modell überführt. Dabei beginnt es mit Categories, die eine Menge von Eigenschaften (Properties) definiert. Eine Category kann einen generischen Typen für ein Element

repräsentieren, dass durch die Properties beschrieben wird. Eine Category kann dabei von anderen Categories erben. Zum Modellieren werden zunächst ElementDefinitions verwendet, die entweder eigene Properties oder Properties von Categories mit Werten versehen. Des Weiteren enthalten ElementDefinitions neben der Struktur die Schnittstellen und das Verhalten des Elements. [33]

4.1.2 Open Concurrent Design Tool (OCDT)

Das Open Concurrent Design Tool (OCDT) wurde von der ESA in Kooperation mit der Industrie entwickelt. Beide waren als Nachfolger des Integrated Design Models geplant (IDM), das aus einer Reihe von Excel-Spreadsheets bestand. Allerdings limitierte der Excel-Ansatz die Möglichkeiten, was realisiert werden konnte, und erschwerte die Wartung und Integration mit anderen Werkzeugen. Es basiert auf dem ECSS-E-TM-10-25 Datenmodell und baut auf den Lektionen von IDM auf.[7] OCDT besitzt ein Client-Server-Architektur mit einem Datenbankserver und basiert auf Open-Source-Komponenten. OCDT selbst wird über die ESA Community Open Source License vertrieben. OCDT besitzt Methoden und Werkzeugkästen für die Integration von domänenspezifischen Werkzeugen und kann in verteilten Designsitzungen verwendet werden.[30] Die Architektur von OCDT ist in Abbildung 4.1 dargestellt.

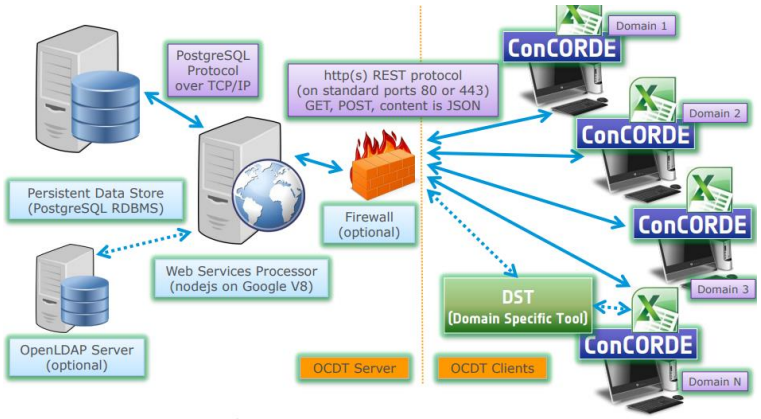


Abbildung 4.1: OCDT Architektur [38]

Die Architektur teilt sich in eine Server- und Clientseite auf. Auf der Serverseite stellt der Web Services Processor eine sprachunabhängige REST-API über HTTP(S) zur Verfügung, die in CoffeeScript und JavaScript implementiert ist und über Node.js ausgeführt wird. Auf dieser Ebene wird eine rollenbasierte Berechtigungsverwaltung implementiert. Der Web Services Processor kann optional hinter eine Firewall geschaltet werden. Als persistenter Datenspeicher wird PostgreSQL verwendet, wo für jedes Engineering Model eine eigene Datenbank

verwendet wird. Optional kann zur Authentifizierung und Autorisierung ein OpenLDAP (Open Lightweight Directory Access Protocol) Server verwendet werden. [30]

Auf der Clientseite kann das Microsoft Excel 2019 Add-In ConCORDE (Concurrent Concepts, Options, Requirements and Design Editor) verwendet werden. Alternativ kann die REST-API durch ein beliebiges domänenspezifisches Tool genutzt werden. Das Excel-Add-In ConCORDE verwendet wie IDM für jede Disziplin eine eigene Arbeitsmappe. Des Weiteren existieren Software Development Kits (SDKs) für die Programmiersprachen C# und Python. Die Architektur ermöglicht es, Updates zwei Mal pro Minute an 25 gleichzeitig verbundenen Nutzern auszuliefern. [30] Neben einer klassischen Installation wird OCDT auch als Docker-Container angeboten [38]. OCDT fokussiert sich auf die frühen Lebensphasen des Raumfahrtsystemlebenszyklus (0, A, B) [30], unterstützt aber nicht komplett den Übergang von den vorherigen Phasen in Phase B [11]. Wie bei VSD unterstützt OCDT das Definieren von benutzerdefinierten Eigenschaften über Engineering Categories [11].

4.1.3 COMET

Concurrent Design Plattform (CDP) ist die kommerzielle Version von OCDT. CDP kann auch von Personen genutzt werden, die nicht für eine Organisation arbeiten, die zu den ESA-Mitgliedstaaten oder Kooperationsstaaten gehört. [39] CDP wurde weiterentwickelt und ist unter dem Produktnamen „COMET“ verfügbar [40]. Trotz der Umbenennung wird in der Dokumentation, im Source-Code und in der Anwendung selbst meistens von CDP gesprochen.

Änderungen der Nutzer werden nicht automatisch mit den anderen Nutzern synchronisiert, sondern in einer Routine zwischengespeichert. Die Änderungen werden allen Nutzern angezeigt, müssen aber von einem Nutzer mit höherer Autorität manuell genehmigt und veröffentlich werden. Ein Beispiel für so einen Nutzer ist ein Teamleiter. Dadurch werden ein ständiges Ändern des gemeinsamen Systemmodells sowie Performanzprobleme vermieden. [40]

Das Design des Systems ist in einem ProductTree gespeichert. Der ProductTree besteht aus Komponenten und Sub-Komponenten. Die Sub-Komponenten beinhalten Parameter, die das System beschreiben. Dabei wird festgelegt, welche Parameter von welchem Nutzer verändert werden können. [40] Die Typen der Komponenten und Sub-Komponenten werden dabei als ElementDefinition definiert und als ElementUsage im ProductTree instanziiert. Dabei können eigene ElementDefinitions sowie auch Typen der Parameter erstellt werden. COMET verfügt außerdem über eine Reference Data Library, in der bereits vordefinierte Komponenten enthalten sind. [41]

Die Systemkomponenten können mit Anforderungen verknüpft werden und durch Kommentare annotiert werden. Änderungen werden in einer Revisionshistorie aufgezeichnet. Des Weiteren können Python-Skripte verwendet werden, um mit dem Systemmodell zu arbeiten. Regeln zur Validierung des Systemmodells können in C# definiert werden. COMET steht als Excel-Add-In, Webservice und Desktopanwendung zur Verfügung. Der Webservice stellt eine REST-API zur Verfügung und kann als Docker-Container deployt werden. Außerdem kann COMET über einen Plugin-Manager erweitert werden. [42] Die Architektur von COMET ist in Abbildung 4.2 dargestellt.

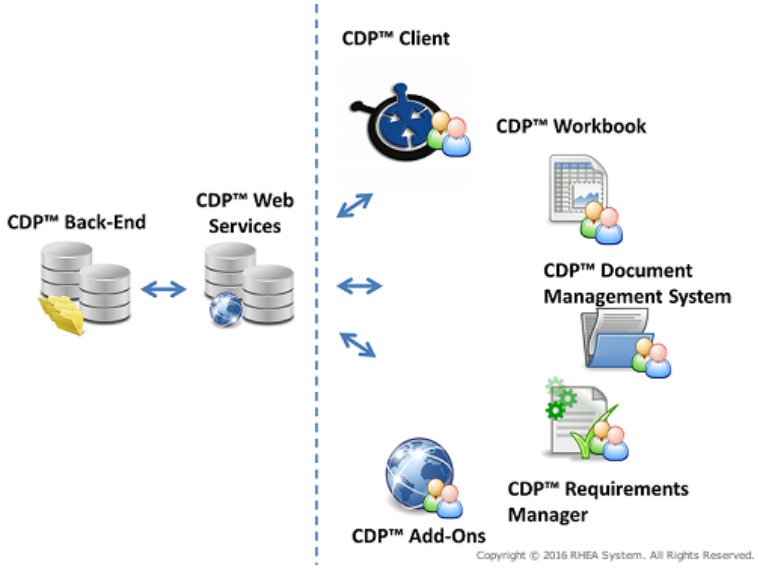


Abbildung 4.2: COMET Architektur [43]

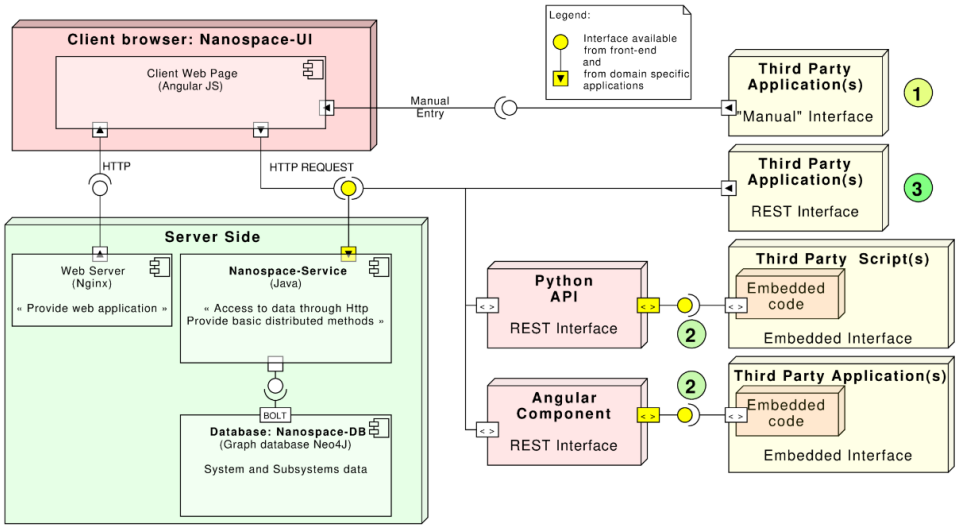
CDP besteht aus einer Client- und Serverseite. Die Serverseite setzt sich aus dem CDP Backend und den CDP Web Services zusammen. Auf der Clientseite sind die wichtigsten Komponenten der CDP Client, das CDP Workbook und die CDP Add-Ons. CDP Add-Ons können wie der CDP Requirements Manager oder das CDP Document Management System in COMET integriert werden oder komplett eigenständig laufen. [43]

Das CDP Backend ist eine Datenbank, die alle Informationen speichert und der Kern von COMET bildet. Alle Komponenten auf der Clientseite sowie externe Werkzeuge kommunizieren mit dem CDP Backend über die CDP Web Services. Die CDP Web Services übernehmen dabei auch die Authentifizierung und Autorisierung. [43] Zur Authentifizierung mit den CDP Web Services werden ein Benutzername und Passwort benötigt. Die Sitzung wird in einem Cookie gespeichert [44]. Zur Autorisierung wird ein rollenbasiertes Berechtigungsschema verwendet, das bestimmt, welche Rolle auf welche Elemente Zugriff erhält [45]. Neben den CDP

Web Services können auch andere Datenquellen, die ECSS-E-TM-10-25 kompatibel sind, sowie JSON-basierte Datenquellen verwendet werden [43].

4.1.4 Nanospace

Nanospace ist ein Open-Source-Framework, das für das Design in Concurrent-Engineering Sitzungen für Machbarkeitsstudien entwickelt wurde. Es besitzt ein zentrales Datenmodell mit 14 festen Domänen. Es besitzt eine webbasierte Benutzeroberfläche und eine Datenbank, die den Zugriff von mehreren Stellen aus gleichzeitig erlaubt. Nanospace kann über eine REST-API mit weiteren Werkzeugen verbunden werden. Ziel von Nanospace ist es, das Concurrent Engineering zu unterstützen, wie es zum Beispiel in der Concurrent Engineering Facility der ESA der Fall ist. Es unterstützt die Teamarbeit von verschiedenen Orten aus gleichzeitig. Nanospace kann durch einen modularen und flexiblen Aufbau an die Bedürfnisse der Nutzer angepasst werden, indem es die Integration mit Drittanbietersoftware ermöglicht. Durch die webbasierte Benutzeroberfläche kann Nanospace von jedem Gerät mit einem aktuellen Webbrowser verwendet werden. [46] Die Architektur von Nanospace ist Abbildung 4.3 dargestellt.



Simplified Nanospace Architecture. The green circles represent the three ways available to connect third-party applications. (1) Manually interacting with Nanospace-UI. (2) Through a provided generic interface (Python API, or Angular Component). (3) Through the Nanospace REST interface.

Abbildung 4.3: Vereinfachte Nanospace Architektur [46]

Nanospace besteht aus drei Hauptteilen Nanospace-UI, Nanospace-DB und Nanospace-Serivce. Nanospace-UI ist eine webbasierte Benutzeroberfläche aus Basis von Angular2, die dem manuellen Zugriff auf das Modell dient. Die Benutzeroberfläche ermöglicht die Authentifizierung sowie das Verwalten der Modellierungsprojekte. Zu den Modellierungsprojekten können andere Nutzer eingeladen werden, um gemeinsam an einem System zu arbeiten. Die

Strukturierung des Modells erfolgt über eine Baumstruktur, in dem jedem Element Parameter hinzugefügt werden können. Die Parameterliste ist an einem Spreadsheet angelehnt und ermöglicht neben der Eingabe von Daten auch das Setzen von Referenzen und Definieren von Berechnungen. Des Weiteren kann das Systemmodell importiert und exportiert werden. Wenn ein Nutzer Daten geändert hat, werden die Änderungen mit allen Nutzern synchronisiert. Die anderen Nutzer werden über die Änderungen benachrichtigt und die Änderungen werden zur Änderungshistorie hinzugefügt. Die Benutzeroberfläche wird durch einen Webserver ausgeliefert, der auf derselben Maschine wie Nanospace-Service und Nanospace-DB laufen. [46]

Nanospace-DB ist als Graphdatenbank implementiert und folgt einer hierarchischen Struktur mit variabler Tiefe. Das Datenmodell besteht aus einem Projekt, das alle Elemente des Modells beinhaltet. Neben den Nutzern werden die Komponenten des Systemmodells im Projekt gespeichert. Jede Komponente besteht aus mehreren Modes, die jeweils einen Wertetyp besitzen. [46] Ein Mode entspricht dabei einer Zeile in dem Spreadsheet-Editor.

Der Nanospace-Service stellt die Datenbank des Systemmodells (Nanospace-DB) über eine REST-API für die Benutzeroberfläche und externe Software zur Verfügung. Die externe Software kann nach einer Authentifizierung auf das Systemmodell zugreifen. Zur Kommunikation zwischen den serverseitigen Anwendungen werden ein asynchroner Event-Broker sowie eine Datenbank, die die Events speichert, verwendet. Nanospace besitzt drei Möglichkeiten, um domänenspezifische Software zu inkludieren. Die erste Möglichkeit ist das manuelle Interagieren mit der Nanospace-UI. Die zweite Möglichkeit ist das Einbetten von Code über die Angular- oder Python-SDK. Die dritte Möglichkeit ist die Verwendung der REST-API.
Des Weiteren implementiert Nanospace keinen konkreten MBSE-Ansatz und legt den Fokus auf den Austausch von Daten. Obwohl Nanospace für zeitgleichen Zugriff von mehreren Stellen aus geeignet ist, können dennoch Inkonsistenzen entstehen, wenn Drittanbieteranwendungen mit alten Daten rechnen. [46]

4.1.5 Capella

Capella ist ein Open-Source Modellierungs-Werkzeug auf Basis von Eclipse [47] und EMF [48]. Ursprünglich wurde es von Thales entwickelt [47] und wurde bereits von großen Raumfahrtorganisationen erfolgreich eingesetzt [49], [50]. Dabei kann Capella nicht nur für die

Modellierung von Raumfahrtsystemen verwendet werden [51]. Es basiert auf der ARCADIA[3]-Methode und beruht auf UML und SysML, verwendet aber eine eigene vereinfachte Semantik. [47] Die eigene Semantik steht über die ARCADIA Domain Specific Modeling Language (DSML) zur Verfügung und besitzt ein simpleres Vokabular als SysML oder UML [51].

Aufgrund mangelnder Erfahrungen von Ingenieuren mit objektorientierten Konzepten von UML und SysML fokussiert sich ARCADIA primär auf die funktionale Analyse und der anschließenden Zuordnung der Funktionen zu den Komponenten. Die ARCADIA DSML verfügt wie SysML über zehn verschiedene Diagrammtypen wie zum Beispiel Fluss- oder Zustandsdiagramme sowie Darstellungen der Komponentenaufteilung. [51] Das Metamodell von Capella kann durch Viewpoints erweitert werden. Dabei können Viewpoints aus neuen Diagrammen oder neuer Analyselogik bestehen. Dadurch können Viewpoints verschiedene Sichten auf das Systemmodell darstellen. [52] Durch die Integration von ARCADIA in Capella können zum Beispiel Analysen, Konsistenz und Konformität zu ARCADIA überprüft werden. Generelles Model Checking sowie Validierung sind ebenfalls möglich. Wiederverwendbare Komponenten können über Replicable Elements Collections zwischen Projekten geteilt werden. [51]

Capella kann außerdem mit anderer Software integriert werden, indem Capella-Modelle als Basis für vorgelagerte Entwicklungsergebnisse initialisiert werden oder nachgelagerte Werkzeuge durch Capella angestoßen werden. [51] Des Weiteren können die Funktionen von Capella durch Add-Ons erweitert werden [53]. Das Systemmodell kann in Capella über einen HTML-Export mit anderen Stakeholdern geteilt werden, wodurch es als Referenz für andere Nutzer verwendet werden kann [54].

Ein Beispiel für ein Add-On für Capella ist das Add-On für Echtzeit-Kollaborationen „Team for Capella“. Änderungen werden über einen Server, der entweder selbst oder in der Cloud gehostet werden kann, automatisch synchronisiert. Das Add-On ist aber weder kostenlos noch Open-Source. [55]

4.1.6 Virtual Satellite 4

Virtual Satellite ist ein internes DLR-Projekt, dessen anfängliches Ziel es war, den Aufbau eines Raumfahrzeugs, Integration der Komponenten und das Verifizieren der Funktionalität mit

[3] Architecture Analysis & Design Integrated Approach

simulierten Modellen zu unterstützen. Der Fokus verlagerte sich auf die frühen Entwicklungsphasen und der Unterstützung der Concurrent Enginnering Facility (CEF). [56] Virtual Satellite wird meistens für den Entwurf von Satelliten verwendet, ist aber auch für das Design von Launchern geeignet [57]. Virtual Satellite ist ein modulares Framework, das in der Version 3 die Phasen 0 und A für das konzeptionelle Design und Mission Requirements Studies verwendet wurde [58].

In der Version 4 werden auch die weiteren Lebensphasen abgedeckt, wofür das Datenmodell angepasst wurde. Raumfahrtmissionen können bis zu 20 Jahre andauern, wodurch es unmöglich ist, alle Anwendungsfälle vorherzusagen. Deswegen kann das Datenmodell von Virtual Satellite dann angepasst werden, wenn der Bedarf dafür besteht. In frühen Lebensphasen wird eine Produktstruktur erstellt, die in späteren Lebensphasen mit State Machines oder anderen Daten erweitert werden kann. Diese Erweiterungen werden in Virtual Satellite Konzepte genannt. Andere Datenbanken können auch erweitert werden, jedoch ermöglicht Virtual Satellite bei der Erweiterung auch die Bereitstellung von Softwarepaketen. Dadurch kann Virtual Satellite zugeschnittene Softwarepakete und Datenmodelle für bestimmte Ingenieure oder Projektphasen im Lebenszyklus bieten. [11] Eine High-Level Architektur der Anpassungs- und Erweiterungsmechanismen des Datenmodells ist in Abbildung 4.4 dargestellt.

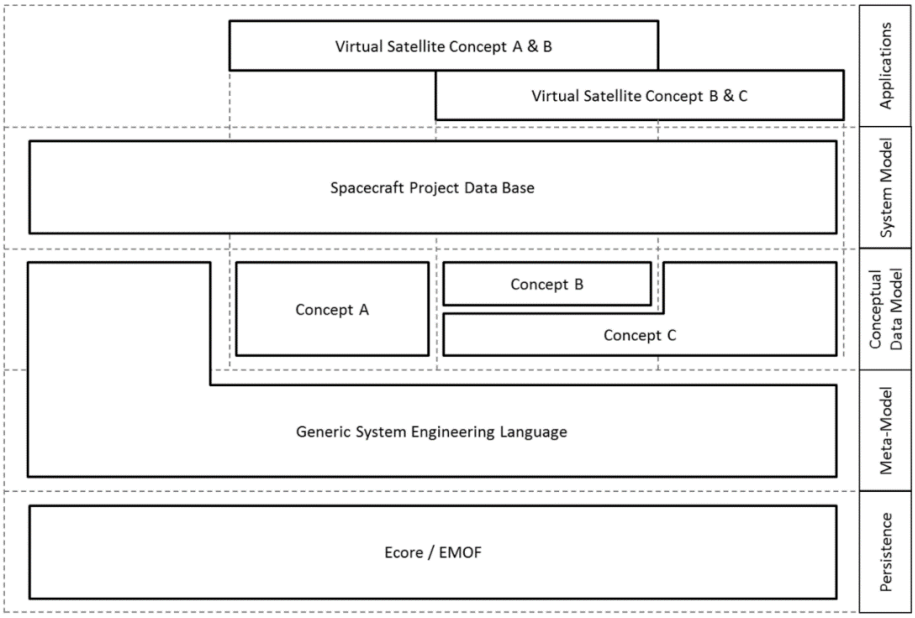


Abbildung 4.4: Virtual Satellite [11]

Als Grundlage wird Ecore (Eclipse spezifische Implementierung des OMG's EMOF) verwendet. Die Persistierung von Informationen, die im Datenmodell gespeichert werden, erfolgt über XMI (XML Medadata Interchange). Darauf aufbauend ist die Generic System Engineering

Language definiert (GSEL). Diese Sprache bietet Metamodellierungsfunktionen, die für das System Engineerung und der Erweiterung des konzeptionellen Datenmodells (CDM) benötigt werden. Die GSEL ermöglicht das Definieren von Strukturelementen und Datencontainern. Die Strukturelemente und Datencontainer werden von einem Konzept instanziiert. Die Konzepte bilden gemeinsam mit der GSEL das komplette CDM. Das CDM kann dann in verschiedenen Versionen von Virtual Satellite verwendet werden. [11] Dadurch ist es möglich, das Datenmodell an die Bedürfnisse für jede Mission anzupassen. Die Konzepte können bei Bedarf von den Ingenieuren zur Laufzeit aktiviert werden. Die Datencontainer beinhalten Parameter bzw. Eigenschaften, Referenzen und Berechnungen. Die Konzepte können auf Basis der Eclipse und EMF-Infrastruktur als Plugin installiert werden. [37] Durch die Anpassbarkeit des Datenmodells kann Virtual Satellite in Projekten eingesetzt werden, ohne alle spezifischen Anforderungen zu kennen. Eine iterative Einführung von MBSE ist dadurch möglich [11]. Virtual Satellite 4 kann auch im Headless-Betrieb ohne grafische Oberfläche betrieben werden [59]. Die Architektur des Headless REST-Server ist in Abbildung 4.5 dargestellt.

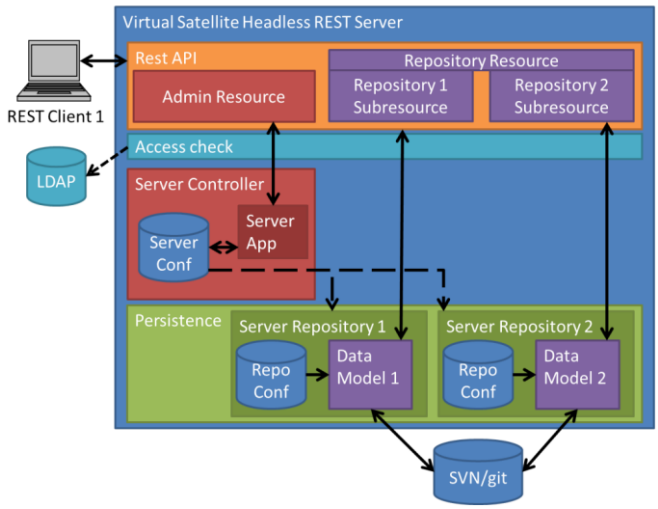


Abbildung 4.5: Komponenten Virtual Satellite Server [59]

Der Server teilt sich in die REST-API, Server-Controller und Persistence auf. Zwischen der REST-API und dem Server-Controller befindet sich der Access check Layer, der den Zugriff anhand der Login-Daten auf die Ressourcen prüft. Dabei handelt es sich um eine serverweite Authentifizierung und Autorisierung. Es werden dieselben Login-Daten verwendet, die auch für die normale Nutzerverwaltung von Virtual Satellite verwendet werden. REST-Clients können auf Admin- und Repository-Ressourcen zugreifen. Zu den Admin-Ressourcen gehören Servereinstellungen, Nutzerverwaltung oder Einstellungen für Repositories. Jedes Repository hat eine eigene Subressource. Der Server-Controller verarbeitet Anfragen von den

Adminressourcen. Anfragen an Repository-Subressourcen verändern direkt das unterliegende Datenmodell. Die Persistenzschicht besteht aus Server Repositories, die jeweils eine Repository Configuration und ein Datenmodell besitzen. Eine Server-Instanz kann mehrere Server-Repositories verwalten.

Das Datenmodell ist jeweils mit einem SVN- oder Git-Backend verbunden. Die verbundenen SVN- und Git-Repositories können von Clients mit der normalen Virtual Satellite UI verwendet werden. Der Server löst Konflikte auf und verwaltet Änderungen von mehreren REST-Clients. Das SVN oder Git Repository fungiert dabei als single source of truth. [59]

4.2 Web- und Cloud-Entwicklungsumgebungen

Entwicklungsumgebungen werden zunehmend über die Cloud bereitgestellt. Web- und cloudbasierte IDEs vereinfachen das Bereitstellen und die Veröffentlichung von neuen Funktionen und können Ressourcen nutzen, die die Leistung eines lokalen Computers übersteigen. Des Weiteren ist keine Installation notwendig. [60] Im Folgenden werden aktuelle Web- und Cloud-Entwicklungsumgebungen vorgestellt.

4.2.1 Visual Studio Code

Visual Studio Code (VS Code) ist ein leichtgewichtiger Source-Code-Editor, der standardmäßig JavaScript, TypeScript und Node.js unterstützt. VS Code kann durch Erweiterungen für andere Sprachen und Laufzeiten erweitert werden und ist unter Linux, macOS und Windows verfügbar. [61] VS Code selbst basiert auf TypeScript, Node.js und Electron [62]. Für die Entwicklung von Erweiterungen bietet VS Code eine API an. Erweiterungen können auch dafür genutzt werden, die Benutzeroberfläche zu erweitern. Viele Funktionen von VS Code sind selbst als Erweiterung implementiert. [63] VS Code baut auf dem Open-Source-Projekt vscode auf. VS Code hat dabei eine Microsoft-Produkt-Lizenz, während vscode eine MIT-Lizenz besitzt. Der Unterschied zwischen den beiden ist, dass VS Code auf vscode aufbaut und einige Funktionen wie den Marktplatz für Erweiterungen hinzufügt. [64] VSCodium ist ein Projekt, das das Open-Source-Projekt vscode als fertiges Endprodukt ohne die Anpassungen von Microsoft anbietet [65]. Im Folgenden wird nur noch das Open-Source-Projekt vscode betrachtet, da VS Code nicht Open-Source ist. Bevor eine Serverimplementierung für vscode veröffentlicht wurde, wurden code-server [66] und OpenVSCode Server [67] entwickelt. Mittlerweile ist auch im vscode Projekt eine Serverimplementierung mit Browserversion im Sourcecode enthalten.

OpenVSCode Server und code-server bauen beide auf einem Fork von vscode auf, haben aber jeweils einen anderen Fokus. Während OpenVSCode Server sich darauf fokussiert, vscode im Browser lauffähig zu machen, bietet code-server noch weitere Funktionen wie zum Beispiel Authentifizierung oder eine Plugin-API. [68] An dieser Stelle sei auch die GitLab Web IDE zu erwähnen, die nur zum Schreiben von Code, aber nicht zum Ausführen verwendet werden kann [69].

4.2.2 Eclipse Theia

Eclipse Theia ist eine erweiterbare Plattform zur Entwicklung von Cloud- und Desktop-IDEs ähnlicher Produkte auf Basis von aktuellen Webtechnologien [70]. Es ist damit kein fertiges Produkt wie VS Code oder VSCodium. Eclipse Theia wird über die Eclipse Public License bereitgestellt [71].

Eclipse Theia kann sowohl als Cloud- als auch als Desktop-IDE deployt werden. Eclipse Theia kann durch das Language Server Protocol um die Unterstützung von weiteren Sprachen erweitert werden. [70] Theia deckt typische Anwendungsgebiete der Eclipse Rich Client Plattform (Eclipse RCP) ab und zeichnet sich durch Erweiterbarkeit auf Client- als auch Serverseite aus. Bestehende Funktionalitäten können geändert oder entfernt sowie neue Funktionalitäten hinzugefügt werden, um Theia an projektspezifische Anforderungen anzupassen. Dabei bietet Theia Kompatibilität mit Erweiterungen von VS Code. Dabei wird Theia unter der Eclipse Foundation herstellerneutral von verschiedenen Firmen entwickelt und setzt auf Standards. [72] Eclipse Theia setzt dabei auf dieselben Basistechnologien wie VS Code.

Eclipse Theia kann durch VSCode Extensions, Theia Extensions und Theia Plugins erweitert werden. VS Code Extensions sind zur Laufzeit installierbar und haben die gleichen Limitierungen wie in VS Code[4]. Theia Extensions werden zur Kompilierungszeit installiert und besitzen vollen Zugriff auf die Interna von Theia. Die Kernfunktionen von Theia sind ebenfalls über Theia Extensions implementiert. Theia Plugins sind mit VS Code Extensions vergleichbar und können ebenfalls zur Laufzeit installiert werden. Theia Plugins haben aber eine umfangreichere Theia API zur Verfügung. [73]

[4] Die VS Code API wird nicht zu 100% unterstützt, wodurch nicht alle Extensions funktionieren. Ebenso fallen wie bei VSCodium die proprietären Erweiterungen von Microsoft weg.

Damit Theia sowohl als Desktop- als auch als Browser-Anwendung mit externem Server verwendet werden kann, läuft Theia in zwei Prozessen (Frontend und Backend). Die Kommunikation erfolgt über JSON-RPC, WebSockets oder REST-APIs. In der Desktopversion laufen beide Prozesse in Electron. Im Remote-Kontext läuft das Frontend im Webbrowser und das Backend auf einem Server. Das Frontend läuft auf JavaScript und setzt einen Webbrowser voraus. Das Backend läuft ebenfalls auf JavaScript mit Node.js. [74]

Ein Beispiel für eine IDE auf Basis von Eclipse Theia ist RIDE. RIDE ist eine auf Theia basierte Web-IDE für die textbasierte Reflex-Sprache, die für cyber-physische Systeme verwendet wird. Durch den modularen Aufbau können sowohl Frontend als auch Backend um die benötigten Funktionen erweitert werden. RIDE läuft lokal auf dem Rechner und benötigt je nach Zielplattform spezielle Module, die extern von der IDE in Docker-Containern laufen. Das steigert die Komplexität für den Nutzer bei der Nutzung sowie bei der Wartung der Software. [75]

4.2.3 EMF.Cloud

EMF.cloud ist ein Dachprojekt für Komponenten und Technologien, die das Eclipse Modelling Framework (EMF) für Web- und Cloudanwendungen verfügbar machen. Dazu gehören sowohl neue Webframeworks als auch Lösungen bestehende EMF-Technologien weiterzuverwenden. Zu den Komponenten zählen ein GLSP-basierter Ecore Editor, ein Model-Server sowie EMF-JSON Jackson. Der Model-Server bietet eine REST-API für bereits existierende EMF-Implementierungen zur Verfügung und EMF-JSON Jackson ermöglicht das Serialisieren und Deserialisieren von EMF in JSON. Außerdem stellt EMF.Cloud einen Baum-Editor sowie den Coffee-Editor zur Verfügung. [76] EMF.Cloud steht wie Eclipse Theia unter der Eclipse Public License [77].

Der Coffee-Editor ist ein Beispiel für eine webbasierte Entwicklungsumgebung zum Modellieren auf Basis von EMF.Cloud. Es unterstützt formular-, tabellen- und diagrammbasiertes Editieren. Des Weiteren unterstützt er das Editieren von eigenen textuellen domänenspezifischen Sprachen (DSLs) mit Syntaxhervorhebung. Auf Basis des erstellen Modells können außerdem Code generiert sowie grafische Analysen durchgeführt werden. [76]

Zu den Projekten zählt die Graphical Language Server Platform (GLSP), mit der, ähnlich wie mit dem Language Server Protocol, Unterstützung für grafische Sprachen bereitgestellt werden kann. Ein weiteres Projekt ist JSON Forms, das, wie EMF-Forms, eine Benutzeroberfläche aus einem deklarativen Modell erzeugt. [76]

4.2.4 Gitpod

Gitpod ist eine Open-Source Kubernetes Anwendung, die es ermöglicht, Entwicklungsumgebungen in der Cloud zu verwenden. Die Entwicklungsumgebungen werden über Code definiert und sind über einen Webbrowser abrufbar. Durch das Definieren über Code [78] sind die Entwicklungsumgebungen reproduzierbar und können unter Versionskontrolle gestellt sowie mit anderen Personen geteilt werden. Werkzeuge, die für die Entwicklungsumgebung benötigt werden, laufen in einem Container im Kubernetes-Cluster. Auf der Frontend-Seite können Erweiterungen genutzt werden [79], wodurch die IDE um Funktionen erweitert werden kann [80]. Zur Erweiterung der IDE auf Serverseite können Werkzeuge manuell im Container installiert oder in der Dockerfile oder Workspace-Konfigurationsdatei hinzugefügt werden [81]. Der Workspace in Gitpod kann entweder über den Browser mit der Browserversion von VS Code, über SSH oder über Desktopanwendungen VS Code oder JetBrains Gateway verwendet werden [82]. Über SSH können auch externe Anwendungen mit dem Workspace verbunden werden. Gitpod kann entweder selbst gehostet oder als Software as a Service (SaaS) genutzt werden [83]. Die Architektur von Gitpod ist in Abbildung 4.6 dargestellt.

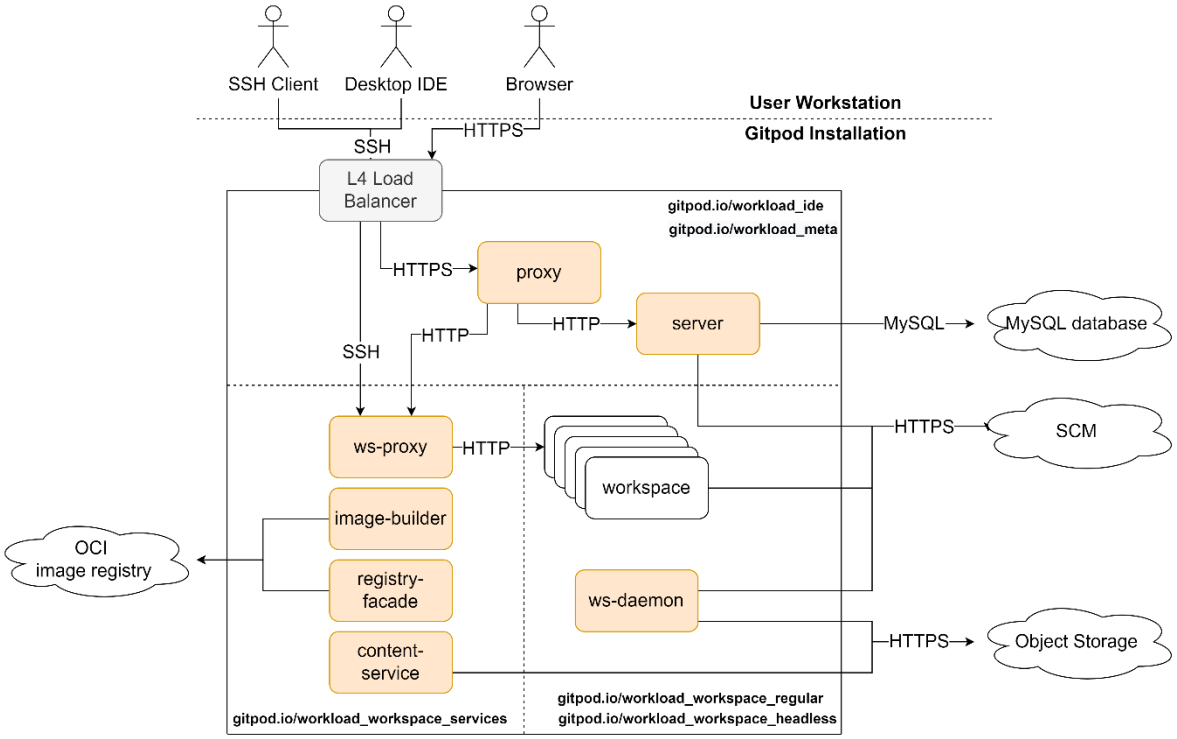


Abbildung 4.6: Gitpod Architektur, nach [84]

Über einen Layer 4 Load Balancer erlangt der Nutzer Zugriff Gitpod. Ein Internet-Proxy verteilt die Last innerhalb von Gitpod. Gitpod überträgt einige Aufgaben an cluster-externe

Komponenten. Dazu zählt eine MySQL Datenbank, die Metadaten über Nutzer und Workspaces speichert. Dazu zählen auch Einstellungen der Gitpod-Instanz wie den Authentifizierungsanbieter. Des Weiteren wird das Source Control Management (SCM) ausgelagert. Gitpod ist ohne ein SCM nicht verwendbar [85]. Ein Object Storage wird zum Speichern von BLOB-Dateien verwendet. Dazu zählen Zustände inklusive IDE-Präferenzen der Workspaces sowie deren Backups. Außerdem wird eine Open Container Initiative (OCI) Image Registry benötigt, die für das Cachen von externen Images sowie benutzerdefinierten Images verwendet wird. Hier liegen nicht die Images für Gitpod selbst. Gitpod nutzt Image Registries für drei Zwecke. Zum Beziehen von Gitpod-Software und Base-Images für Workspaces sowie zum Beziehen und Speichern von benutzerdefinierten Workspace-Images. Das ist die einzige Image-Registry, zu der die Gitpod-Instanz Schreibzugriff auf eine Registry benötigt. In einem Szenario ohne externen Internetzugriff können auch alle Images von derselben (internen) Registry bezogen werden. [84] Dieser Betrieb wird als air-gapped bezeichnet und wird nicht in der kostenlosen Open-Source-Version unterstützt [86].

Das Kubernetes-Cluster von Gitpot besteht aus drei Node-Pools, weil die Workloads stark unterschiedliche Ressourcenbedürfnisse haben. Der Services Node Pool (obere Hälfte des Diagramms) beinhaltet die Gitpod-Anwendung mit ihren Services. Die Gitpod-Anwendung stellt das Dashboard zur Verfügung und übernimmt die Provisionierung der Workspaces. Im Regular Workspaces Node Pool (Linke Hälfte des Diagramms) werden die eigentlichen Workspaces deployt, die von dem Nutzer verwendet werden. Im Headless Workspace Node Pool deployt Gitpod (Rechte Hälfte des Diagramms) den Imagebuilder sowie Workspaces, die vorgebaut werden. [84]

4.2.5 Eclipse Che

Eclipse Che ist eine zentralisierte Entwicklungsumgebung, die auf Kubernetes oder OpenShift lauffähig ist. Eclipse Che wird von den Entwicklern als eine Kubernetes-native Entwicklungsumgebung und Entwickler Kollaborationsplattform bezeichnet. Die Ziele von Che sind einerseits, das Onboarding von neuen Nutzern durch „zero install“ zu beschleunigen, indem Entwicklungsumgebungen im Browser laufen. Andererseits verfolgt Eclipse Che das Ziel, Inkonsistenzen zwischen Entwicklungsumgebungen von Projektmitgliedern zu vermeiden. Des Weiteren will Eclipse Che integrierte Sicherheit und Unternehmenstauglichkeit bieten, indem es eine Virtual Desktop Infrastructure (VDI) bietet, die Sourcecode von lokalen Maschinen fernhält und Sicherheit über ein rollenbasiertes Zugriffsmanagement ermöglicht. [87] Eclipse Che kann selbst gehostet werden oder als Software-as-a-Service genutzt werden [88]. Bei einer

eigenen Installation kann Eclipse Che außerdem ohne Internetverbindung betrieben werden, das aktuell aber nur auf OpenShift unterstützt wird [89].

Zum Erfüllen dieser Ziele bietet Che zum einen container-basierte Workspaces, die alle Werkzeuge und Abhängigkeiten enthalten, die zum Programmieren, Bauen, Ausführen und Debuggen von Anwendungen benötigt wird. Workspaces sind voneinander isoliert und sind selbst für den Lebenszyklus ihrer Komponenten verantwortlich. Außerdem bietet Eclipse Che verschiedene Browser-basierte IDEs an sowie die Möglichkeit, eigene IDEs zu verwenden. [87]

Eclipse Che ist eine erweiterbare Plattform, wo neben einer eigenen IDEs auch eigene Werkzeuge durch Plugins auf Client- als auch Serverseite hinzugefügt werden können. Ein Stack ist eine Vorlage für einen Workspace, der über eine Konfigurationsdatei alle Werkzeuge des Workspace definiert. [87] Dabei können in der Konfigurationsdatei mehrere Container definiert werden, die die Werkzeuge betreiben. Durch das Definieren als Konfigurationsdatei lassen sich die Workspaces replizieren. Eclipse Che bietet eine SDK an, um eine eigene Plattform zu entwickeln. Eine Integration in ein Unternehmen ist durch die Unterstützung von mehreren Benutzern sowie Authentifizierung mit Keycloak, LDAP oder AD möglich. [87] Die Architektur von Eclipse Che ist in Abbildung 4.7 dargestellt.

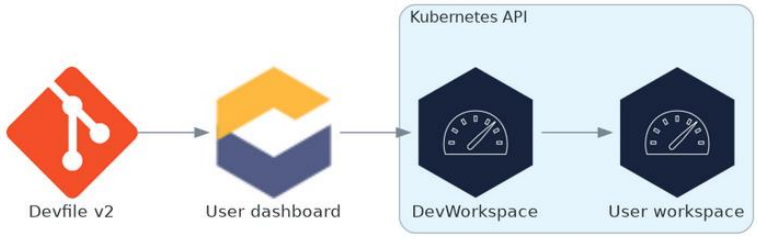


Abbildung 4.7: Eclipse Che Architektur [90]

Che betreibt drei Arten von Komponenten. Che-Server Komponenten verwalten den User-Namespace und User-Workspaces. Die Hauptkomponente ist das User-Dashboard, von dem Nutzer ihre Workspaces verwalten können. Die zweite Gruppe ist der Dev Workspace Operator, der die benötigten Kubernetes Objekte erstellt und verwaltet, die für den Betrieb der Workspaces benötigt werden. Dazu gehören Pods, Services und PersistentVolumes. Die User-Workspaces. Beinhalten die container-basierten Entwicklungsumgebungen inklusive der IDE. Die Kommunikation der drei Komponentengruppen geschieht über die Dev Workspace Custom Ressources. Diese Custom Ressources sind che-spezifische Kubernetesobjekte, die die User-Workspaces repräsentieren und als Kommunikationskanal dienen. Der Zugriff auf alle Ressourcen wird über Kubernetes role-based access control (RBAC) realisiert. [90] Als

Basis für die User-Workspaces dient eine Devfile in Version 2, die den Aufbau des Workspaces beschreibt [91]. Der Aufbau eines User-Workspaces ist in Abbildung 4.8 darstellt.

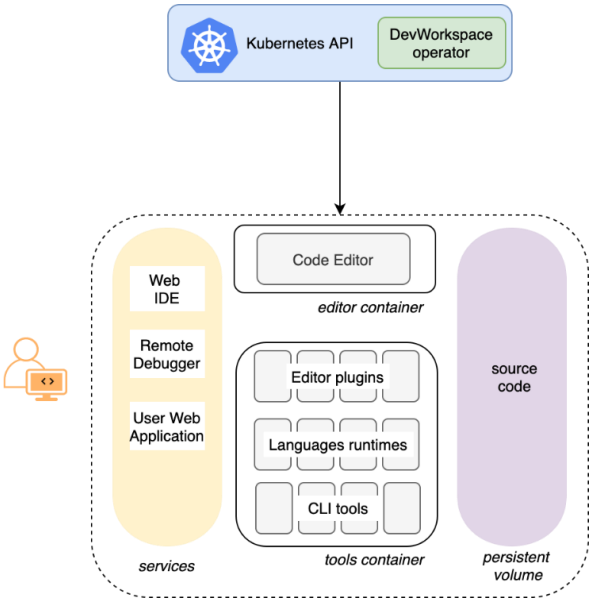


Abbildung 4.8: Eclipse Che User-Workspace [92]

Ein User-Workspace ist eine Web-Anwendung, die aus Microservices besteht. Die Microservices laufen innerhalb der Container und bieten die Services an, die von dem Nutzer verwendet werden können. Dazu zählt einmal die Web-IDE, mit der der Nutzer den Workspace nutzen kann. Außerdem zählen dazu auch Funktionen wie der Remote-Debugger, mit der der geschriebene Code debuggt werden kann. [92] Außerdem ermöglicht Che die Ausführung des geschriebenen Codes. Webanwendungen (User Web Application) können dabei auch in der Benutzeroberfläche in Che verwendet werden.

Dazu zählen neben Autovervollständigung auch Erweiterungen, Laufzeitumgebungen oder andere Werkzeuge. Der Workspace ist ein Kubernetes Deployment, das alle Container des Workspaces, Erweiterungen und dazugehörige Kubernetes Objekte enthält. Dazu zählen neben den Containern auch definierte Endpunkte, Secrets und Persistent Volume (PVs). Der Source-Code der Projekte wird in einem Kubernetes oder OpenShift PV persistiert. Die Microservices des Workspaces haben Lese- sowie Schreibberechtigungen auf das geteilte Verzeichnis im Workspace. [92]

Ein Modellierungsframework, das sowohl Eclipse Theia als auch Eclipse Che einsetzt, ist tCollab. Verwendet wurde es für die Implementierung einer IDE für die User Requirements Notation (URN) und Generierung von Diagrammen. Es unterstützt Echtzeit-Kollaboration, verfolgt

aber eine Last-Write-Win-Policy, die zu Inkonsistenzen führen kann, wenn ein Nutzer ein Element löscht und ein anderer Nutzer das Hinzufügen eines Elementes rückgängig macht. [93]

In einer darauffolgenden Arbeit unterstützt es auch konfliktfreie Echtzeitkollaboration. Allerdings in der proprietären VS Code Version als IDE. Eine Version, die auch in Eclipse Theia funktioniert, befindet sich noch in Entwicklung. [94]

4.3 Was fehlt für eine cloud-native Entwicklungsumgebung?

Keines der vorgestellten MBSE-Tools ist cloud-native oder nutzt die Vorteile einer Cloudumgebung. Die meisten Tools bieten eine Web-API über einen Server an, aber nur Nanospace und COMET bieten eine webbasierte Benutzeroberfläche an. Das Datenmodell von Nanospace ist erweiterbar, folgt aber keiner Struktur wie bei VSD, Virtual Satellite oder Capella.

Des Weiteren ist Nanosat zwar webbasiert und wird als Docker-Container verteilt, bietet aber keine Lösung an, wie es in einer Cloudumgebung verwendet werden kann. Capella ist von den vorgestellten Tools am ausgereiftesten, bietet aber weder eine webbasierte Benutzeroberfläche noch eine Server-API an oder ermöglicht den Betrieb in einer Cloudumgebung.

Berechnungen und Simulationen sind bei allen Lösungen auf die Ressourcen des lokalen Computers limitiert, da sie nicht die Vorteile einer Cloudumgebung nutzen können. Des Weiteren müssen alle Tools vom Nutzer installiert und gewartet werden.

In den letzten Jahren wurden Open-Source-Projekte wie Eclipse Theia, EMF.Cloud und Eclipse Che entwickelt, die wichtige Wegbereiter für das Entwickeln von Entwicklungsumgebungen zum Modellieren in der Cloud darstellen. Es wäre auch vorher bereits möglich gewesen, eine Entwicklungsumgebung zum Modellieren in der Cloud zu entwickeln, es wäre aber deutlich zeitaufwendiger gewesen. Workspace-Server-Lösungen wie Eclipse Che und Gitpod sowie webbasierte Entwicklungsumgebungen wie Eclipse Theia sind allerdings stark auf das klassische textbasierte Programmieren ausgelegt. Das zeigt sich auch in der Integration mit externen Anwendungen, wo zurzeit nur Versionskontrollsysteme wie GitLab oder GitHub unterstützt werden.

EMF.Cloud bietet mit dem Coffee-Editor auf Basis von Eclipse Theia ein Beispielprojekt für das Modellieren eines Systems an. Allerdings bietet es kein zur Laufzeit anpassbares Datenmodell an. Des Weiteren kann es für Berechnungen oder Simulationen nicht die Vorteile einer Cloudumgebung nutzen oder externe Software anbinden.

Mit tCollab existiert ein Modellierungsframework für die URN, das mit Eclipse Theia und Che aktuelle Web- und Cloud-Technologien nutzt. Es ermöglicht jedoch nicht das Modellieren eines Systems inklusive Struktur mit einem zur Laufzeit erweiterbarem Datenmodell. Außerdem zieht es für z. B. Berechnungen und Simulationen keine Vorteile aus einer Cloudumgebung.

Der Gap lässt sich mit den folgenden Stichpunkten zusammenfassen:

Mangelnde Nutzung der Vorteile einer Cloudumgebung

- Keines der vorgestellten MBSE-Tools nutzt eine Cloudumgebung für Berechnungen oder externe Software
- Keines der vorgestellten web- und cloudbasierten Entwicklungsumgebungen nutzen die Cloud für Berechnungen oder externe Software

Webbasierte MBSE-Tools nur ohne erweiterbares Datenmodell

- Keines der vorgestellten MBSE-Tools bietet sowohl eine Web-API mit Webfrontend als auch ein erweiterbares Datenmodell

Entwicklungsumgebungen nur ohne erweiterbares Datenmodell

- Keines der vorgestellten web- und cloudbasierten Entwicklungsumgebungen unterstützt das Modellieren eines Systems mit einem erweiterbarem Datenmodell

5 Konzept

Im folgenden Kapitel wird das Konzept der Arbeit erläutert. Zunächst wird die Methodik der Anforderungsermittlung vorgestellt und im Anschluss werden die zusammengefassten und kategorisierten Anforderungen dargestellt. Anhand der Anforderungen wird ein Konzept für die Lösung erarbeitet. Zunächst wird die Architektur des Clusters vorgestellt. Danach wird näher auf das Konzept für die Workspaces und Entwicklungsumgebung eingegangen, die vom Ingenieur zum Modellieren verwendet werden. Abschließend wird das Konzept für das Concurrent Engineering erörtert, bei dem es um das gemeinsame Modellieren mit anderen Ingenieuren geht.

5.1 Methodik

Anforderungen sind die Beschreibung eines IT-Systems, bevor es existiert. Zum ingenieurmäßigen Arbeiten gehört, dass ein zu entwickelndes IT-System spezifiziert wird, bevor es entwickelt wird. In einem IT-Projekt arbeiten Menschen mit verschieden ausgeprägten IT-Kenntnissen zusammen, was das gemeinsame Verständnis behindern kann. Die Anforderungen dienen dem gemeinsamen Verständnis zwischen Kunden und Entwickler. [95]

Anforderungen können in funktionale und Qualitätsanforderungen (nicht-funktional) aufgeteilt werden. Eine funktionale Anforderung beschreibt eine Funktion, die das System besitzen soll. Eine Qualitätsanforderung beschreibt die Art, wie ein System eine Funktion besitzen soll. Funktionale Anforderungen an ein IT-System sind oft kundenspezifisch, da sie deren Geschäftsprozesse widerspiegeln. Qualitätsanforderungen können in einem hohen Maße wiederverwendet werden. Es gibt verschiedene Standards, die beschreiben, welche Typen von Qualitätseigenschaften (Qualitätsattribute) die Qualität einer Software beschreiben. [95]

Die ISO/IEC 2010 beschreibt ein Qualitätsmodell für Software mit den Qualitätsattributen Funktionalität, Zuverlässigkeit, Leistungseffizienz, Benutzerfreundlichkeit, Sicherheit, Komptabilität, Wartbarkeit und Portabilität (Siehe Abbildung A.3 im Anhang) [95].

Um im Rahmen der Arbeit zu bleiben, werden Anforderungen zur Usability nicht erhoben, da ein Prototyp mit eingeschränkter Funktionalität implementiert wird. Zur Dokumentation der Anforderungen werden User Stories verwendet. In diesem Kapitel wird die Methodik der Anforderungsermittlung erklärt und im Anschluss die gesammelten und zusammengefassten Anforderungen dargestellt.

5.1.1 Ermittlung der Anforderungen

User Stories sind ein pragmatischer Kompromiss zwischen vollständig natürlicher Sprache und einer formalen Spezifikationssprache. Insbesondere in der agilen Softwareentwicklung hat sich die User Story als Standard-Schablone etabliert. Aber auch außerhalb der Agilität ist die User Story eine sinnvolle Formulierung von Anforderungen im Problemraum aus Benutzersicht, die Szenarien mit Zielen verknüpft. [95]

Eine User Story folgt dabei der Struktur: *„Als [Benutzerrolle] möchte ich [Funktionalität] so dass [Nutzen]“*, wobei der Nutzen-Teil optional ist [85], [86]. Diese Form wird auch „Three Rs (Role, Requirement, Reason) oder Connextra-Format genannt [86]. Qualitätskriterien für User Stories folgen den INVEST-Kriterien und sind in den User Stories festgeschrieben [95].

Jede User Story sollte:

- in sich geschlossen sein und möglichst unabhängig von anderen erfüllt werden können. (**Independent**)
- nicht zu restriktiv formuliert sein und verändert oder verworfen werden können. (**Negotiable**)
- für das Produkt wertvoll sein. (**Valuable**)
- genug Informationen enthalten, um den Aufwand einzuschätzen. (**Estimable**)
- gerade groß genug sein um ein atomares Problem zu lösen. Kleine User Stories sind einfacher zu verstehen. (**Small**)
- so geschrieben werden, dass sie testbar ist. Durch das Testen kann überprüft werden, ob die User Story erfüllt ist. (**Testable**)

Akzeptanzkriterien sind Bedingungen, die erfüllt sein müssen, damit die User Story erfüllt und vom Stakeholder akzeptiert wird [97]. Akzeptanzkriterien machen User Stories testbar [98]. Außerdem sind Akzeptanzkriterien eine Möglichkeit, Details zu einer User Story hinzuzufügen [99].

Eine Möglichkeit zur Klassifikation von Anforderungen ist das Kano-Modell. Mit den Kano-Modell können Anforderungen durch eine Zwei-Kriterien-Klassifikation eingeteilt werden. Das bedeutet, dass zwei Kriterien für die Priorisierung verwendet werden. Bei der Kano-Klassifikation werden den Stakeholdern eine funktionale und eine dysfunktionale Frage gestellt, wie es wäre, wenn die Anforderung erfüllt bzw. nicht erfüllt wird. Anhand der Antworten können die Anforderungen in verschiedene Kategorien eingeteilt werden. [95]

- Basismerkmale (**Must-Be**): Werden vom Nutzer als selbstverständlich vorausgesetzt und sind ihm nicht bewusst. Die Nichterfüllung resultiert in Unzufriedenheit.
- Leistungsmerkmale (**One Dimensional**) Die Erfüllung der Anforderung resultiert in Zufriedenheit sowie die Nicht-Erfüllung zu Unzufriedenheit.

- Begeisterungsmerkmale (**Attractive**): Sind Eigenschaften, die nicht erwartet werden, aber bei Erfüllung für Zufriedenheit sorgen, da Nutzer teilweise nicht wissen, dass die Anforderungen machbar sind.
- Unerhebliches Merkmal (**Indifferent**): Das Erfüllen der Anforderung sorgt weder für Zufriedenheit noch Unzufriedenheit
- Rückweisungsmerkmal (**Reverse**): Das Erfüllen der Anforderung sorgt für Unzufriedenheit.

Anforderungen werden als fragwürdig (**Questionable**) deklariert, wenn jeweils die funktionale und dysfunktionale Frage positiv bzw. negativ beantwortet wird [100]. Das deutet darauf hin, dass die Frage falsch verstanden oder gestellt wurde. Auf die Auswertung dieser Fragen sollte verzichtet werden. [101] Zur Veranschaulichung ist der Zusammenhang zwischen Zufriedenheit und Unzufriedenheit und der Erfüllung einer Anforderung ist in Abhängigkeit der Kategorie in Abbildung 5.1 dargestellt.

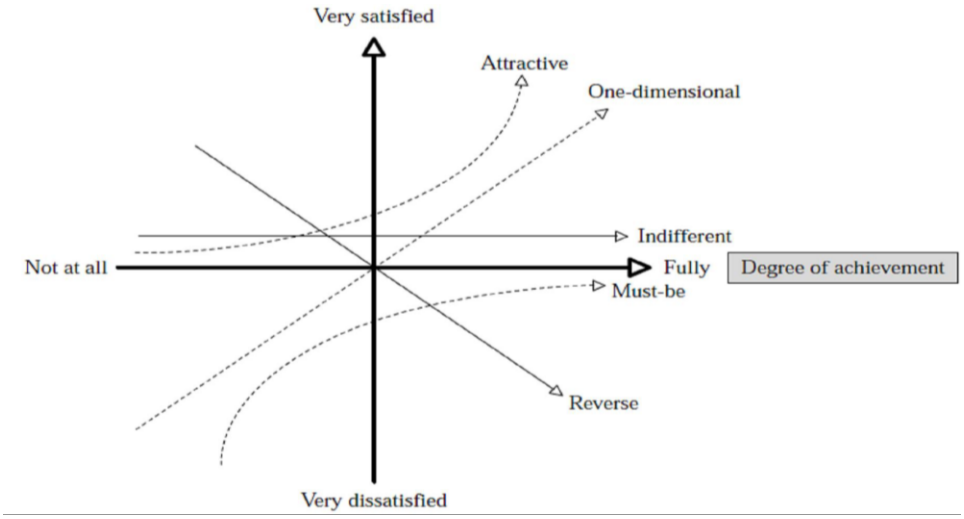


Abbildung 5.1: Kano-Modell [102]

Das Kano-Modell wird im Rahmen der Arbeit verwendet, um herauszufinden, welche Anforderungen von den Ingenieuren als wichtig empfunden werden, um diese bei der Implementierung zu priorisieren. Des Weiteren verwendet das Kano-Modell eine relativ geringe Anzahl von Kriterien (zwei) im Vergleich zu anderen Techniken und die Anzahl der Schätzungen pro bestehende und neue Anforderung wächst linear an (vgl. Abbildung A.1).

Das International Requirements Engineering Board (IREB) empfiehlt, mehrere Techniken und Quellen zur Anforderungsermittlung heranzuziehen. Für Basisanforderungen werden Beobachtungstechniken und dokument- bzw. systembasierte Techniken empfohlen. Für Leistungsanforderungen werden Befragungstechniken und für Begeisterungsanforderungen Kreativtechniken empfohlen. [95]

Für Basisanforderungen werden daher bereits existierende Systeme und deren Dokumente als Anforderungsquelle herangezogen und daraus User Stories formuliert. Für Leistungsanforderungen wird ein Fragebogen verwendet, in dem User Stories erfragt werden. Für Begeisterungsanforderungen wird aus Zeitgründen und schlechter Verfügbarkeit der Stakeholder ebenfalls der Fragebogen verwendet. In diesem werden auch Anforderungen erfragt, die von aktuell verwendeten Tools nicht erfüllt werden, um Begeisterungsmerkmale zu sammeln. Ergänzt wird dies durch existierende Systeme und deren Dokumente. Dabei geschieht die eigene Anforderungsermittlung unabhängig von dem Fragebogen, um eine Beeinflussung zu vermeiden. Am Ende werden die gesammelten Anforderungen zusammengeführt und in die Gruppen der ISO 25010 aufgeteilt. Die Methodik der Anforderungsermittlung ist in Abbildung 5.2 dargestellt.

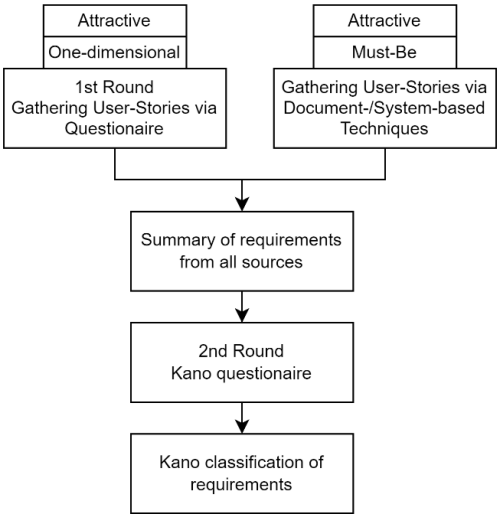


Abbildung 5.2: Methodik Anforderungsermittlung

Die Anforderungsermittlung besteht aus vier Schritten. Im ersten Schritt werden Anforderungen als User Stories über einen Fragebogen sowie eigene Recherchen ermittelt. Im zweiten Schritt werden die Anforderungen aus allen Quellen gefiltert und zusammengefasst sowie in die ISO 25010 Gruppen eingeteilt. Im dritten Schritt wird eine zweite Fragebogenrunde durchgeführt, in der die Anforderungen nach dem Kano-Modell bewertet werden. Im letzten Schritt werden die Anforderungen nach Kano auf Basis des 2. Fragebogen klassifiziert.

Zur Konsolidierung wird bei der Kano-Klassifizierung die von Puliot vorgeschlagene Kategorisierungstabelle verwendet, die widersprüchliche Angaben bei der Bewertung berücksichtigt (vgl. Abbildung A.2).

5.1.2 Zusammengefasste und kategorisierte Anforderungen

Insgesamt wurden sieben Ingenieure aus verschiedenen Disziplinen befragt, die bereits Erfahrungen mit MBSE im Raumfahrtbereich besitzen. Von den sieben Ingenieuren nahmen sechs an der zweiten Runde teil. Insgesamt ergaben sich 50 Anforderungen nach der Filterung und Zusammenfassung des Fragebogens.

In folgenden Tabellen findet sich eine vereinfachte Darstellung der Anforderungen mit Kano Kategorisierung. Die Anforderungen wurden in die Hauptgruppen funktionale und nicht-funktionale Anforderungen aufgeteilt. Zu den funktionalen Anforderungen gehören die Gruppen Modellierung, Concurrent Engineering, Schnittstellen & Export, Produktlinien & Versionierung sowie Authentifizierung & Autorisierung. Zu den nicht funktionalen Anforderungen gehören die Gruppen Leistungseffizienz, Komptabilität, Zuverlässigkeit, Sicherheit, Wartbarkeit und Portabilität. Im Anhang ist eine komplette Darstellung hinterlegt.

Funktionale Anforderungen

Modellierung

Nr.	Anforderung	Kano
fM1	Mit der Software kann ein System modelliert werden.	M
fM2	Mit der Software kann an einem eigenen Subsystem an der eigenen Domäne gearbeitet werden.	M
fM3	Die Software bietet unterschiedliche Sichten auf die Daten an und ermöglicht das definieren von eigenen Sichten.	M
fM4	Mit der Software kann die Modellierungssprache erweitert werden.	A
fM5	Mit der Software können Berechnungen und Referenzen definieren werden.	A
fM6	Mit der Software können Anforderungen definiert und mit Elementen vom Systemmodell verknüpft werden.	M
fM7	Mit der Software können Simulationen durchgeführt werden.	A
fM8	Mit der Software kann das Systemmodell auf Konsistenz geprüft werden.	M
fM9	Mit der Software kann eine Impact Analysis durchgeführt werden.	A
fM10	Mit der Software können Daten in weitere Lebensphasen übernommen werden.	O

Tabelle 5.1: Funktionale Anforderungen: Modellierung

Concurrent Engineering

Nr.	Anforderung	Kano
fC1	Mit der Software können mehrere Ingenieure gleichzeitig Zugriff auf das Systemmodell haben.	M
fC2	Mit der Software können Merge-Konflikte aufgelöst werden.	O
fC3	Mit der Software können Änderungen am Datenmodell in einer Kopie vorgenommen werden, ohne das gemeinsame Systemmodell zu überschreiben.	A
fC4	Mit der Software können Daten im Systemmodell als Favorit markiert werden.	I
fC5	Die Software informiert umgehend über aktuell eingecheckte Systemmodelländerungen und lässt mich entscheiden diese direkt oder später zu übernehmen.	A
fC6	Die Software kann den aktuellen Status des Systemmodells darstellen, ohne auf alle Commits der anderen Ingenieure zu warten.	I
fC7	Mit der Software können Daten als sichtbar und unsichtbar deklariert werden.	I
fC8	Die Software kann darstellen, ob Daten aktuell von anderen Ingenieuren bearbeitet werden oder ob diese eingeloggt sind.	A
fC9	Mit der Software können Kommentare an spezielle Ingenieure am Systemmodell hinterlegt werden.	A
fC10	Die Software kann darstellen, was sich seit der letzten Session geändert hat.	A
fC11	Mit der Software kann ein Workspace mit einem anderen Ingenieur geteilt werden.	A
fC12	Die Software bietet einen toolgestützten Change-Management-Prozess für jegliche Systemmodellelemente an.	A
fC13	Die Software bietet ein toolgestütztes Task-Management für jegliche Systemmodellelemente an.	A
fC14	Mit der Software kann das Systemmodell auch offline bearbeitet werden können.	M

Tabelle 5.2: Funktionale Anforderungen: Concurrent Engineering

Schnittstellen & Export von Daten

Nr.	Anforderung	Kano
fS1	Die Software stellt das Systemmodell über eine Web-API zur Verfügung.	A
fS2	Die Software kann externe Software über deren Web-API ansteuern.	I
fS3	Die Software kann das Systemmodell exportieren.	M

Tabelle 5.3. Funktionale Anforderungen: Schnittstellen & Export von Daten

Produktlinien & Versionierung

Nr.	Anforderung	Kano
fP1	Die Software bietet eine Versionsverwaltung für das Systemmodell an.	M
fP2	Die Software ermöglicht es frühere Systemmodelle wiederzuverwenden.	M
fP3	Die Software ermöglicht es Varianten eines Systems zu erstellen, die eine gemeinsame Basis haben.	A

Tabelle 5.4: Funktionale Anforderungen: Produktlinien & Versionierung

Authentifizierung & Autorisierung

Nr.	Anforderung	Kano
fA1	Die Software bietet eine externe Authentifizierung mit DLR-User-Daten an.	A
fA2	Die Software ermöglicht, dass bestimmte Projekte, Workspaces und Teile des Systemmodells nur für gewisse Benutzerrollen zugänglich sind.	O
fA3	Die Software ermöglicht es verschiedene Projekte zu gruppieren.	A

Tabelle 5.5: Funktionale Anforderungen: Authentifizierung & Autorisierung

Nicht-funktionale Anforderungen

Leistungseffizienz

Nr.	Anforderung	Kano
nfL1	Die Software skaliert mit der Last.	I
nfL2	Die Software ist günstiger oder gleichteuer als die aktuelle Lösung.	A

Tabelle 5.6: Funktionale Anforderungen: Leistungseffizienz

Kompatibilität

Nr.	Anforderung	Kano
nfK1	Die Software kann mit anderer Software Daten austauschen.	A
nfK2	Die Software kann auf bereits existierender Hardware verwendet werden.	M
nfK3	Die Software kann von verschiedenen Geräten aus verwendet werden.	M

Tabelle 5.7: Funktionale Anforderungen: Kompatibilität

Zuverlässigkeit

Nr.	Anforderung	Kano
nfZ1	Die Software ist bei hoher Last erreichbar.	M
nfZ2	Die Software ist bei Fehlern betriebsfähig.	M
nfZ3	Die Software kann bei einem Absturz die Daten wieder in den vorherigen Zustand wiederherstellen.	M

Tabelle 5.8: Funktionale Anforderungen: Zuverlässigkeit

Sicherheit

Nr.	Anforderung	Kano
nfS1	Die Software speichert die Daten des Systemmodells verschlüsselt.	I
nfS2	Die Software verschlüsselt die Daten des Systemmodells auf dem Kommunikationsweg.	I
nfS3	Die Software speichert nicht das komplette Systemmodell lokal.	I
nfS4	Die Software protokolliert Zugriff und Änderungen am Systemmodell.	M
nfS5	Die Software stellt sicher, dass die Systemmodelle aus Geheimhaltungsgründen nur von berechtigten Ingenieuren genutzt werden können.	M
nfS6	Die Software kann auf eigener Hardware deployt werden.	I
nfS7	Die Software kann offline bzw. ohne externe Internetverbindung verwendet werden.	M

Tabelle 5.9: Funktionale Anforderungen: Sicherheit

Wartbarkeit

Nr.	Anforderung	Kano
nfW1	Die Software ist Open-Source und hat eine freie Lizenz.	I
nfW2	Die Software kann durch Plugins/Extensions erweitert werden.	A

Tabelle 5.10: Funktionale Anforderungen: Wartbarkeit

Portabilität

Nr.	Anforderung	Kano
nfP1	Die Software bietet einen leichten Zugriff auf das Systemmodell.	M
nfP2	Die Software ist auf Standard-Hard- und -Software lauffähig.	M

Tabelle 5.11: Funktionale Anforderungen: Portabilität

5.2 Architektur des Clusters

Wie im Stand der Technik bereits festgestellt, besitzt eine lokallaufende webbasierte Anwendung noch viele Nachteile einer Desktopanwendung. Dazu zählt, dass zum Beispiel die Serverseite entweder separat oder innerhalb eines Frameworks wie Electron betrieben werden muss. Dadurch muss die Software lokal gewartet werden und es stehen nur die Ressourcen des lokalen Computers zur Verfügung.

Damit einem Ingenieur eine Entwicklungsumgebung im Webbrowser bzw. in der Cloud für einen leichten Zugriff auf das Systemmodell zur Verfügung gestellt werden kann, müssen die benötigten Funktionalitäten extern (außerhalb vom Rechner des Ingeneiurs) bereitgestellt werden (Anforderung *nfP1*). Als Lösung wird wie bei Eclipse Che und Gitpod ein Cluster vorgeschlagen, das die benötigten Softwares und Funktionen bereitstellt. Dadurch, dass das Cluster

dem Softwaresystem zugrunde liegt, erfüllt es primär nicht-funktionale Anforderungen und ermöglicht es den anderen Softwarekomponenten, ihre Funktion bereitzustellen.

Im Kontext von MBSE für Raumfahrtsysteme werden sensible Daten verarbeitet. Deswegen ist ein Deployment in einer Public oder Private Cloud nicht immer möglich. Mit der Nutzung von cloud-native Technologien wie Container und Microservices kann die zugrunde liegende Umgebung inklusive Rechenressourcen abstrahiert werden. Die Architektur des Clusters ist in Abbildung 5.3 dargestellt.

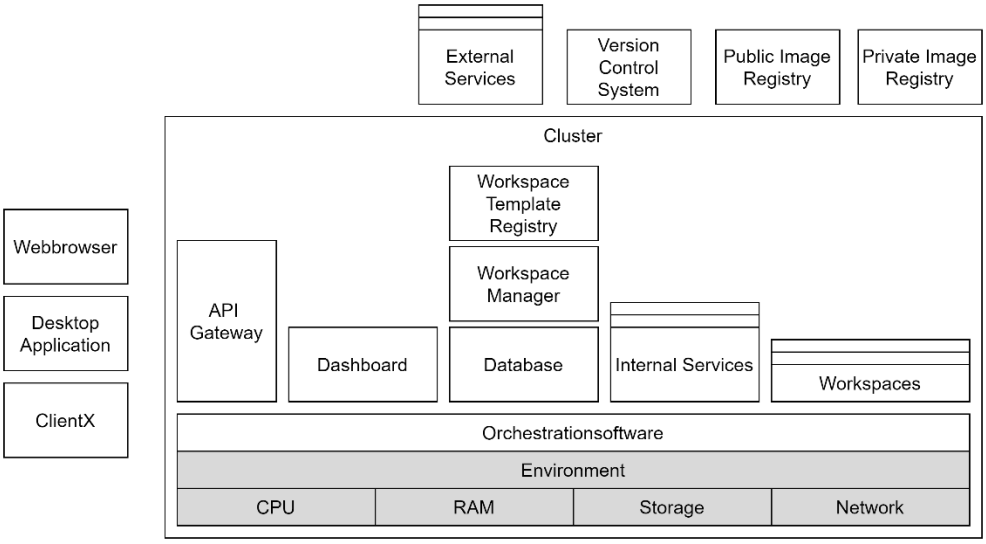


Abbildung 5.3: Cluster Architektur

Container haben sich als Alternative für virtuelle Maschinen etabliert, insbesondere in Microservice-Architekturen. Container-Laufzeiten alleine eignen sich für den Produktivbetrieb aber nur bedingt, da Container abstürzen können (Anforderung *nfZ2*) oder die Anforderung bestehen kann, Anfragen auf mehrere Container zu verteilen (Anforderungen *nfZ1* & *nfL1*). Für diese Aufgabe wird eine Orchestrierungssoftware benötigt, die die Container verwaltet. Die Orchestrierungssoftware abstrahiert außerdem die darunterliegende Umgebung für die Container, die am Ende die Hardwareressourcen wie CPU, Arbeitsspeicher und Netzwerk zur Verfügung stellt und verwaltet. Dadurch kann das Cluster auf einer beliebigen Umgebung betrieben werden (Public-Cloud, Private-Cloud, On-Premise etc.). Infolgedessen ist das Softwaresystem portabel (Anforderung *nfP2*), indem es auf beliebiger Hardware deployt werden kann. Durch die Nutzung eines Clusters besitzt das Softwaresystem eine erhöhte Komptabilität, indem es auf (falls vorhanden) bereits bestehender Hardware verwendet werden kann (Anforderung *nfK2*). Die Sicherheit kann dadurch auch erhöht werden, indem das System aufgrund von Geheimhaltungsklauseln auf eigener Hardware deployt wird (Anforderung *nfS6*). Wie bei

Gitpod und Eclipse Che ist es sinnvoll, die Dienste innerhalb des Clusters in Gruppen aufzuteilen. Die Aufteilung ist in Abbildung 5.4 dargestellt.

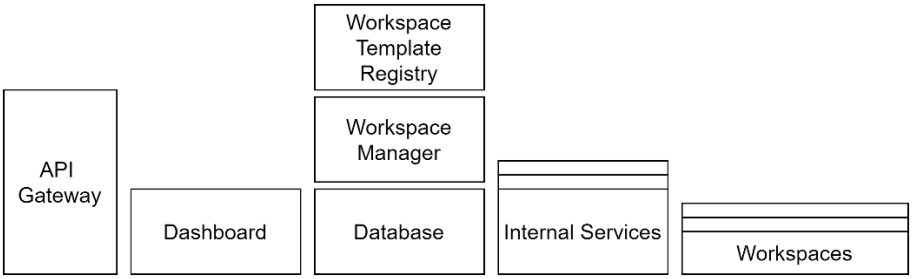


Abbildung 5.4: Dienste innerhalb des Clusters

Innerhalb des Clusters werden fünf Arten von Diensten unterschieden.

- **API Gateway**: Routet den Netzwerkverkehr und übernimmt die Authentifizierung und Autorisierung
- **Dashboard**: Startseite des Nutzers zur Verwaltung seiner Workspaces
- **Cluster Services**: Dienste die innerhalb des Clusters aber nicht vom Nutzer verwendet werden
- **Internal Services**: Dienste die allen Workspaces zur Verfügung stehen
- **Workspaces**: On-Demand erstellte Gruppe an Containern, die vom Ingenieur verwendet werden, um seine Arbeit auszuführen

Die Gruppe der Internal Services findet sich weder bei Eclipse Che noch Gitpod wieder. Die unterschiedlichen Dienste werden im Folgenden näher erläutert. Von außen wirkt das Cluster wie eine einzelne Software, die über das Gateway erreicht werden kann. Dabei können ein Webbrowser oder eine beliebige Desktop-Anwendung oder sonstige Software mit dem Gateway interagieren. Durch die Unterstützung verschiedener Clients wird eine geringe Nutzungshürde (*nfP1*) gewährleistet, indem keine Installation von Software zur Verwendung benötigt wird.

Das Gateway ist der Einstiegspunkt in das Cluster und übernimmt die Authentifizierung und Autorisierung (Anforderung *fA2*), da sie sich bereits bei den vorgestellten Lösungen als sinnvoll herausgestellt hat. An dieser Stelle lässt sich dann auch ein rollenbasiertes Zugriffsmanagement implementieren. Über das Gateway könnte auch eine Authentifizierung & Autorisierung mit einem bereits bestehendem Drittanbietersystem (Anforderung *fA1*) implementiert werden. Das Gateway leitet die Anfragen an den entsprechenden Dienst weiter. Das Gateway stellt dabei sicher, dass On-Demand erstellte Workspaces von außen zugänglich gemacht werden. Ein statisches Mapping wie bei einer klassischen Geschäftsanwendung ist hierbei nicht ausreichend, da die Workspaces nicht im Voraus bekannt sind.

Das Dashboard wird von den Ingenieuren direkt verwendet und dient als Einstiegspunkt und zum Verwalten ihrer Workspaces. Über das Dashboard erfolgt außerdem die Verbindung über das Gateway mit den Workspaces des Nutzers.

Cluster Services sind Dienste, auf die der Nutzer nicht direkt zugreift, die aber für die Funktion des Clusters essentiell sind. Zu den Cluster Services gehört unter anderem der Workspace Manager, der das Anlegen, Starten, Stoppen und Löschen der Workspaces übernimmt. Aus welchen Komponenten ein Workspace bestehen soll, erfährt dieser vom Dashboard über eine Konfigurationsdatei (Siehe Kapitel 5.3), die aus der Workspace Template Registry stammt oder selbst bereitgestellt wird. Die Workspace Template Registry stellt vordefinierte Konfigurationsdateien für Workspaces (Stacks) zur Verfügung, mit denen der Nutzer über das Dashboard Workspaces anlegen kann. In der Datenbank (Database) werden Informationen über die Nutzer sowie Metadaten über die Workspaces gespeichert.

Die internen Dienste (Internal Services) bieten Services für alle Workspaces an. Der Grund für das Anbieten von Internal Services auf Clusterebene ist, dass Simulationen die Ressourcen eines Workspaces überschreiten können oder Lizenzbestimmungen die Anzahl der laufenden bzw. installierten Instanzen beschränken können. Des Weiteren können diese genutzt werden, um Simulationsergebnisse zu teilen, um ein Ausführen bei jedem Teilnehmer zu vermeiden. Außerdem können Lizenzkosten einer Software über einen clusterweiten Internal Service minimiert werden (Anforderung *nfL2*).

Die Workspaces (Siehe Kapitel 5.3) beinhalten eine Menge an Containern, die bei Bedarf provisioniert wird. Der Nutzer hat dabei vollen Zugriff auf den Workspace. Die Container stellen dabei die Funktionen bereit, die von dem Nutzer benötigt werden, um seine Arbeit durchzuführen. Die Workspaces laufen getrennt und eigenständig voneinander und anderen Diensten, damit sich diese nicht gegenseitig beeinflussen und der Zugriff leichter gesteuert werden kann (Anforderungen *fA2* & *nfS5*).

Neben den Internal Services, die innerhalb des Clusters laufen, wird auch eine Reihe von externen Diensten (External Services) benötigt. Internal und External Services unterscheiden sich darin, ob diese innerhalb oder außerhalb des Clusters laufen. Funktionell sind diese austauschbar. Die External Servicese aus Abbildung 5.3 sind erneut in Abbildung 5.5 dargestellt.

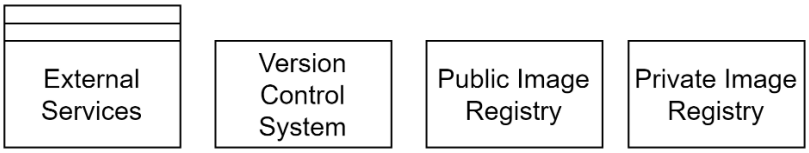


Abbildung 5.5: External Services

Die benötigten External Services können sich wie die Internal Services von Ingenieur zu Ingenieur unterscheiden. Beispiele für External Services sind Simulationen oder Berechnungen, die die Ressourcen des Clusters überschreiten oder nicht kostendeckend im eigenen Cluster durchgeführt werden können. Ein weiterer Grund kann auch die mangelnde Verfügbarkeit des Dienstes im Cluster sein.

Zu den External Services, die von jedem Ingenieur benötigt werden, gehört ein Versionskontrollsystem (Version Control System - VCS) sowie eine öffentliche und private Registry für Container-Images. Die External Services werden im Rahmen der Arbeit nicht implementiert. Es wird auf bereits vorhandene Software zurückgegriffen.

Das VCS dient der Persistierung der Projektdaten im Workspace. Der Grund dafür ist, dass keine Daten lokal bei dem Ingenieur gespeichert werden und Workspaces bei Bedarf erstellt und gelöscht werden. Dadurch müssen die Projektdaten an einer anderen Stelle persistiert werden. Zur Wahrung der Kompatibilität mit aktuellen Anwendungen wird hierfür ein VCS vorgeschlagen. Dadurch kann das VCS beim Start des Workspaces verwendet werden, um es mit Dateien aus dem zu bearbeitenden Projekt zu füllen.

Dadurch können mehrere Nutzer mit den Daten des gleichen Projekts arbeiten. (Anforderung *fC1*) Des Weiteren wird das VCS für die Synchronisierung zwischen den Workspaces beim Concurrent Engineering (siehe Kapitel 5.5) verwendet. Dabei wird von dem verwendetem VCS erwartet, dass es Systeme für ein Aufgaben- und Änderungsmanagement (Anforderungen *fC12* & *fC13*) bereitstellt.

Die öffentliche Registry für Container-Images (Public Image Registry) beinhaltet alle Container-Images, die für die Orchestrationssoftware sowie das verwendete Diensten benötigt werden. Die private Registry für Container-Images (Private Image Registry) beinhaltet die Container für selbst entwickelte oder nicht öffentlich zugängliche Container-Images. Beispiele dafür sind intern entwickelte Dienste oder lizensierte Dienste.

In Szenarien, in denen das Cluster keinen externen Internetzugriff besitzt, können die Container-Images aus der Public Image Registry in der Private Image Registry gehostet werden. Die

External Services werden von der vorgeschlagenen Lösung nicht implementiert, sondern lediglich eingebunden. Damit das Cluster auch ohne externe Internetverbindung verwendet werden kann, müssen alle benötigten Funktionen durch das Cluster (inklusive der Internal Services) oder durch External Services abgedeckt werden, die im selben Netzwerk laufen (Anforderung *nfS7*).

5.3 Workspaces

Die Workspaces sind jeweils eigenständige Microservice-Anwendungen, die von dem Workspace-Manager verwaltet werden. Die vorgeschlagene Workspace-Architektur orientiert sich dabei an die Workspace von Eclipse-Che, die ebenfalls einer Microservice-Architektur besitzen und aus mehreren Containern bestehen. Die Architektur des vorgeschlagenen Workspaces ist in Abbildung 5.6 dargestellt

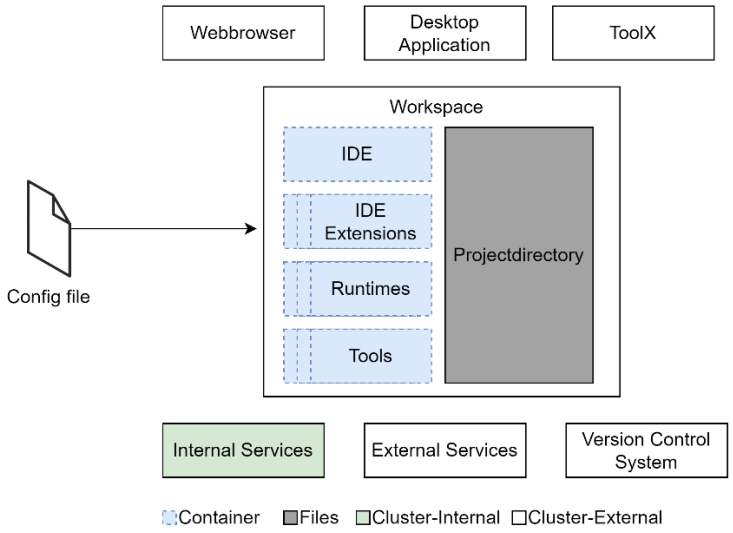


Abbildung 5.6: Konzept Workspace

Bei Eclipse Che und Gitpod hat sich ein deklarativer Ansatz über eine Konfigurationsdatei (Config File) für die Erstellung eines Workspaces bewährt. Um eine Reproduzierbarkeit der Workspaces zu gewährleisten, wird ebenfalls ein deklarativer Ansatz gewählt. Die verwendete Workspace-Manager (Siehe vorheriges Kapitel) muss dabei in der Lage sein, aus einer Konfigurationsdatei den Workspace zu erstellen.

Im Workspace befinden sich im Projektverzeichnis (Projectdirectory) die Projektdateien, zu denen das Systemmodell oder andere Dateien gehören, die vom Ingenieur für seine Arbeit

benötigt werden. Dadurch liegen die Projektdateien nicht lokal auf dem Rechner des Ingenieurs, was das Entwenden von Daten erschwert (Anforderung *nfS3*).

Da sich die benötigten Anwendungen bei jedem Ingenieur unterscheiden können, wird wie bei Eclipse Che ein Workspace mit mehreren Containern vorgeschlagen. Dadurch kann der Workspace um weitere Software erweitert werden (Anforderung *nfW2*). Die dargestellten Containerarten unterscheiden sich vom Aufbau nicht voneinander und sollen mögliche Arten von Diensten darstellen, die über einen Container bereitgestellt werden können. Die verschiedenen Containerarten sind optional und an die Anforderungen des einzelnen Ingenieurs anzupassen. Das bedeutet, dass alle Container inklusive der IDE ausgetauscht oder ausgelassen werden können, um dem Ingenieur Flexibilität bei der Erstellung des Workspace zu bieten. Dabei können auch mehrere Dienste im selben Container laufen. Die Container können Zugriff auf das gemeinsame Projektverzeichnis erhalten. Im Rahmen der Arbeit wird davon ausgegangen, dass immer eine IDE benötigt wird. Insbesondere bei technisch weniger versierten Personen bietet die IDE eine grafische Benutzeroberfläche über den Browser, die einen leichten Einstieg bietet. Das macht die IDE aus Nutzersicht zum wichtigsten Dienst im Workspace.

Zu der möglichen benötigten Software gehören u.a. ein Container für die Entwicklungsumgebung (IDE) sowie Container für Erweiterungen der IDE, falls diese nicht selbst im IDE-Container laufen können. Dazu zählen zum Beispiel Language-Server, die die Unterstützung für andere Sprachen bereitstellen. Der IDE-Container stellt eine Entwicklungsumgebung als Webanwendung für Webbrowser zur Verfügung.

Des Weiteren kann der Workspace Laufzeitumgebungen (Runtimes) benötigen, die zum Beispiel Code ausführen (Java-Runtime). Eine weitere Art von Containern kann Werkzeuge (Tools) enthalten, wie zum Beispiel Simulationsprogramme oder Datenbanken, die weder in der Benutzeroberfläche oder im IDE-Container laufen.

Neben dem Austausch über ein gemeinsames Laufwerk können die Container sowohl innerhalb als auch außerhalb eines Workspaces über das Netzwerk kommunizieren. Bei der Kommunikation mit Diensten außerhalb des Workspaces wird zwischen cluster-interner und cluster-externer Kommunikation unterschieden. Die Kommunikation außerhalb des Workspaces wird außerdem auch für das Nutzen von Diensten benötigt (Anforderung *fS2*), die nicht innerhalb des Workspaces laufen. Gründe dafür sind Lizenzbestimmungen der benötigten Software oder Ressourcenlimits des Clusters oder Workspaces.

Neben der Nutzung von Diensten kann der Workspace ebenfalls Dienste nach außen anbieten (Anforderung *fS1*). Dadurch kann das Systemmodell auch ohne die Entwicklungsumgebung

verwendet werden, wenn ein entsprechender Container das Systemmodell über eine API anbietet (Anforderung *nfK1*). Durch diese Bidirektionalität können Anwendungen wie Simulationsprogramme im Workspace von Desktopanwendungen verwendet werden und umgekehrt.

Der Nutzer kann über einen Webbrowser, einer Desktopanwendung oder einem beliebigen anderen Client und damit mit einem beliebigen Gerät auf den Workspace bzw. auf die Dienste der Container zugreifen (Anforderung *nfK3*), sofern diese vom Cluster nach außen freigegeben sind. Durch das Bereitstellen des Workspaces über einen Link können auch andere Ingenieure über die IDE oder Dienste im Workspace auf das Systemmodell zugreifen (Anforderung *fC11*).

5.4 Entwicklungsumgebung und Modellierung

Wie in dem vorherigen Kapitel beschrieben ist die Entwicklungsumgebung (IDE) der wichtigste Dienst für den Nutzer im Workspace. Die IDE ist eine Möglichkeit für den Ingenieur, ein System zu modellieren.

Um den Ingenieuren eine einfache Möglichkeit zu bieten, das Systemmodell zu bearbeiten, wird eine webbasierte Benutzeroberfläche vorgeschlagen. Die webbasierte Benutzeroberfläche, die von dem IDE-Dienst angeboten wird, benötigt keine Installation, was eine einfache Verwendung ermöglicht (Anforderung *nfK3* & *nfP1*). Das Konzept der Entwicklungsumgebung ist in Abbildung 5.7 dargestellt.

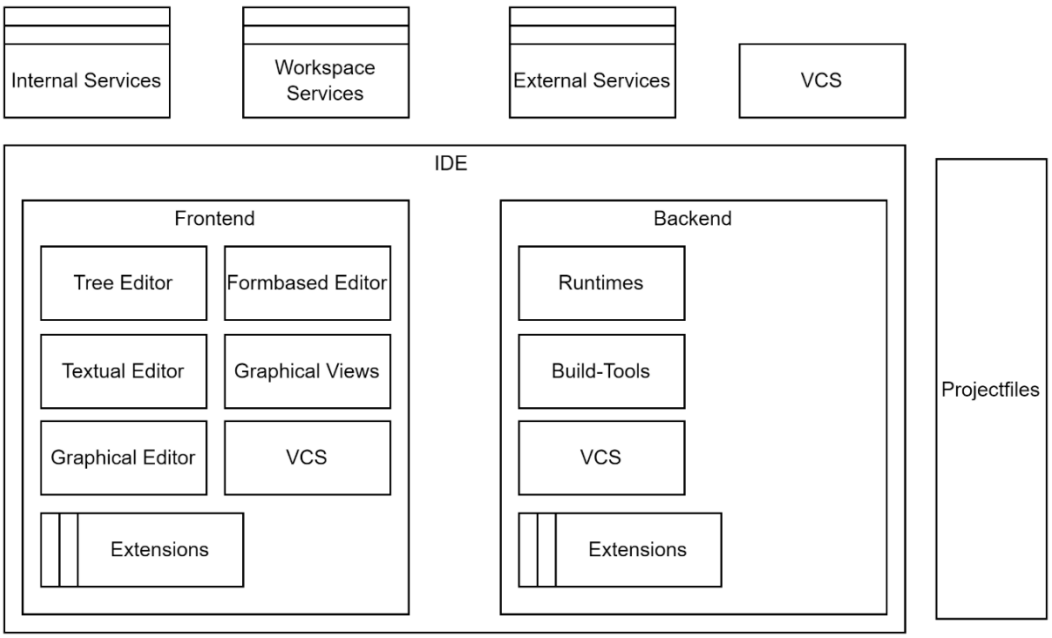


Abbildung 5.7: Aufbau und Funktionen der Entwicklungsumgebung

Die IDE bietet dem Ingenieur eine Standardentwicklungsumgebung, mit der das Systemmodell bearbeitet werden kann. Da es sich um eine webbasierte Benutzeroberfläche handelt, benötigt IDE einen Client (Frontend)- und Server (Backend)-Teil.

In der IDE erfolgt mit dem Modellieren die Hauptarbeit des Ingenieurs. Zu Modellieren werden verschiedene Editoren benötigt, mit der auch verschiedene Sichten des Systemmodells dargestellt werden können (Anforderung *fM3*). Zu den Editoren gehören:

- Baumeditor:
    - Darstellung und Modellierung hierarchischer Strukturen
- Formular-basierte-Editoren:
    - Dateneingabe
- Grafische Editoren:
    - Darstellung und Modellierung
- Grafische Darstellungen:
    - Darstellungen von Berechnungen oder Simulationen
- Textuelle Editoren
    - Schreiben von Quellcode oder Konfigurationsdateien

Das Backend übernimmt hierbei die Synchronisierung der einzelnen Editoren. Die Anforderung der Erweiterbarkeit (Anforderung *nfW2*) gilt neben dem Cluster und den Workspaces auch für die IDE. Dabei sollten die Erweiterungen (Extensions) frei vom Ingenieur gewählt und installiert werden, damit die IDE an die Anforderungen des Ingenieurs weiter angepasst werden kann. Im Gegenzug zu bis jetzt vorgestellten Erweiterungsmöglichkeiten ist für die Anpassung der IDE über Extensions auch ohne technisches Wissen möglich.

Die IDE benötigt eine Reihe von Diensten, die sie entweder selbst oder über andere externe Dienste bereitgestellt bekommt. Dazu gehören Laufzeitumgebunden oder Build-Tools sowie andere Werkzeuge wie einen VCS-Client, der für das Concurrent Engineering (Siehe Kapitel 5.5) benötigt wird und beim Auflösen von Merge-Konflikten hilft. Ein weiteres Beispiel für einen Dienst für die IDE wäre die Darstellung, welche Ingenieure das Systemmodell gerade bearbeiten bzw. sich in einen Workspace eingeloggt haben (Anforderung *fC8*). Dabei können diese Dienste entweder über die IDE, Workspaces, das Cluster, externe Dienste oder eigene Erweiterungen bereitgestellt werden. Des Weiteren benötigt die IDE Dateizugriff auf die Projektdateien sowie Netzwerkzugriff auf andere Dienste innerhalb des Workspaces und Clusters und außerhalb des Clusters, um diese über ihre API anzusteuern (Anforderung *fS2*). Für Anwendungen, die keine API besitzen, muss das Systemmodell in gängigen Formaten exportiert werden können (Anforderung *fS3*)

Die Hauptfunktion der IDE ist das Modellieren eines Systems. Zum Modellieren des Systems wird eine Sprache (Modellierungssprache) benötigt, die ein Vokabular zur Verfügung stellt, mit dem das System beschrieben werden kann. Das Konzept der Modellierungssprache ist in Abbildung 5.8 dargestellt.

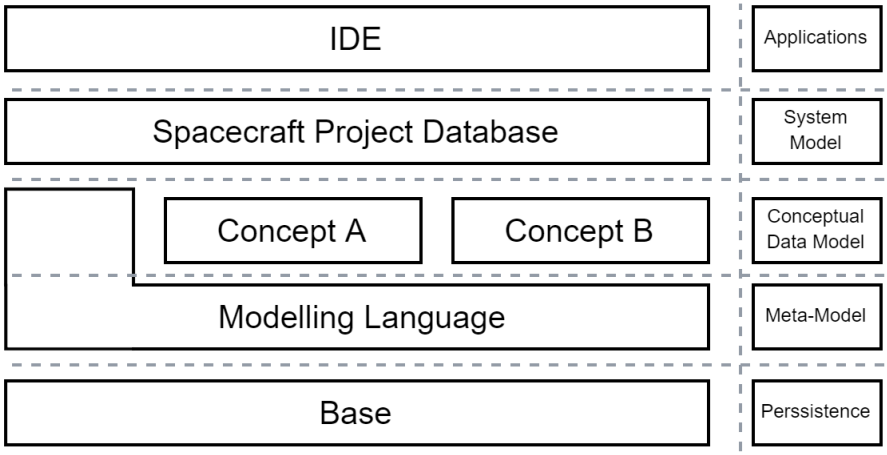


Abbildung 5.8: Konzept der Modellierungssprache

Die Schichten des Datenmodells der Modellierungssprache sind eine vereinfachte Version des Datenmodells von Virtual Satellite 4. Als Basis (Base) wird eine Persistenzschicht benötigt, mit der das Datenmodell abgespeichert werden kann. Aufbauend auf der Basisschicht wird eine Modellierungssprache definiert, die grundlegende Modellierungsfunktionen besitzt. Auf Basis der Modellierungssprache können Konzepte definiert werden, die die Modellierungssprache erweitern. Die Modellierungssprache kann dann für das Definieren des Systemmodells verwendet werden. Das Systemmodell selbst kann dann von Anwendungen wie der IDE verwendet werden.

Raumfahrtsysteme haben eine Lebensspanne von Jahrzehnten, in der sie sich kontinuierlich weiterentwickeln. Deswegen ist es notwendig, dass das Systemmodell erweitert werden kann. Um das zu erreichen, wird eine vereinfachte Version von Engineering Categories verwendet, die in VSD oder Virtual Satellite 4 Anwendung finden. Die Konzeption des Datenmodells mit Engineering Categories sind in Abbildung 5.9 dargestellt

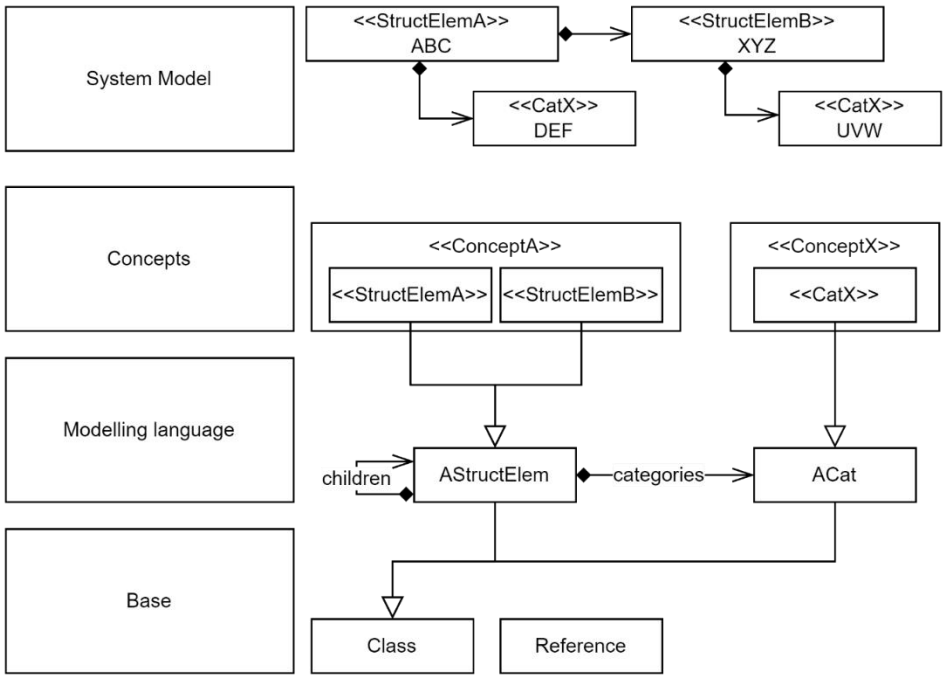


Abbildung 5.9: Schichten des Datenmodells mit Engineering Categories

Als Basis wird eine Sprache benötigt, die Klassen und Referenzen bereitstellt. Darauf aufbauend kann eine Modellierungssprache definiert werden, die aus Structural Elements und Categories bestehen. Structural Elements stellen die Struktur des Systemmodells dar, während die Categories Container für Daten sind. Dabei stellt ein Structural Element eine rekursive Datenstruktur dar, die weitere Structural Elements als Kinderelemente enthalten kann. Des Weiteren referenziert ein Structutal Element eine beliebige Anzahl an Categories.

Dadurch adressiert das Konzept die Anforderungen für das generelle Modellieren (Anforderung *fM1*) des Systems sowie das Aufteilen in Subsysteme (Anforderung *fM2*). Das Datenmodell ermöglicht außerdem die Erstellung von Sichten (Anforderung *fM3*), indem bereits vorhandene Daten oder durch Konzepte eingebrachte Daten für Sichten verwendet werden können.

Aufbauend auf der Modellierungssprache können Konzepte definiert werden, die Structural Elements sowie Categories enthalten. Durch Konzepte kann die Modellierungssprache erweitert werden (Anforderung *fM4*). Structural Elements und Categories aus den Konzepten können dann verwendet werden, um ein Systemmodell zu erstellen. Die Konzepte ermöglichen außerdem das Definieren von Berechnungen und Referenzen (Anforderung *fM5*) sowie das Erstellen von Anforderungen inklusive Verknüpfungen (Anforderung *fM6*). Berechnungen können dabei zu anspruchsvoll für die Entwicklungsumgebung sein und der Einsatz von internen oder externen Cluster-Services erforderlich machen. Dasselbe gilt für das Durchführen von

Simulationen (Anforderung *fM7*), Konsistenzprüfungen (Anforderung *fM8*) oder Analysen von Änderungen (Impact Analysis) (Anforderung *fM9*). Weitere Daten, die über Konzepte erstellt werden können, sind das Markieren als Favorit (Anforderung *fC4*), das sichtbar/unsichtbar Schalten von Daten (Anforderung *fC7*) sowie das Hinterlegen von Kommentaren (Anforderung *fC9*). Über ein Konzept könnte, wie bei Virtual Satellite, das Übertragen von Daten in weitere Lebensphasen implementiert werden. Dies wird aber, um im Rahmen der Arbeit zu bleiben, nicht weiter betrachtet (Anforderung *fM10*).

5.5 Concurrent Engineering

Der Datenaustausch zwischen den Ingenieuren ist eine Kernanforderung beim Concurrent Engineering. Folgende Probleme können beim Concurrent Engineering auftreten [7]:

- Gleiche Daten können bearbeitet werden
- Berechnungen und Datenpropagation beeinflussen sich gegenseitig
- Das Mergen von Konflikten lenkt die Ingenieure von der eigentlichen Arbeit ab

Im Stand der Technik wurden mehrere Arten von Concurrent Engineering dargestellt. Die Ingenieure können auf verschiedene Arten auf das gleiche Systemmodell zugreifen (Anforderung *fC1*):

- Ein Workspace für alle Ingenieure
    - Sofortiges Synchronisieren
        - Keine Kontrolle über andere Änderungen
        - Hohes Konfliktpotenzial
        - Schnelle Änderungen
- Pro Ingenieur ein Workspace
    - Automatische Synchronisierung
        - Jeder Ingenieur arbeitet in eigenem Workspace
        - Änderungen werden nach einer festen Zeit synchronisiert oder durch eine Person mit erhöhter Autorität freigegeben
    - Benachrichtigungen bei Änderungen
        - Ingenieure würden visuell auf Änderungen hingewiesen
        - Synchronisierung kann bestätigt oder verschoben werden
    - VCS
        - Ingenieure arbeiten mit lokaler Arbeitskopie
        - Dadurch arbeitet jeder Ingenieur zunächst unabhängig voneinander
        - Synchronisierung erfolgt manuell bei Bedarf

Von den Ingenieuren wurde es als Begeisterungsmerkmal bewertet, wenn die Lösung einen toolgestütztes Change-Management-Prozess für Systemmodellelemente hätte (Anforderung *fC13*). Die Lösungen mit einem Workspace für alle Ingenieure bieten kaum Kontrolle über

Änderungen und ein hohes Konfliktpotential durch häufige Änderungen. Das gleiche gilt in abgeschwächter Form beim automatischen Synchronisieren der Workspaces. Ansätze, bei denen eine Person mit erhöhter Autorität eine automatische Synchronisierung von Änderungen freigibt, können ebenfalls den Arbeitsfluss der Ingenieure stören. Dasselbe trifft auf den Ansatz mit Benachrichtigungen zu, bei dem der Ingenieur bei Änderungen für seinen Workspace entscheidet, ob diese sofort oder später übernommen werden.

Im Rahmen der Arbeit wird sich für den Ansatz entschieden, bei dem jeder Ingenieur einen eigenständigen Workspace besitzt. Dadurch entscheidet der Ingenieur selbst, wann Änderungen veröffentlicht und wann Änderungen von anderen Nutzern übernommen werden. Im Vergleich zu den anderen Verfahren sinkt dadurch die Geschwindigkeit, in der Änderungen verteilt werden, dafür werden die Änderungen systematischer eingebracht. Die Entscheidung wird auch durch die Anforderungen der Ingenieure bestärkt, die eine Versionsverwaltung für das Systemmodell fordern (Anforderung *fP1 - Basismerkmal*). Dagegen spricht auf der anderen Seite die Anforderung, dass die Ingenieure über Änderungen umgehend informiert werden möchten (Anforderung *fC5* - Begeisterungsmerkmal). Da die Anforderung *fP1* eine höhere Wichtigkeit als *fC5* besitzt, wird der Anforderung *fP1* der Vorzug gegeben. In späteren Lebensphasen werden Änderungen pro Nutzer seltener, dafür arbeiten mehr Ingenieure an dem Systemmodell, weswegen ein VCS im Laufe des Raumfahrtlebenszyklus an Bedeutung gewinnt. Des Weiteren ist es über ein VCS-System möglich, den Zugriff auf Repositories auf definierte Nutzergruppen zu beschränken (Anforderung fA3) sowie Repositories zu gruppieren, um Zugriffsrechte zu vererben (Anforderung fA2). Im Folgenden wird das Concurrent Engineering über Workspaces vorgestellt. Das Konzept für das Concurrent Engineering über eigene Workspaces ist in Abbildung 5.10 dargestellt.

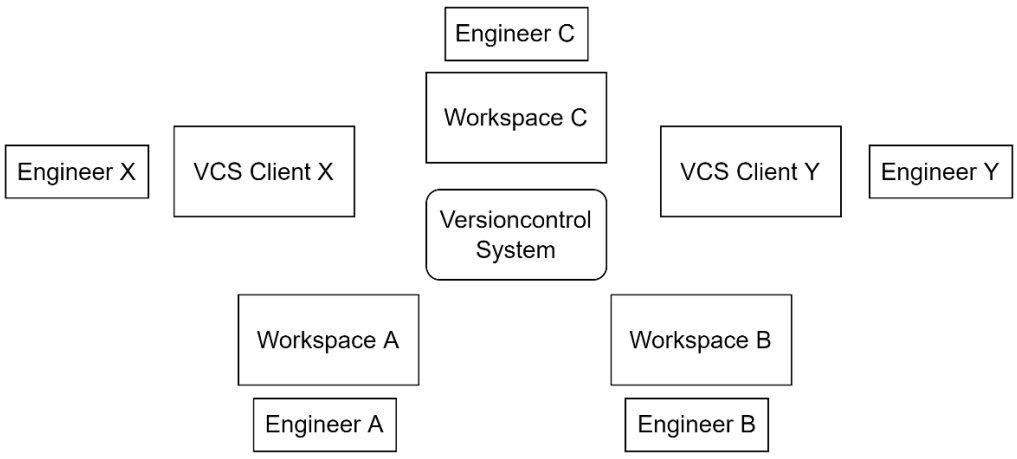


Abbildung 5.10: Concurrent Engineering über eigene Workspaces

Über das VCS können Änderungen des Systemmodells zwischen den einzelnen Ingenieuren synchronisiert werden. Dadurch, dass die Workspaces über eine URL zur Verfügung gestellt werden, kann dieser mit anderen Ingenieuren geteilt werden, um gemeinsam in einem Workspace an einem Problem zu arbeiten (Anforderung *fC11*). Dadurch muss ein anderer Ingenieur das Projekt bei sich selbst nicht aufsetzen, um mit einem anderen Ingenieur im selben Workspace zu arbeiten.

Von den Ingenieuren wurde es als Basismerkmal bewertet, dass das Systemmodell auch offline bearbeitet werden kann (Anforderung *fC14*). Durch die Nutzung eines VCS werden die Daten des Systemmodells extern von dem Cluster gehalten. Dadurch gibt es die Möglichkeit, das Systemmodell über das VCS runterzuladen und offline zu bearbeiten. Dabei können die Dienste des Clusters nicht verwendet werden und der Ingenieur muss andere Software zur Bearbeitung nutzen. Sobald wieder eine Internetverbindung besteht, können Änderungen mit dem VCS synchronisiert und die Dienste des Clusters erneut verwendet werden. Im Folgendem wird erläutert, wie die Anforderungen für das Concurrent Engineering erfüllt werden.

Über ein VCS können Änderungen des Systemmodells in Commits aufgezeichnet werden. Dabei stellt ein Commit eine Rücksetz-Einheit dar, zu der zurückgesprungen werden kann. Ein Commit enthält neben den Änderungen zum vorherigen Zustand Metainformationen wie den Autor sowie Kommentare zur Änderung.

Dadurch, dass das VCS die Änderungen an dem Projekt protokolliert, kann der Ingenieur nachvollziehen, welche Änderungen seit seiner letzten Sitzung vorgenommen wurden (Anforderungen *fC10*) und in welcher Reihenfolge diese durchgeführt wurden. An dieser Stelle ist es dann auch möglich, schreibenden Zugriff auf das Systemmodell mit Zeitpunkt und Ingenieur zu protokollieren (Anforderung *nfS4*). Durch die schrittweisen Änderungen wird es außerdem vereinfacht, Merge-Konflikte zu beheben (Anforderung *fC2).* Das Nachvollziehen der Änderungen als auch das schrittweise Auflösen von Merge-Konflikten setzt allerdings voraus, dass die Ingenieure das Systemmodell in kleinen Schritten aktualisieren.

Das VCS übernimmt dabei nicht nur die Aufgabe der Synchronisierung. Die Ingenieure wünschten sich, dass Änderungen am Systemmodell vorgenommen werden können, ohne das gemeinsame Systemmodell zu überschreiben (Anforderung *fC3*). Dieser Umstand ist in Abbildung 5.11 dargestellt.

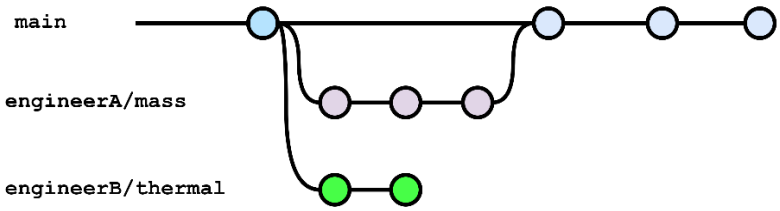


Abbildung 5.11: Abzweigung vom gemeinsamen Systemmodell

Das VCS muss dafür die Möglichkeit anbieten, von dem gemeinsamen Stand des Systemmodells abzuzweigen. Die im Folgenden vorgestellte Methode orientiert sich an dem Branching Modell Trunk-Based, weil es aufwendige Merge-Konflikte vermeidet, indem langlaufende Branches verhindert werden [103].

Innerhalb der Abzweigung kann der Ingenieur Änderungen veröffentlichen, ohne die anderen Ingenieure zu beeinflussen. Sollen die Änderungen mit den anderen Ingenieuren geteilt gemacht werden, kann der Ingenieur die Änderungen in den Hauptbranch (main) einfließen lassen. Branches mit fehlgeschlagenen Experimenten können dabei gelöscht werden, ohne einen Einfluss auf den Hauptbranch zu haben. Dabei kann der Hauptbranch geschützt werden und nur Personen mit erhöhter Autorität wird das Einfließen erlaubt. Das kann dazu genutzt werden, Änderungen auf eine systematische Art und Weise (Change-Management-Prozess) einzubringen (Anforderung *fC12*) und Aufgaben zu verwalten (Anforderung *fC13*). Das Konzept für das Change- und Task-Management ist in Abbildung 5.12 dargestellt.

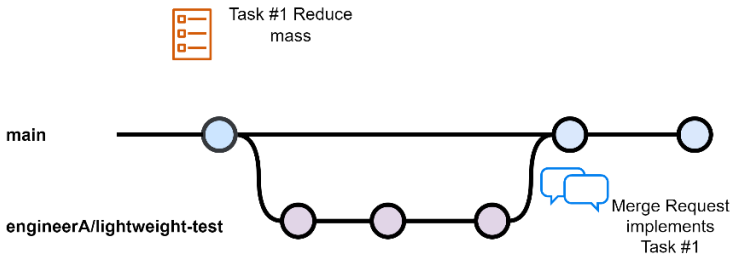


Abbildung 5.12: Change- und Task-Management

Zu dem Aufgabenmanagement gehört einmal die Dokumentation von Aufgaben. Das VCS muss die Funktion bereitstellen, Aufgaben zu definieren. Aufgaben werden in einer Reihe von Commits abgearbeitet und müssen dann kontrolliert in das gemeinsame Systemmodell integriert werden. Als Change-Management-Prozess werden die von GitHub und GitLab bekannten Merge bzw. Pull-requests vorgeschlagen. So können Änderungen aus einem Branch zum Beispiel erst nach der Freigabe eines anderen Ingenieurs oder Teamleiters in den Hauptbranch einfließen. Innerhalb des Freigabeprozesses können dem Branch weitere Commits

hinzugefügt werden, bis dieser in den Hauptbranch einfließt. Des Weiteren können in dieser Phase automatische Validierungen durchgeführt werden, die verhindern, dass Dateien in den gemeinsamen Stand einfließen, die nicht den Konventionen entsprechen (z.B. Namen der Subsysteme müssen großgeschrieben werden oder dürfen keine negative Masse besitzen.).

Zur Dokumentation von Entscheidungen können innerhalb der Commits oder des Merge Requests die Aufgaben verlinkt werden. Dafür können den Commits Nachrichten angehängt werden, die die Änderungen dokumentieren und Verlinkungen zu Aufgaben verwenden. Das VCS muss dann für die Verlinkung zwischen Aufgaben, Commits und Merge Requests sorgen. Innerhalb des Merge Request können auch Ingenieure markiert werden, wodurch sie nach einer Benachrichtigung zu den Änderungen Stellung nehmen (Anforderung *fC9*) und bei Änderungen benachrichtigt werden können (Anforderung *fC4*).

Die Ingenieure fordern außerdem, dass frühere Systemmodell wiederverwendet werden können Anforderung (*fP2*), um nicht bei jeder neuen Designsitzung von vorne anfangen zu müssen. Eine Möglichkeit, Systemmodelle wiederverwenden zu können, ist, bereits existierende Systemmodelle über das VCS zu beziehen. Durch das VCS besitzen die Ingenieure auch unabhängig vom aktuellen Workspace Zugriff auf ihre Systemmodelle. Des Weiteren möchten die Ingenieure Varianten von bereits existierenden Modellen erstellen, die eine gemeinsame Basis besitzen (Anforderung *fP3*). Die bisher beschriebenen Möglichkeiten beschränken sich auf ein Projekt bzw. Repository. Das Verwenden einer gemeinsamen Basis ist in Abbildung 5.13 dargestellt.

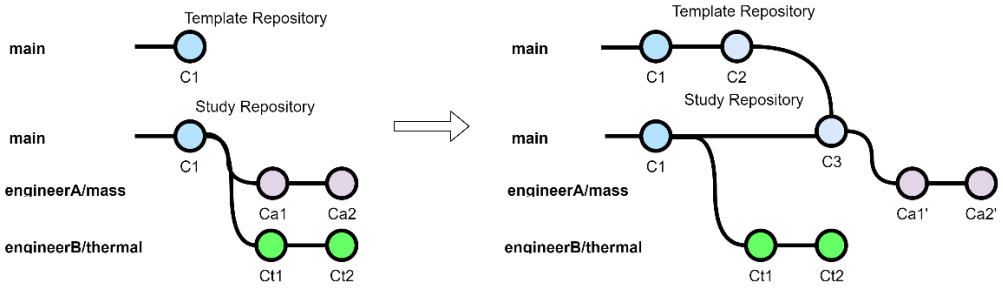


Abbildung 5.13: Nutzung einer gemeinsamen Basis

Zum Wiederverwenden von Systemen bzw. Projekten kann ein Repository (Template Repository) die Vorlage für ein anderes Repository darstellen. Die Vorlage kann die grundlegen Struktur eines Systems bzw. Projekts enthalten. Die Vorlage besitzt im oben gezeigten zu Beginn einen Commit. Die Vorlage wird dann für eine Designstudie in ein anderes Repository (Study Repository) kopiert. Die Ingenieure arbeiten in ihren Branches an dem Systemmodell im Study Repository. Wird die Vorlage aktualisiert, können diese Änderungen bei Bedarf in das Study

Repository übernommen werden. Dasselbe trifft auch auf die Ingenieure zu, die in ihrem Branch arbeiten. Sie können den Zeitpunkt selbst bestimmen, wann ihr Branch auf die geänderte Vorlage aktualisiert wird. In dem gezeigten Beispiel hat der erste Ingenieur seinen Branch auf die geänderte Vorlage aktualisiert, während der andere Ingenieur noch mit der alten Version arbeitet.

Die Anforderung, dass das gesamte Systemmodell inklusive nicht veröffentlichter Änderungen dargestellt werden kann, wurde als unerhebliches Merkmal klassifiziert (Anforderung *fC6*). Da das Zusammenführen verschiedener Versionen des Systemmodells wegen möglicher Konflikte nicht trivial ist und die Anforderung als unerhebliches Merkmal klassifiziert wurde, wird diese Anforderung im Rahmen der Arbeit nicht betrachtet.

6 Implementierung

Im folgenden Kapitel wird die Implementierung des Konzepts behandelt. Zunächst wird die Implementierung des Clusters vorgestellt. Danach wird auf die Implementierung der Workspaces eingegangen. Darauffolgend folgt die Implementierung der Entwicklungsumgebung auf Basis des Coffee Editors von dem EMF.Cloud Projekt. Abschließend wird die Implementierung des Concurrent Engineerings dargestellt. Bei der Implementierung wurde nur Open-Source Software mit freier Lizenz (Anforderung *nfW1*) verwendet.

6.1 Cluster

Aufgrund von Vorgaben des DLR kann die implementierte Lösung nicht bei einem Cloudanbieter gehostet werden. Auch das Betreiben im DLR On-Premise ist für studentische Arbeiten nicht möglich. Deswegen wird die Implementierung auf einem lokalen Rechner durchgeführt. Durch den Einsatz von Orchestrationssoftware kann die zugrundeliegende Hardware abstrahiert werden, wodurch die Implementierung zu einem späteren Zeitpunkt in einer Cloud deployt werden kann. Als Orchestrationssoftware kann OpenShift oder Kubernetes verwendet werden. Die lokale Version von OpenShift ist allerdings nur auf CentOS lauffähig. Auf der zur Verfügung stehenden Hardware stürzte CentOS in der virtuellen Maschine häufig ab. Kubernetes hingegen kann auf verschiedenen Linux Distributionen installiert werden, was die Portabilität (Anforderung *nfP2*) der Lösung erhöht. Im Rahmen der Arbeit traten außerdem keine Stabilitätsprobleme wie bei CentOS auf, weswegen Kubernetes als Orchestrationssoftware verwendet wird.

Auf Basis von Kubernetes stehen außerdem mehrere Softwarelösungen zur Verfügung, die die Verwaltung von Workspaces anbietet. Zu möglichen Lösungen zählen Eclipse Che, Gitpod und Theia.cloud. Alle Lösungen sind Open-Source und sind generell kostenlos oder bieten eine kostenlose Version an. Gitpod kann jedoch nur in der kostenpflichtigen Version ohne externe Internerverbindung verwendet werden. Eclipse Che ist aber die einzige bekannte Lösung, die es ermöglicht, einen Workspace aus mehreren Containern zusammenzusetzen. Da der Ansatz eine höhere Flexibilität versprach, setzt die Implementierung auf Eclipse Che. Die Implementierung des Clusters ist in Abbildung 6.1 dargestellt.

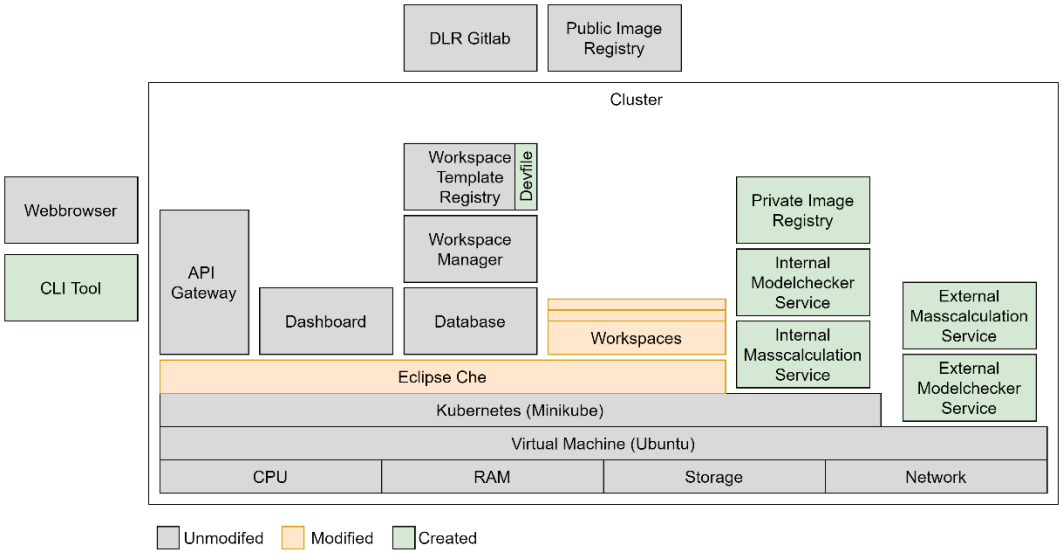


Abbildung 6.1: Implementierung Cluster

Aufgrund der zuvor genannten Einschränkungen wird das Cluster auf einer virtuellen Maschine deployt. Die virtuelle Maschine bildet die Basis, die die darunterliegenden Ressourcen virtualisiert. Auf der virtuellen Maschine werden aufgrund von Exportkontrollbestimmungen außerdem auch die externen Services deployt. Als Proof of Concept wurden zwei eigene Services implementiert. Der erste Service dient der Berechnung der Gesamtmasse eines gegebenen Systemmodells. Dieser wurde in NodeJS mit Express entwickelt und stellt eine REST-API zur Verfügung, über die eine Berechnung der Gesamtmasse der des übergebenen Systemmodells durchgeführt werden kann.

Der zweite Service analysiert anhand eines Flussdiagrammes über einen Modelchecker, ob das Flussdiagramm in einen Deadlock kommen kann. Als Modelchecker wird Storm verwendet. Storm verfügt über keine Web-API, weswegen hierfür ein weiteres NodeJS-Programm mit Express entwickelt wurde. Dadurch kann Storm über eine REST-API verwendet werden. Storm benötigt als Übergabeparameter einen Dateipfad zu der Datei, die das Flussdiagramm beschreibt. Um Performanzverluste durch das Dateisystem zu vermeiden, schreibt das NodeJS Programm das übergebene Systemmodell in das dev/shm Verzeichnis. Dieses Verzeichnis ist ein temporäres RAM-Verzeichnis, in das Dateien im Arbeitsspeicher abgelegt werden können. Nachdem Storm mit der Bearbeitung fertig ist, wird die Datei wieder gelöscht um den Arbeitsspeicher freizugeben.

Für den Betrieb von Kubernetes wird Minikube verwendet, da es die Installation und den Betrieb von Kubernetes vereinfacht. Minikube wird dabei mit dem Docker-Treiber verwendet. Innerhalb des Kubernetes Cluster läuft eine private Image Registry sowie die beiden selbst

implementierten Services als externe Services. Die private Image Registry beinhaltet dabei die Docker-Images der selbst implementierten Services sowie der IDE. Die anderen in dem Konzept definierten Services werden von Eclipse Che bereitgestellt. Die Installation erfolgt dabei über chectl, was ein Kommandozeilenwerkzeug für Eclipse Che ist. Für die Modifikationen am Cluster werden Helm sowie Terraform genutzt, um die Veränderungen am Cluster in Dateien festzuhalten. Damit in Eclipse Che eine eigene IDEs bzw. Workspaces verwendet werden können, wurde eine eigene Devfile definiert, die in der Workspace Template Registry gehalten wird. Als Versionskontrollsystem wird die GitLab-Instanz des DLR verwendet. Für die Images für Minikube als auch für die Standardkomponenten von Eclipse Che werden öffentliche Image Registies verwendet (Google Container Registry, DockerHub, RedHat Quay).

6.2 Workspaces

Im Rahmen der Arbeit wird ein Workspace implementiert, der alle benötigten Funktionen beinhaltet, die von einem Ingenieur für das Modellieren eines Raumfahrtsystems benötigt werden. Abbildung 6.2 zeigt die Implementierung des Workspaces.

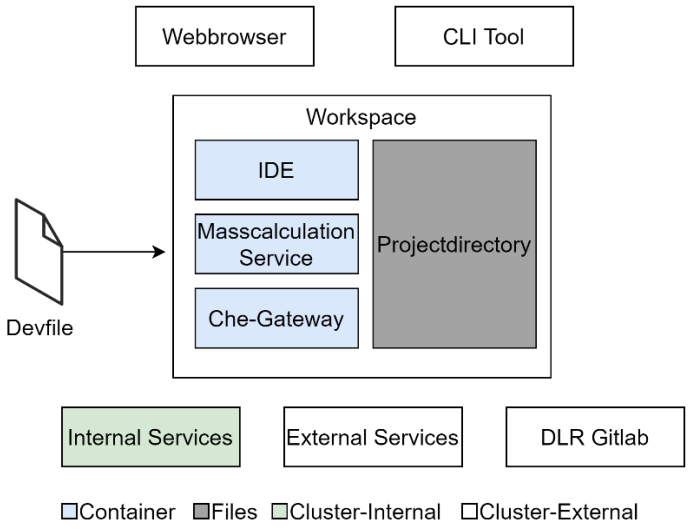


Abbildung 6.2: Implementierung Workspace

Für das Erstellen eines Workspaces benötigt Eclipse Che die Beschreibung des Workspaces als Devfile. Als prototypische Implementierung wird eine Devfile definiert, die neben der IDE auch einen Masscalculation Service beinhaltet. In Eclipse Che werden die Workspaces als Pod deployt, was sicherstellt, dass alle Container gemeinsam auf demselben Rechnerknoten ausgeführt werden. Von Eclipse Che wird automatisch ein weiterer Container (Che-Gateway) in dem Pod deployt, der die Kommunikation zu den anderen Che-Komponenten übernimmt.

Dadurch wird außerdem ermöglicht, dass der Workspace über einen Link erreichbar ist (Anforderung *fC11*).

6.3 Entwicklungsumgebung und Modellierung

EMF.Cloud wurde als geeignetes Framework identifiziert, um eine Entwicklungsumgebung für das Modellieren von Raumfahrtsystemen prototypisch zu implementieren. Während der Arbeit wurden zwei Möglichkeiten identifiziert, die Entwicklungsumgebung mit EMF.Cloud und über Eclipse Che zur Verfügung zu stellen:

- Che-Theia mit EMF.Cloud erweitern und als IDE in Eclipse Che verwenden
- Eclipse Theia mit EMF.Cloud erweitern und eine Devfile für Eclipse Che erstellen

Che-Theia ist eine für Eclipse Che modifizierte Version von Eclipse Theia. Sie bietet eine Integration mit Eclipse Che, wodurch von der Entwicklungsumgebung auf das Cluster zugegriffen werden kann. Die zweite Möglichkeit besitzt keine Integration mit Eclipse Che. Bei Che-Theia handelt es sich aber um eine stark modifizierte Version von Eclipse Theia und EMF.Cloud bietet ein Beispielprojekt an, indem EMF.Cloud bereits in Eclipse Theia integriert wird. Im Rahmen der Arbeit wird daher die zweite Möglichkeit gewählt, da sie eine bessere Basis für Anpassungen bietet.

Das EMF.Cloud Projekt bietet mit dem Coffee Editor ein Beispielprojekt an, das zeigt, wie eine webbasierte Entwicklungsumgebung auf Basis von Eclipse Theia und EMF.Cloud umgesetzt werden kann. Der Coffee Editor wird für die Implementierung als Grundlage verwendet.

Es kann allerdings nicht die aktuelle Version des Coffee Editor genutzt werden, da in der Version keine Änderungen über die Entwicklungsumgebung im Systemmodell vorgenommen werden können. Für die Implementierung wird auf die Version mit dem Commit-Hash *9a488d0* aufgesetzt. Abbildung 6.3 zeigt die Erweiterung des Coffee Editors. Während der Coffee Editor einige Funktionen bereits in einem geeigneten Maße bereitstellt, muss dieser anhand der gestellten Anforderungen erweitert werden. Die Implementierung der Entwicklungsumgebung ist in Abbildung 6.3 dargestellt.

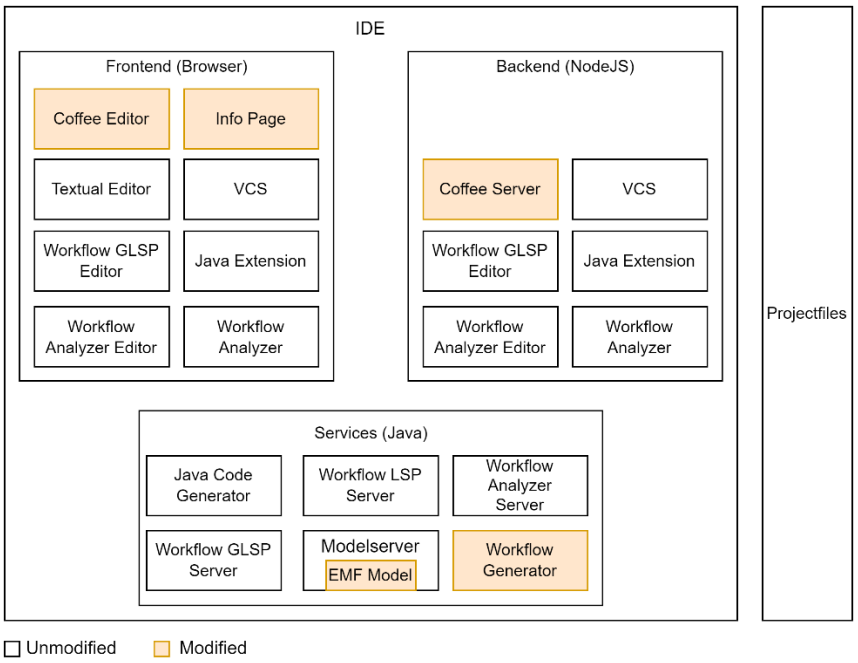


Abbildung 6.3: Implementierung Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus 3 Teilen. Das Frontend ist die Benutzeroberfläche für den Ingenieur, die im Webbrowser ausgeführt wird. Über das Backend wird das Frontend ausgeliefert sowie Funktionen bereitgestellt. Dabei bietet es die Funktionen entweder selbst an oder reicht Anfragen an in Java geschriebenen Services weiter. Die Services, die in Java implementiert sind, könnten auch außerhalb der IDE und innerhalb des Workspaces implementiert werden. Eclipse Theia basiert selbst aus Erweiterungen, die entweder Funktionen im Frontend und Backend oder einem von beiden hinzufügen.

Zu den nicht modifizierten Extensions gehört ein textueller Editor, in dem Programmcode oder Text verfasst werden kann. Zu den Extensions, die sowohl im Frontend als auch Backend Funktionen hinzufügen, gehört der Workflow GLSP Editor, der das grafische Modellieren von Abläufen ermöglicht. Für die grafische Modellierung können Regeln definiert werden (Workflow Analyzer Editor), die bei einer Analyse des Ablaufdiagramms beachtet werden sollen. Die Erweiterung Workflow Analyzer führt die eigentliche Analyse aus. Für die Generierung von Java-Code wird die Java Extension verwendet. Für die Integration mit einem VCS bietet der Coffee-Editor ebenfalls eine Erweiterung.

Auf der Frontendseite wurde die Coffee Editor Extension an das neue Datenmodell angepasst. Dazu zählen sowohl die Schemata für JSON Forms für die formularbasierten Editoren als auch die Erstellung des Theia-Tree-Editors. Eine weitere Extension, die angepasst wurde, ist die

Info Page Extension, die die Funktionen des Coffee Editor vorstellen. Diese Extension wurde durch die Masseberechnung und das Modelchecking erweitert.

Auf der Backendseite wurde die Coffee Server Extension erweitert. Es wurde eine Backend-Contribution hinzugefügt, die die Anfragen zur Masseberechnung und zum Modelchecking an den passenden Service außerhalb der IDE weiterleitet. Dadurch können auch Dienste im Cluster angesprochen werden, die von außen (z.B. vom Webbrowser direkt) nicht zugänglich sind.

Auf der Seite der Services musste der Java-Code- sowie Workflow-Generator angepasst werden, damit dieser mit dem neuen Datenmodell zurechtkommt. Dasselbe gilt für den Modelserver bzw. für das Datenmodell, was über den Modelserver zur Verfügung gestellt wird. Abgesehen von den bereits installierten Extensions können auch weitere Extensions zur Kompilierzeit sowie Plugins zur Laufzeit installiert werden.

Das für die Modellierung zugrunde liegende Datenmodell musste ebenfalls angepasst werden. Das Datenmodell vom Coffee Editor besitzt die Möglichkeit, eine Kaffeemaschine strukturell zu modellieren und mit vordefinierten Elementen zu befüllen. Es besitzt aber eine vorgegebene Struktur und kann nicht wie das Datenmodell von Virtual Satellite mit Konzepten erweitert werden. Anstatt wie bei Virtual Satellite mit dem Type-Of sowie Dynamic Template Pattern die Konzepte zu implementieren, wird zur Vereinfachung ein Vererbungs-Ansatz gewählt. Dabei wird wie bei VSD und Virtual Satellite das Prinzip von Engineering Categories umgesetzt. Die Implementierung des Datenmodells ist in Abbildung 6.4 dargestellt.

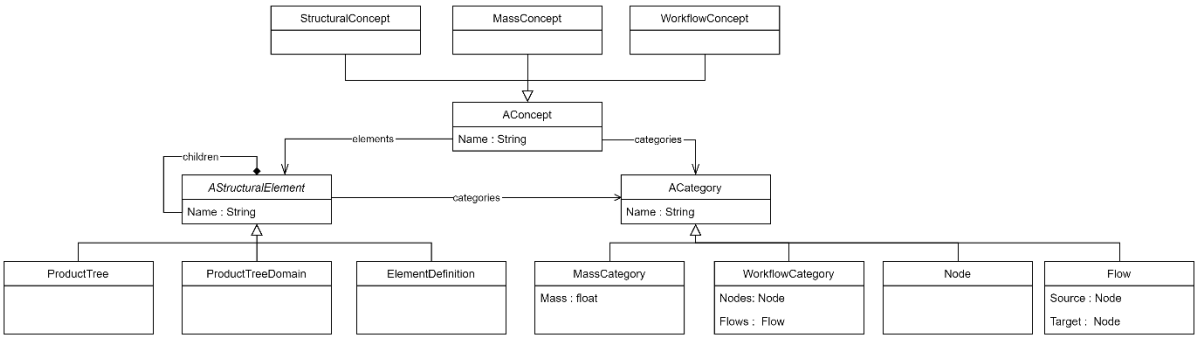


Abbildung 6.4: Implementierung Datenmodell

Beispielhaft wurden drei Konzepte definiert. Ein Konzept zur Beschreibung von Strukturen (StructuralConcept), ein Konzept zur Beschreibung von Masse (MassConcept) und ein Konzept zur Beschreibung von Abläufen (WorkflowConcept). Diese erben von einem abstrakten Konzept (AConcept), das definiert, dass ein Konzept StructuralElements und Categories beinhaltet. Ein StructuralElement kann dabei Kinderelemente besitzen, die ebenfalls

StructuralElements sind. Des Weiteren kann ein StructuralElement mehrere Categories beinhalten, die beschreibende Daten enthalten. Beispielhaft wurden als StructuralElement die Klassen ProductTree, ProductTreeDomain und ElementDefinition definiert, mit der ein hierarschich gegliedertes Systemmodell erstellt werden kann. Als beispielhafte Category wurde einmal die MassCategory definiert, die als Eigenschaft Masse beinhaltet. Für das WorkflowConcept wurden eine WorkflowCategory definiert, die einen Ablauf mithilfe von Nodes und Flows beschreibt.

Die Konzepte können wie bei Virtual Satellite zur Laufzeit aktiviert werden, wodurch die Modellierungssprache bei Bedarf angepasst werden kann. Durch die Modellierung der Masse können Berechnungen durchgeführt und durch das Modellieren von Abläufen können Analysen erstellt werden. Des Weiteren ermöglicht das Hinzufügen einzelner Categories, verschiedene Sichten auf das Systemmodell zu erstellen. Eine ontologische Darstellung der Schichten des Datenmodells ist in Abbildung 6.5 dargestellt.

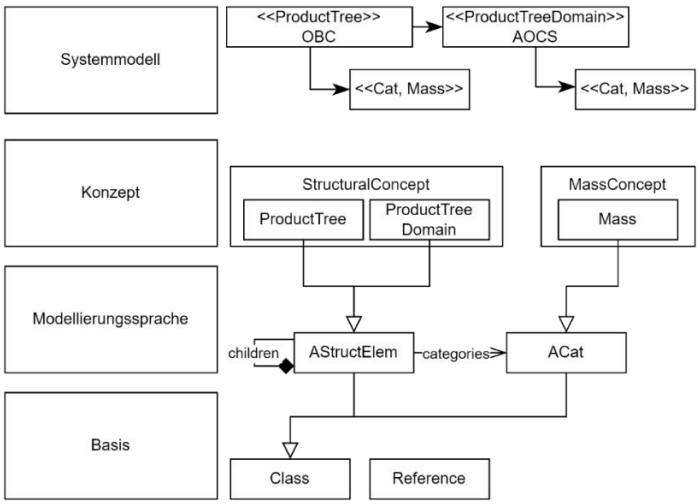


Abbildung 6.5: Implementierung Schichten des Datenmodells

Als Persistenzschicht wurde Ecore verwendet, das eine Basisklasse (EClass) und Basisreferenz (EReference) zur Verfügung stellt. In der Schicht der Modellierungssprache erben die abstrakten StructuralElements und Categories von der der EClass und die Referenzen children und categories von der EReference. In der Schicht der Konzepte werden dann das StructuralConcept und das MassConcept definiert, in denen die konkreten Klassen definiert werden. Im StructuralConcept sind das der ProductTree und die ProductTreeDomain und im MassConcept die Mass. Diese Klassen können im Systemmodell dann instanziiert werden. In dem obigen Beispiel sieht man einen Onboard-Computer (OBC) als ProductTree definiert, der als Category eine Masse besitzt. Als Kindelement besitzt es eine Attitude Determination and Control

System (AOCS) als ProductTreeDomain, die ebenfalls eine Instanz der Category Mass besitzt. Die implementierte Entwicklungsumgebung ist in Abbildung 6.6 dargestellt.

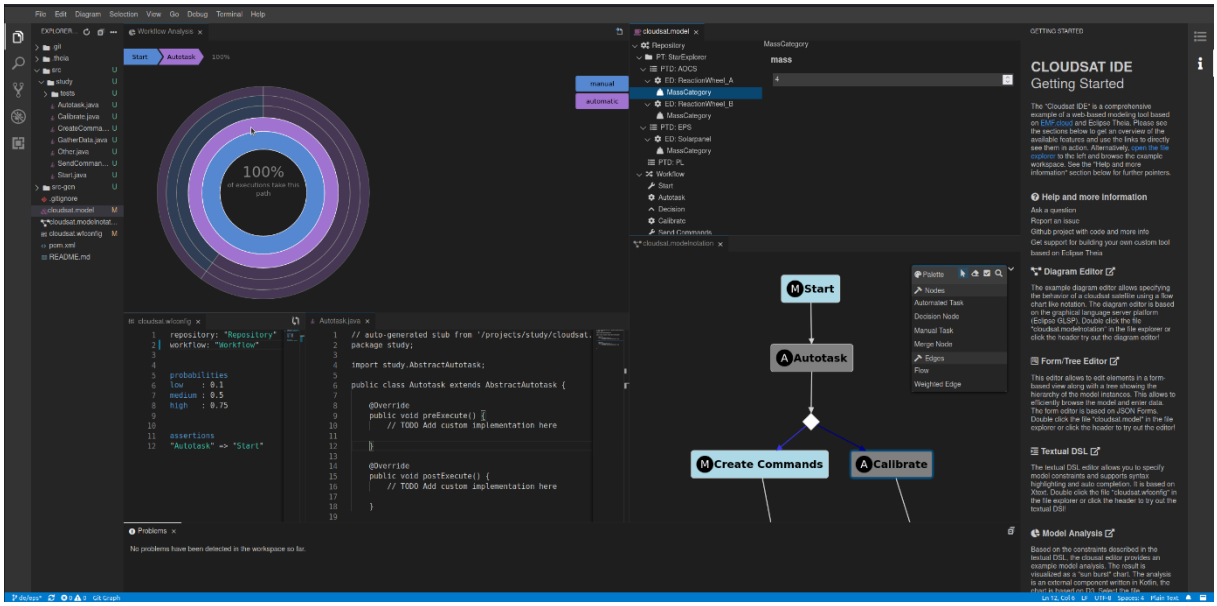


Abbildung 6.6: Implementierung Entwicklungsumgebung

6.4 Concurrent Engineering

Zur Implementierung wird das GitLab des DLR als VCS verwendet. Die Daten werden damit über das DLR GitLab gesichert und synchronisiert. Die daraus resultierende Architektur ist in Abbildung 6.7 dargestellt.

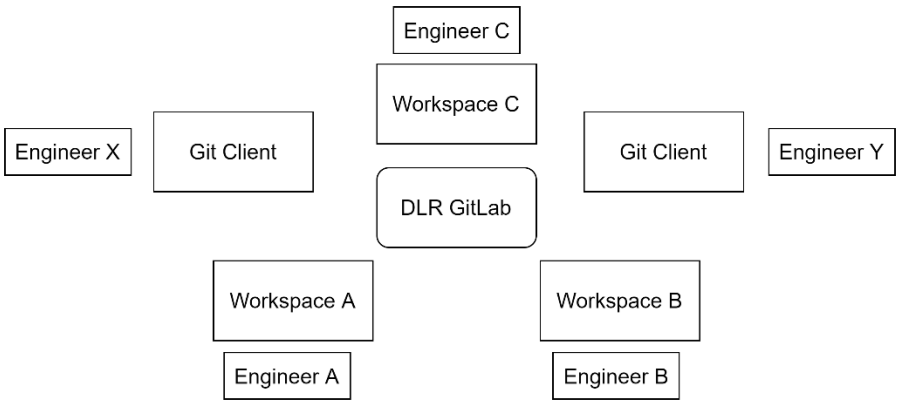


Abbildung 6.7: Implementierung Concurrent Engineering über eigene Workspaces

Durch die Verwendung des DLR GitLab können nicht nur Ingenieure über ihre Workspaces auf das GitLab zugreifen, sondern auch Ingenieure mit einem Git Client. Das GitLab des DLR ist nur mit einem gültigen Account einsehbar. Das GitLab ist in verschiedene Gruppen

aufgeteilt, die wiederum Sub-Gruppen enthalten können. Im Rahmen der Arbeit wird ein Repository in einer Sub-Gruppe (Template Repository) und ein privates Repository (Study Repository) verwendet. Dadurch, dass das GitLab vom DLR nur mit einem Account einsehbar ist, muss Eclipse Che im DLR GitLab als autorisierte Anwendung hinzugefügt werden. Dafür müssen eine Anwendungs-ID und ein Geheimnis erstellt werden. Diese beiden Informationen müssen dann als Base64-String als Secret im Cluster im Eclipse Namespace hinterlegt werden, damit sich das Cluster gegenüber dem DLR GitLab authentifizieren kann.

Durch die Nutzung der GitLab-Instanz kann das Systemmodell (Anforderung *fC14)* heruntergeladen werden und auch außerhalb des Clusters verwendet werden, um mit einer lokalen Arbeitskopie ohne Internetverbindung arbeiten zu können. Für die Arbeit im Cluster verfügt die Entwicklungsumgebung über einen Git Client, über den eine lokale Arbeitskopie des Systemmodells heruntergeladen werden kann.

Das Abzweigen von einem gemeinsamen Systemmodell sowie das Wiederverwenden eines Systemmodells wird im Folgenden dargestellt. Es wurde das Repository „Template“ (upstream) geforkt, um das Repository „Study“ (origin) zu erstellen. Als Startpunkt der Betrachtung ist das Repository in Abbildung 6.8 dargestellt.

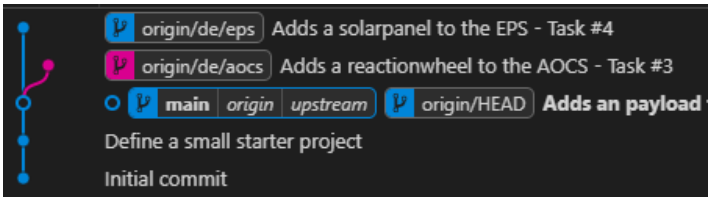


Abbildung 6.8: Concurrent Engineering Ausgangsrepository

Nach dem Fork wurden auf dem Study-Repository zwei Branches erstellt, die zwei Ingenieure darstellen sollen. Beide nutzen das Issue-System von GitLab, was an den Commit-Nachrichten ersichtlich wird. Ein beispielhaftes Issue ist in Abbildung 6.9 dargestellt.

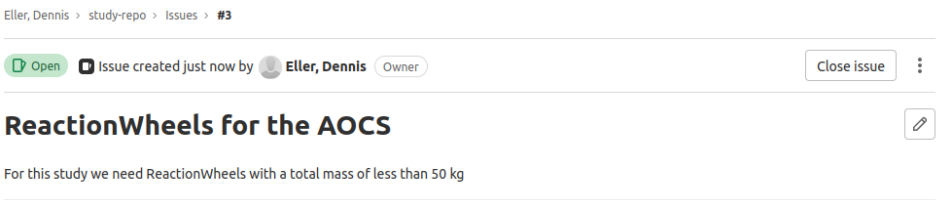


Abbildung 6.9: GitLab Taskmanagement

Zum Abarbeiten des Issue können die Ingenieure einen Merge-Request erstellen, um mit den anderen Ingenieuren in Kontakt zu treten. Ein beispielhafter Merge-Request ist in Abbildung 6.10 dargestellt.

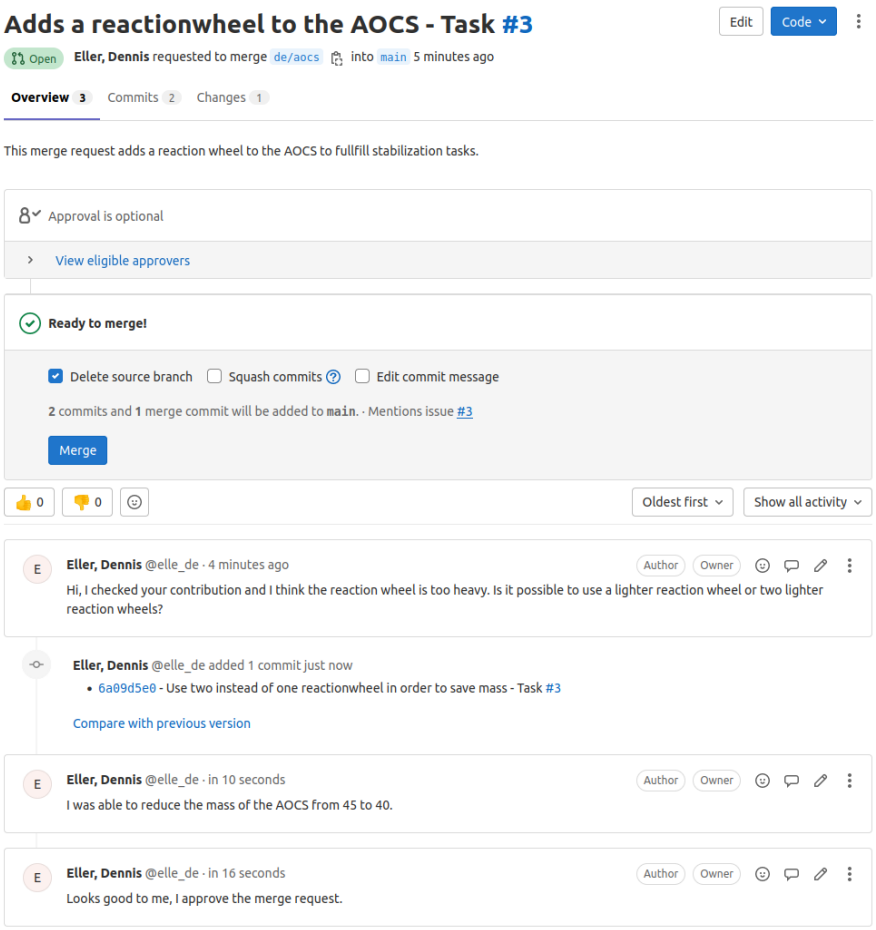


Abbildung 6.10: GitLab Merge-Request

Im Gespräch ist das Thema aufgefallen, dass das Gewicht des AOCS zu hoch ist. Auf Grundlage der Diskussion wird ein weiterer Commit erstellt, der die Gesamtmasse des Systems verringert. Der neue Commit sorgt dafür, dass der Merge-Request angenommen wird und in den main übernommen wird. Während die Ingenieure in ihren Branches arbeiten, wird das Template Repository aktualisiert. Das aktualisierte Repository ist in Abbildung 6.11 dargestellt.

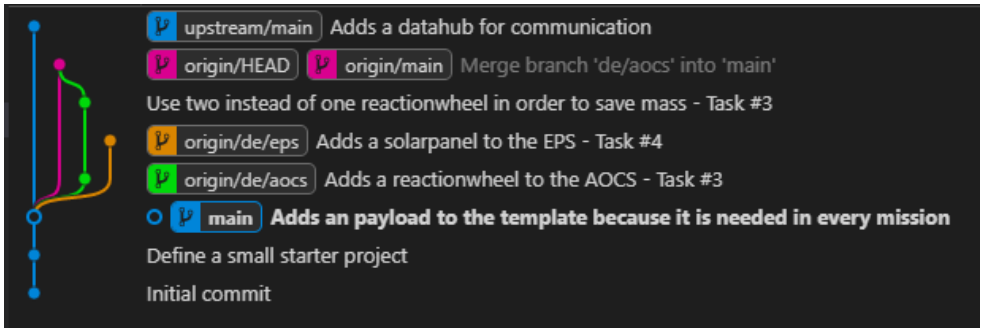


Abbildung 6.11: Concurrent Engineering Feature Branch gemergt und Template-Repository aktuallisiert

Die Änderungen des Branches über das AOCS (de/aocs) wurden durch den Merge-Request in den main übernommen. Im Template Repository wurde ein neuer Commit veröffentlicht, der die Vorlage um ein Beispiel für ein Datenverarbeitungssystem (Datahub) ergänzt. Im nächsten Schritt werden die Änderungen des Template Repository in den main über einen Merge übernommen. Der resultierende Merge ist in Abbildung 6.12 dargestellt.

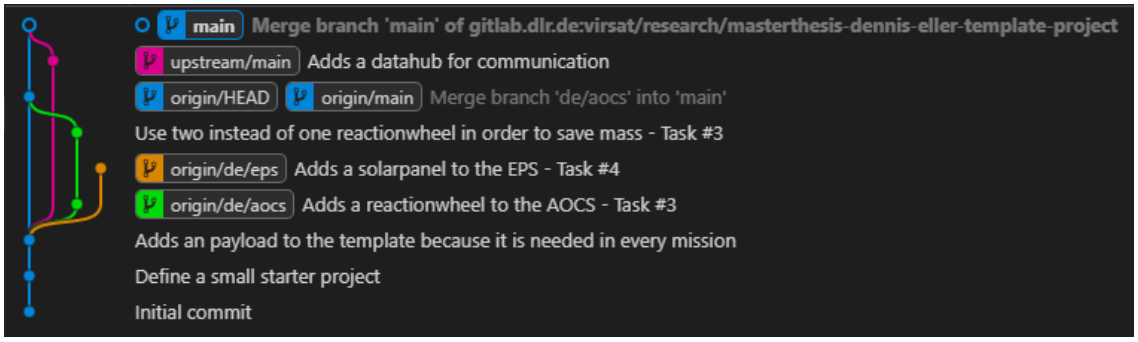


Abbildung 6.12 Rebase auf Template-Repository

Nach dem Merge beinhaltet der main den Commit aus dem Template Repository, in dem eine Vorlage für ein Datahub hinzugefügt wurde. Der Merge-Request mit dem AOCS kann gelöscht werden, da dieser schon in den main übernommen wurde.

Der Merge von dem Merge-Request mit dem AOCS ist weiterhin der aktuellste Commit. Nun können der AOCS-Branch gelöscht und der Hauptbranch vom Study-Repository auf den lokalen Stand gebracht werden. Danach wird noch ein Rebase mit dem Branch für die

Energieversorgung (de/eps) auf den lokalen main durchgeführt. Das finale Study- Repository ist in Abbildung 6.13 dargestellt.

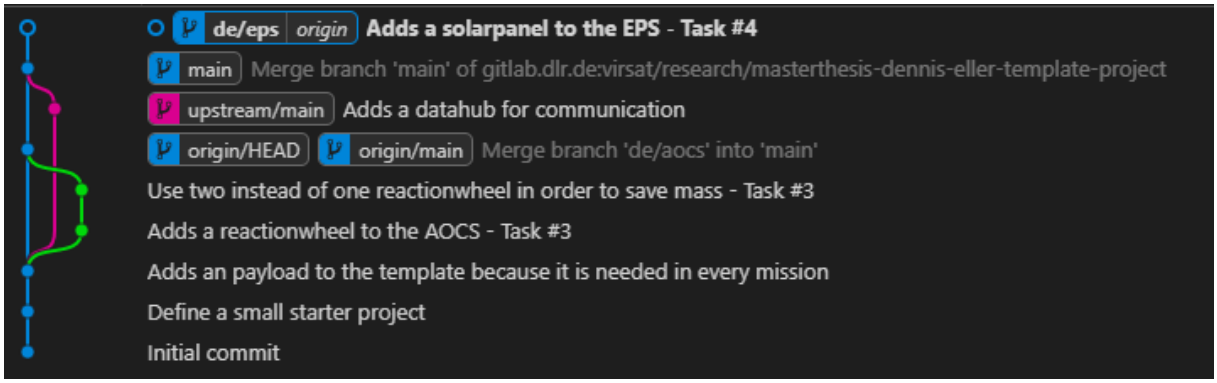


Abbildung 6.13: Finales Study-Repository

Da der main von verschiedenen Ingenieuren verwendet wird, darf auf das Template Repository kein Rebase ausgeführt werden, weil sich sonst die Historie bzw. die Commit-Hashes verändern. Ein Rebase ist nur eine Alternative, wenn in dem Repository alleine gearbeitet wird.

7 Evaluierung

Zur Evaluierung einer Softwarearchitektur gibt es unterschiedliche Methoden [104]:

- Szenariobasierte Software Architektur Evaluierungsmethoden
- Mathematisch-modelbasierte Softwarearchitektur Evaluierung
- Software-architekturbasierende Zuverlässigkeitsanalysen
- Software-architekturbasierende Performanzanalysen

Die Methoden beinhalten ein iteratives Vorgehen mit mehreren Stakeholdern, das den Rahmen dieser Arbeit überschreiten würde. Deswegen konzentriert sich die Evaluierung auf die Beantwortung der zuvor formulierten Forschungsfragen. Dafür wird zunächst eine Metadokumentation über die Evaluierung dargestellt. Danach werden die Ergebnisse der Evaluierung vorgestellt. Im Anschluss werden die Ergebnisse diskutiert.

7.1 Metadokumentation

Zur Evaluierung wird ein Lenovo ThinkPad P15v Gen 2i mit Intel Core i9-11950H (2.60 GHz) mit 64 GB Arbeitsspeicher verwendet. Als Betriebssystem kommt Windows 10 Enterprise zum Einsatz. Auf dem Laptop wird die in Kapitel 6 implementierte Lösung deployt. Der virtuellen Maschine mit Ubuntu 20.04 LTS werden 8 Prozessoren und 50 GB Arbeitsspeicher zur Verfügung gestellt. In der virtuellen Maschine wird Minikube mit 8 Prozessoren und 32 GB Arbeitsspeicher betrieben.

7.2 Ergebnisse

Im folgenden Kapitel werden die Ergebnisse der Evaluierung vorgestellt. Dabei teilen sich die Ergebnisse auf die drei Forschungsfragen auf. Zur Evaluierung der ersten Forschungsfrage werden die zuvor identifizierten Anforderungen dargestellt. Bei der Evaluierung der zweiten Forschungsfrage wird speziell auf einige Aspekte konkreter eingegangen. Bei der Evaluierung der dritten Forschungsfrage werden generierte Datensätze zum Messen der Performanz verwendet. Wie bereits in Kapitel 6.1 erwähnt, war das Deployment in einer Cloud aufgrund von Einschränkungen seitens des DLR nicht möglich. Daher erfolgt die Evaluierung über den oben genannten Laptop.

7.2.1 Anforderungen von Concurrent Engineering und MBSE

Die erste Forschungsfrage beschäftigt sich damit, welche Anforderungen das Concurrent Engineering mit dem modellbasierten System Engineering für den Betrieb in einer Cloud stellen. Die zusammengefassten Anforderungen wurden bereits in Kapitel 5.1.2 vorgestellt. In Abbildung 7.1 ist die Verteilung Kano-Kategorisierungen der Anforderungen dargestellt.

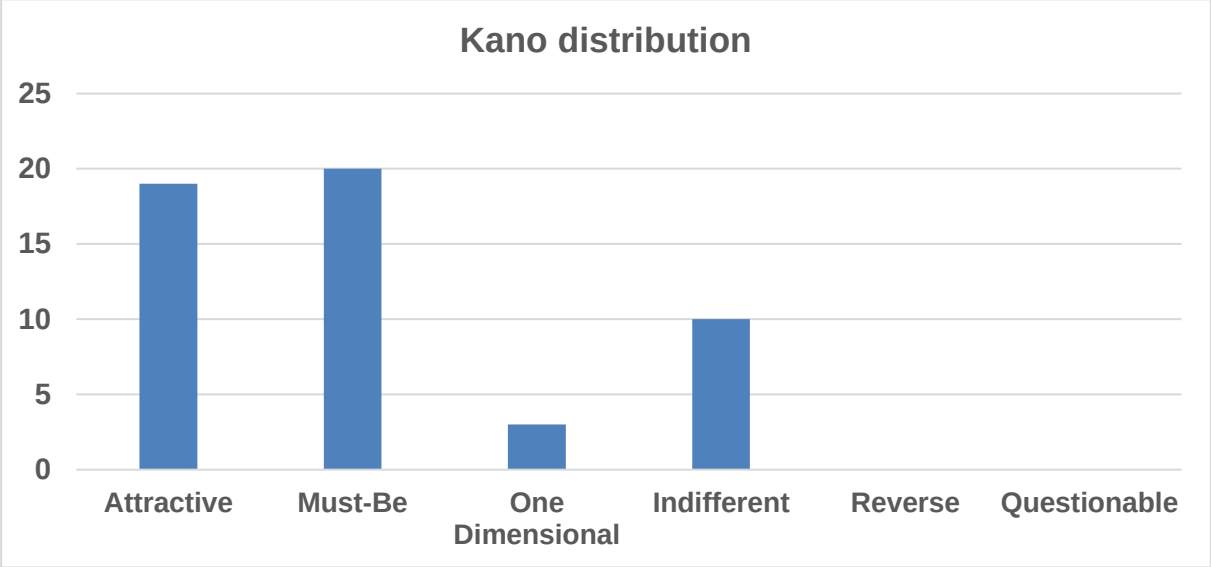


Abbildung 7.1: Verteilung Kano-Kategorisierung der Anforderungen

Insgesamt wurden 52 Anforderungen identifiziert, wovon 19 als Begeisterungs-, 20 als Basis-, 3 als Leistungs- und 10 als unerhebliches Merkmal identifiziert wurden. Keine Anforderung wurde als Rückweisungsmerkmal oder fragwürdig kategorisiert.

7.2.2 Prozessunterstützung für Raumfahrtsysteme

Die zweite Forschungsfrage beschäftigt sich damit, wie gut aktuell verfügbare Tools die Prozesse für Raumfahrtsysteme bei Produktlinien, Versionierung, Synchronisierung und Privacy sowie das Abbilden von verschiedenen Sichten bzw. Abstraktionsebenen auf das Systemmodell in der Cloud unterstützen. Im Folgenden wird auf die einzelnen Aspekte eingegangen.

Produktlinien, Versionierung, Synchronisierung

Über die Verwendung eines VCS, wie es heutzutage Standard bei Quellcode ist, können Änderungen mit anderen Ingenieuren synchronisiert werden. Ebenso ermöglicht ein VCS das Versionieren der Änderungen, wodurch auch ältere Stände gesichert werden. Über das Forken

von Repositories oder das Branchen innerhalb eines Repositories können außerdem Produktlinien erstellt werden, die ebenso über das VCS versioniert und synchronisiert werden können.

Privacy

Wenn Produkte oder Systeme entwickelt werden, sind diese Daten besonders zu schützen. Die Kommunikation mit dem Cluster erfolgt ausschließlich über HTTPS. Insbesondere wenn Daten in einer Cloud gehalten werden, liegt die Datenhoheit bei einer dritten Partei. Im Rahmen der Arbeit konnte keine Möglichkeit gefunden werden mit verschlüsselten Systemmodelldateien Concurrent Engineering über GitLab zu betreiben.

Abbildung von verschiedenen Sichten auf das Systemmodell

Durch die Verwendung von EMF.Cloud mit der Verbindung mit Engineering Categories ist es möglich, verschiedene Editoren mit verschiedenen Sichten auf das Systemmodell zu erstellen. Des Weiteren kann durch EMF kann ein Metamodell definiert werden, was die Erweiterung des Datenmodells erlaubt. Durch Darstellung können verschiedene Editoren genutzt werden. Der Tree Editor kann zum Beispiel für eine strukturelle Darstellung genutzt werden, während ein Ablaufdiagramm für die Verhaltensmodellierung genutzt werden kann. Darauf aufbauend können Sichten für Analysen erstellt werden, die dann z.B. durch eine DSL konfiguriert werden können. In späteren Lebensphasen wird auch die Generierung von Quellcode relevant, was auch eine weitere Sicht auf das Systemmodell darstellt.

Abbildung von verschiedenen Abstraktionsebenen auf das Systemmodell

EMF.Cloud macht EMF für Web- und Cloudtechnologien zugänglich. Dadurch kann es (in der Theorie) alles abdecken, was auch von EMF bereitgestellt wird. Wie im Stand der Technik (Kapitel 4.2.3) bereits vorgestellt und in der Implementierung (Kapitel 6.3) umgesetzt, kann EMF bzw. EMF.Cloud genutzt werden, um verschiedene Abstraktionsebenen auf das Systemmodell zu definieren. Auf der Serverseite können alle Funktionen von EMF genutzt werden, jedoch stehen diese nicht alle auf der Clientseite zur Verfügung.

7.2.3 Skalierungen von Simulationen und Berechnungen

In der dritten Forschungsfrage soll untersucht werden, wie Simulationen und Berechnungen im MBSE-Kontext in einer Cloud skalieren. Zur Evaluierung wird ein Internal Service verwendet, der auf Minikube deployt wird. Als Client dient dabei die Entwicklungsumgebung, die inklusive IDE-Backend und Modelserver in einem Workspace läuft. Es wird ein Internal Service gewählt, da sich dieser möglichst weit von der Anfragenquelle (IDE-Frontend) befindet und

damit einen möglichst weiten Kommunikationsweg im Cluster besitzt. Der Internal Service befindet sich damit noch im Cluster und im Gegensatz zu externen Services noch im Einflussbereich des Clusters. Damit kann evaluiert werden, welchen Einfluss das Einbringen der vorgeschlagenen Architektur auf das Laufzeitverhalten hat. Der Versuchsaufbau ist in Abbildung 7.2 dargestellt.

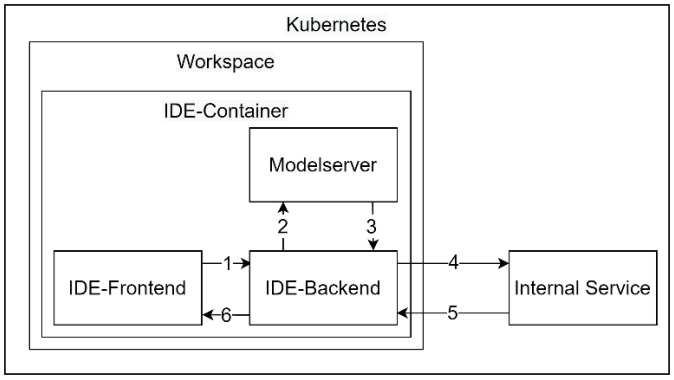


Abbildung 7.2: Versuchsaufbau dritte Forschungsfrage

Die Anfrage wird vom IDE-Frontend an das IDE-Backend gesendet. Das IDE-Backend bezieht das Systemmodell als JSON von dem Modelserver. Danach sendet das IDE-Backend die Anfrage an den Internal Service. Nach der Bearbeitung der Anfrage wird die Antwort an das IDE-Backend zurückgesendet, das die Antwort an das IDE-Frontend weiterleitet. Nach dem Erhalt der Antwort wird die nächste Anfrage versendet. Die Zeit von dem Absenden der Anfrage bis zum Erhalt der Antwort wird für die Evaluierung gemessen.

In einem zweiten Versuch werden mehrere Anfragen gleichzeitig versandt. Dabei wird die Zeit vom Abschicken der Anfragen bis zum Ankommen aller Antworten gemessen. Die gemessenen Zeiten werden dabei mit einer Version des Internal Services verglichen, der lokal ausgeführt wird und keine Netzwerkkommunikation durchführt. Für jeden Datenpunkt wird die Messung 1000 Mal wiederholt. Eine Mittelwertbetrachtung mit Standardabweichung ist nicht möglich, da der Betrag der Standardabweichung durch das Vorhandensein von Ausreißern größer als der Mittelwert ist. Dadurch würden die unteren Fehlerbalken in den negativen Bereich fallen, was für eine Betrachtung der Laufzeit keine verwertbaren Ergebnisse liefert. Deswegen wir der Median mit dem 99% Perzentil als Fehlerbalken aufgetragen.

Im ersten Teil wird die Berechnung der Gesamtmasse des Systemmodells evaluiert, während im zweiten Teil die Performanz eines Modelcheckers beim Überprüfen auf Deadlocks betrachtet wird. Für die Berechnung der Gesamtmasse wird ein selbst geschriebenes Programm verwendet, während als Modelchecker Storm in der Travis Version verwendet wird. Es wurden

diese zwei Arten von Services ausgewählt, da die Berechnung der Gesamtmasse im Vergleich zum Modelchecker in kurzer Zeit durchgeführt wird. Dadurch kann untersucht werden, welchen Einfluss der Rechenaufwand auf das Laufzeitverhalten besitzt. Des Weiteren unterscheiden sich die zwei Services darin, dass die Berechnung der Gesamtmasse auch in der IDE implementiert ist, was bei dem Modelchecker nicht der Fall ist.

Berechnung der Gesamtmasse

Zur Evaluierung wird zunächst das Zeitverhalten für die Berechnung der Gesamtmasse des Systemmodells gemessen. Dafür wird ein Systemmodell mit dem Structural- und MassConcept mit Masse modelliert. Es werden Systemmodelle generiert, die jeweils 1, 10, 100, 1.000 und 10000 MassCategories enthalten. Die Dauer für die Berechnung der Gesamtmasse des Systems in Abbildung 7.3 dargestellt.

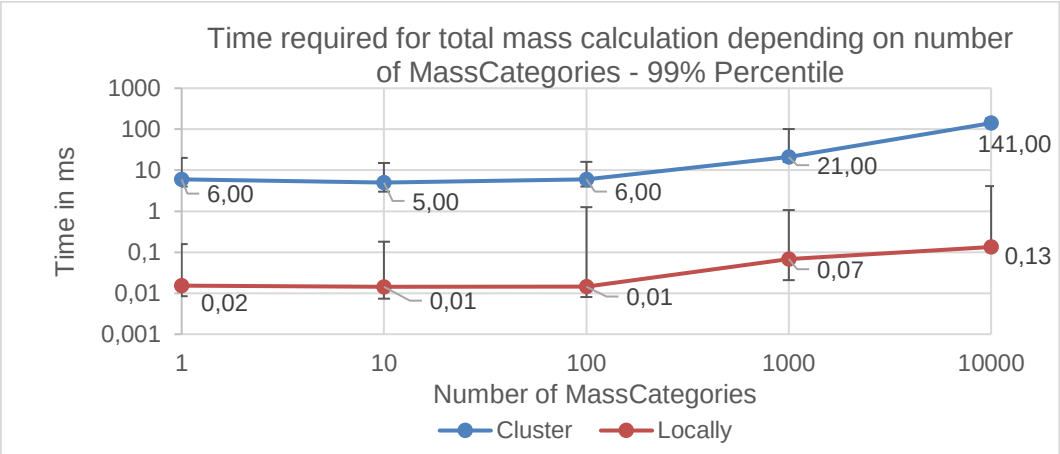


Abbildung 7.3: Zeit zur Berechnung der Gesamtmasse (sequenziell)

Bei einer Ausführung im Cluster liegt der Median der benötigten Zeit der Gesamtmassenberechnung pro Aufruf zwischen 5 und 6 ms bis 100 MassCategories. Bei 1000 MassCategories wächst die benötigte Zeit auf 21 ms an und erreicht bei 10000 MassCategories eine Dauer von 141 ms. Bei der lokalen Ausführung liegt der Median zwischen 0,01 und 0,02 ms. Bei 1000 MassCategories steigt der Median auf 0,07 ms und erreicht bei 10000 MassCateogories eine Dauer von 0,13ms. Die Mediane der lokalen Ausführung sind um den Faktor 300 bis 1084 kleiner als die der Ausführung über das Cluster.

Im nächsten Schritt wird das Zeitverhalten der Berechnung der Gesamtmasse betrachtet, wenn mehrere Anfragen gleichzeitig an einen Internal Service gestellt werden. Bei jeder Anfrage wird ein generiertes Systemmodell mit 1000 MassCategories verwendet. Die Anzahl

der gleichzeitigen Anfragen beträgt jeweils 1, 10, 100 und 1000. Die Dauer für die Berechnung der Gesamtmasse des Systems mit 1000 MassCategories in Abbildung 7.4 dargestellt.

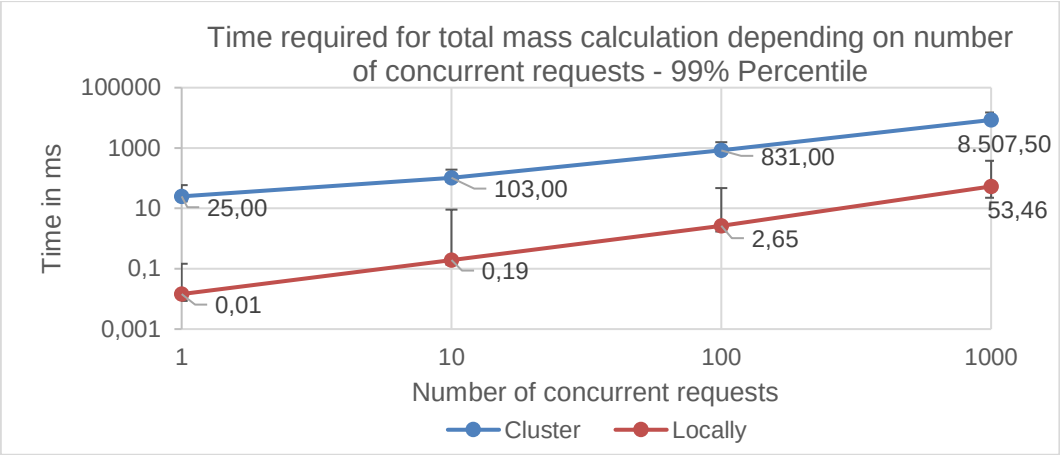


Abbildung 7.4: Zeit zur Berechnung der Gesamtmasse (sequenziell)

Bei einer Ausführung im Cluster steigt die benötigte Zeit der Gesamtmassenberechnung in Abhängigkeit der gleichzeitigen Aufrufe an. Auf der anderen Seite steigt bei der lokalen Ausführung die benötigte Zeit ebenfalls mit der Anzahl der gleichzeitigen Aufrufe an. Die Mediane der lokalen Ausführung sind um den Faktor 159 bis 2500 kleiner als die der Ausführung über das Cluster.

Überprüfung auf Deadlocks über einen Modelchecker

Zur Evaluierung des Modelcheckers wird die Zeit gemessen, die die Überprüfung auf einen Deadlock benötigt. Dafür werden Systemmodelle mit Ablaufdiagrammen (Workflows) erstellt, die jeweils 1, 2, 3, 4, 5, 6 und 7 Workflows mit jeweils 10 Zuständen besitzen. Die Zeit für die Deadlock-Überprüfung ist in Abbildung 7.5 dargestellt.

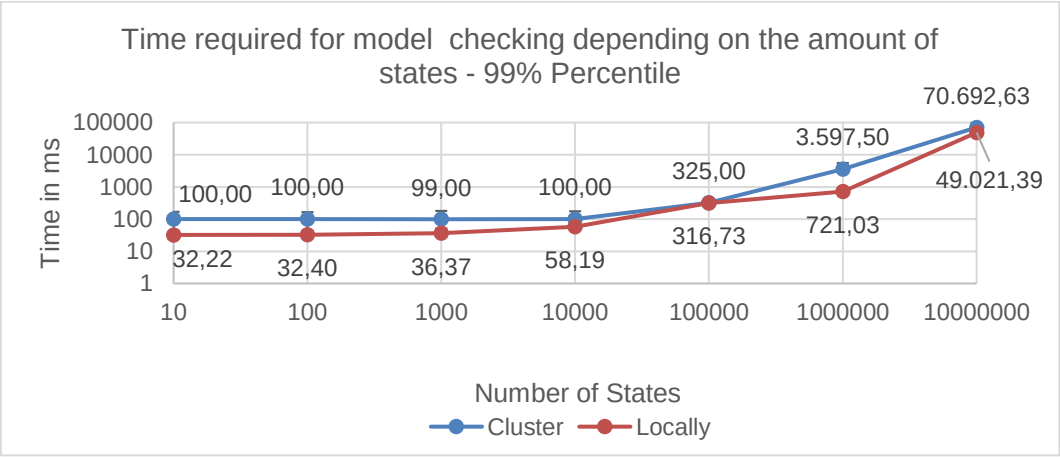


Abbildung 7.5: Zeit für die Deadlock-Überprüfung eines Modelcheckers

Bis zu einer Workflow-Anzahl von 4 Workflows (10000 States) bleibt die benötigte Zeit bei einer Ausführung im Cluster bei ca. 100 ms. Danach steigt die benötigte Zeit mit der Anzahl der Workflows an bis sie bei 7 Workflows auf 70692,63 ms erreicht. Bei der lokalen Ausführung bleibt die benötigte Zeit bis zu einer Workflow-Anzahl von 3 Workflows (1000 States) relativ konstant. Danach steigt die benötigte Zeit bis 7 Workflows auf 49021,39 ms an. Die Mediane der lokalen Ausführung sind um den Faktor 1,03 bis 4,99 schneller als die der Ausführung über das Cluster.

Im nächsten Schritt wird das Zeitverhalten gemessen, wenn mehrere Anfragen gleichzeitig bearbeitet werden. Die Anzahl der gleichzeitigen Anfragen beträgt jeweils 1, 10, 100 und 1000. Bei jeder Anfrage wird ein generiertes Systemmodell mit 4 Workflows (10000 States) verwendet. Die benötigte Zeit für die Deadlock-Überprüfung bei mehreren Anfragen gleichzeitig ist in Abbildung 7.6 dargestellt.

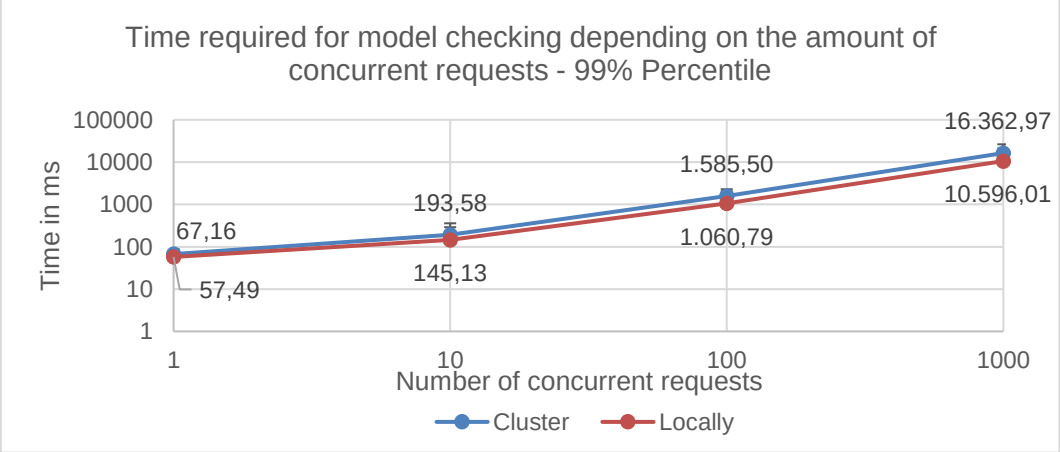


Abbildung 7.6: Zeit für die Deadlock-Überprüfung eines Modelcheckers bei gleichzeitigen Aufrufen

Sowohl bei der Ausführung im Cluster als auch lokal wächst die benötigte Zeit kontinuierlich mit der Anzahl der gleichzeitigen Anfragen. Die Mediane der lokalen Ausführung sind um den Faktor 1,16 bis 1,54 schneller als die der Ausführung über das Cluster.

7.3 Diskussion

Im Folgenden werden die Ergebnisse der Evaluierung der Forschungsfragen diskutiert. Dafür wird für die erste Forschungsfrage geprüft, welche Anforderungen in welchem Maß erfüllt werden konnten. Bei der zweiten Forschungsfrage wird speziell auf einige Aspekte konkreter eingegangen. Bei der Diskussion der dritten Forschungsfrage werden die Ergebnisse der Performanzmessungen untersucht.

7.3.1 Anforderungen von Concurrent Engineering und MBSE

Im Folgendem sind die zusammengefassten Anforderungen mit einer Bewertung der Erfüllung dargestellt. Für die Bewertung der Erfüllung werden die Akzeptanzkriterien der Anforderungen verwendet. Grün steht dabei für voll erfüllt, gelb für teilweise erfüllt und rot für nicht erfüllt.

Funktionale Anforderungen

Modellierung

Nr.	Anforderung:	Kano
fM1	Mit der Software kann ein System modelliert werden.	M
fM2	Mit der Software kann an einem eigenen Subsystem an der eigenen Domäne gearbeitet werden.	M
fM3	Die Software bietet unterschiedliche Sichten auf die Daten an und ermöglicht das Definieren von eigenen Sichten.	M
fM4	Mit der Software kann die Modellierungssprache erweitert werden.	A
fM5	Mit der Software können Berechnungen und Referenzen definieren werden.	A
fM6	Mit der Software können Anforderungen definiert und mit Elementen vom Systemmodell verknüpft werden.	M
fM7	Mit der Software können Simulationen durchgeführt werden.	A
fM8	Mit der Software kann das Systemmodell auf Konsistenz geprüft werden.	M
fM9	Mit der Software kann eine Impact Analysis durchgeführt werden.	A
fM10	Mit der Software können Daten in weitere Lebensphasen übernommen werden.	O

Tabelle 7.1: Funktionale Anforderungen: Modellierung

Die Anforderung *fM1* wird durch die definierte Modellierungssprache erfüllt. Es ermöglicht das Modellieren einer Hierarchie des Systems über Structural Elements und das Hinzufügen von Eigenschaften über Categories, wodurch zusammengehörige Informationen zusammen gespeichert und verknüpft werden können. Dadurch, dass die Modellierungssprache in Konzepte aufgeteilt ist, kann der Modellierungsprozess mit einer einfachen Modellierungssprache begonnen werden. Durch das Modellieren über Structural Elements können außerdem für jede Domäne eigene Subsysteme definiert werden (Anforderung *fM2*). Des Weiteren erfüllt die Lösung die Anforderung *fM3* teilweise, da es verschiedene Sichten bzw. Editoren für das Systemmodell zur Verfügung stellt, die sich gegenseitig synchronisieren. Die verschiedenen Sichten wurden bereits in Abbildung 6.6 dargestellt. Allerdings ermöglicht die Lösung nicht das Definieren von neuen Sichten. Durch die Aufteilung der Modellierungssprache in Konzepte ist es außerdem möglich, die Modellierungssprache erweitern zu können, um je nach Projekt die benötigten Konzepte zu verwenden (*fM4* Anforderung). Die Anforderung *fM5* wird teilweise erfüllt, da Berechnungen wie die Kalkulation der Gesamtmasse zwar möglich sind, aber nicht direkt im Systemmodell integriert sind.

Des Weiteren können Referenzen nur für das Ablaufdiagramm definiert werden, da der verwendete Tree-Editoren Referenzen nicht unterstützt. Aus demselben Grund können mit der Lösung keine Anforderungen definiert werden (Anforderung *fM6*). Es könnten zwar generelle Anforderungen über ein Konzept definiert werden, allerdings nicht mit Komponenten des Systemmodell verknüpft werden. Das Durchführen von Simulationen (Anforderung *fM7*) wurde beispielhaft anhand des Modelcheckers Storm implementiert. Über den Modelchecker kann überprüft werden, ob die Auflaufdiagramme des Systems in einen Deadlock gelangen können.Anforderungen *fM8* und *fM9* konnten aus Zeitgründen nicht implementiert werden. Allerdings würde die aktuelle Version des Coffee Editors eine Konsistenzprüfung sowie die Darstellung von Auswirkungen bei Änderungen unterstützen. Dafür müssten die zuvor dargestellten Probleme behoben werden. Die Anforderung *fM10* wurde nicht erfüllt, da der Aufwand der Implementierung außerhalb des Rahmens der Arbeit gewesen wäre.

Concurrent Engineering

Nr.	Anforderung	Kano
fC1	Mit der Software können mehrere Ingenieure gleichzeitig Zugriff auf das Systemmodell haben.	M
fC2	Mit der Software können Merge-Konflikte aufgelöst werden.	O
fC3	Mit der Software können Änderungen am Datenmodell in einer Kopie vorgenommen werden, ohne das gemeinsame Systemmodell zu überschreiben.	A
fC4	Mit der Software können Daten im Systemmodell als Favorit markiert werden.	I
fC5	Die Software informiert umgehend über aktuell eingecheckte Systemmodelländerungen und lässt mich entscheiden, diese direkt oder später zu übernehmen.	A
fC6	Die Software kann den aktuellen Status des Systemmodells darstellen, ohne auf alle Commits der anderen Ingenieure zu warten.	I
fC7	Mit der Software können Daten als sichtbar und unsichtbar deklariert werden.	I
fC8	Die Software kann darstellen, ob Daten aktuell von anderen Ingenieren bearbeitet werden oder ob diese eingeloggt sind.	A
fC9	Mit der Software können Kommentare an spezielle Ingenieure am Systemmodell hinterlegt werden.	A
fC10	Die Software kann darstellen, was sich seit der letzten Session geändert hat.	A
fC11	Mit der Software kann ein Workspace mit einem anderen Ingenieur geteilt werden.	A
fC12	Die Software bietet einen toolgestützten Change-Management-Prozess für jegliche Systemmodellelemente an.	A
fC13	Die Software bietet ein toolgestütztes Task-Management für jegliche Systemmodellelemente an.	A
fC14	Mit der Software kann das Systemmodell auch offline bearbeitet werden können.	M

Tabelle 7.2: Funktionale Anforderungen: Concurrent Engineering

Die Lösung ermöglicht den zeitgleichen Zugriff auf das Systemmodell (Anforderung *fC1*) über ein VCS (GitLab), wodurch jeder Ingenieur mit einer lokalen Arbeitskopie arbeitet. Bei dem Zusammenführen von Änderungen ermöglicht die Lösung über das Git Plugin auch das Auflösen von Merge-Konflikten (Anforderung *fC2*), allerdings nur auf Textbasis, was je nach Dateityp unübersichtlich sein kann. Durch den Einsatz von GitLab als VCS ist es möglich, in einer lokalen Kopie Änderungen vorzunehmen, ohne das gemeinsame Systemmodell zu ändern (Anforderung *fC3*). Daten im Systemmodell können nicht als Favorit markiert werden Anfordeurng (*fC4*), um über Änderungen informiert zu werden. Allerdings können im Rahmen des Change-Management-Prozesses Ingenieure markiert werden, um bei der Diskussion um Änderungen informiert zu werden und teilzunehmen. Durch die Verwendung von einem Workspace pro Ingenieur beim Concurrent Engineering wurde die Anforderung *fC5* nicht erfüllt, da

Ingenieure selbst entscheiden sollen, wann die Änderungen einfließen sollen. Anforderung *fC6* wurde nicht erfüllt, da sie als unerheblich deklariert wurde und es nicht trivial ist, alle ausstehenden Änderungen der Ingenieure zusammengeführt darzustellen. Das Ausblenden von Daten des Systemmodells (Anforderung fC7) wird vom Coffee Editor nicht unterstützt. Des Weiteren wurde die Anforderung als unerhebliches Merkmal klassifiziert, weswegen die Anforderung nicht implementiert wurde. Die Anforderung fC8 wurde nicht erfüllt, da Eclipse Che keine Informationen über die eingeloggten Benutzer ausgeben kann. Des Weiteren besitzt der Coffee Editor keine Anzeige, mit der die eingeloggten Benutzer angezeigt werden könnten.

Über die Verwendung von GitLab können im Rahmen vom Merge-Requests Kommentare an Elemente des Systemmodells angehängt werden (Anforderung *fC9*). Allerdings liegen diese nicht wie gefordert im Systemmodell, weswegen die Anforderung teilweise erfüllt wird. Durch den Einsatz von Git bzw. GitLab werden die Änderungen am Systemmodell protokolliert, wodurch bei der nächsten Sitzung nachvollzogen werden kann, welche Änderungen durchgeführt wurden. Allerdings arbeiten Git bzw. GitLab textbasiert, was das Vollnachziehen von Änderungen je nach Dateiart erschwert.

Die Anforderung *fC11* wurde teilweise erfüllt, da in dem Workspace sensible Daten des Workspacebesitzers wie SSH-Keys für Git gespeichert werden. Da dies ein Sicherheitsrisiko darstellt, deaktiviert Eclipse Che die Möglichkeit, Workspaces für andere Nutzer freizuschalten standardmäßig. Bei der Aktivierung der Funktion kann es außerdem zu Inkonsistenzen kommen, weil Eclipse Che einer „last write wins policy“ folgt, in der die letzte Bearbeitung die vorherige überschreibt. Allerdings können einzelne Container des Workspaces nach außen zugänglich gemacht werden, wodurch zumindest über einen Link zu einer API auf den Workspace zugegriffen werden kann. Durch die Verwendung von GitLab wird ein Change-Management-Prozess (Anforderung *fC12*) über Merge-Requests bereitgestellt, wodurch Änderungen mit den anderen Ingenieuren diskutiert werden können. GitLab bietet außerdem ein Task-Management (Anforderung *fC13*) über Issues an, worüber Aufgaben an einzelne Ingenieure verteilt werden können, die wiederum in Merge-Requests referenziert werden können.

Dadurch, dass bis auf das Frontend alles im Cluster läuft, benötigt der Ingenieur zumindest zum Cluster eine Internetverbindung. Ohne eine Netzwerkverbindung zum Cluster sind die Dienste des Clusters nicht verwendbar. Durch den Einsatz eines VCS ist es aber möglich, die Daten über einen VCS-Client lokal zu beziehen. Sobald wieder eine Internetverbindung besteht, können Änderungen mit dem VCS synchronisiert und die Dienste des Clusters erneut verwendet werden.

Schnittstellen & Export von Daten

Nr.	Anforderung	Kano
fS1	Die Software stellt das Systemmodell über eine Web-API zur Verfügung.	A
fS2	Die Software kann externe Software über deren Web-API ansteuern.	I
fS3	Die Software kann das Systemmodell exportieren.	M

Tabelle 7.3. Funktionale Anforderungen: Schnittstellen & Export von Daten

Der verwendete Modelserver von EMF.Cloud stellt das Systemmodell als REST-API zur Verfügung, wodurch es auch ohne die vorgegebene Benutzeroberfläche bearbeitet werden kann (Anforderung *fS1*). Die Lösung kann die Web-APIs von externer Software ansteuern, da sie auf Cloud- und Webtechnologien aufbaut. Die Anforderung *fS3* wird teilweise erfüllt, weil der Export vom Systemmodell nicht in gängigen Dateiformaten wie CSV geschieht, sondern so wie sie im VCS gespeichert werden würden.

Produktlinien & Versionierung

Nr.	Anforderung	Kano
fP1	Die Software bietet eine Versionsverwaltung für das Systemmodell an.	M
fP2	Die Software ermöglicht es, frühere Systemmodelle wiederzuverwenden.	M
fP3	Die Software ermöglicht es, Varianten eines Systems zu erstellen, die eine gemeinsame Basis haben.	A

Tabelle 7.4: Funktionale Anforderungen: Produktlinien & Versionierung

Über die Integration von GitLab und dem Git Plugin in der Entwicklungsumgebung kann eine Versionsverwaltung für das Systemmodell angeboten werden (Anforderung *fP1*). Des Weiteren ist es möglich, frühere Systemmodelle wiederzuverwenden (Anforderung *fP2)*, indem die Repositories als Grundlage für ein neues Repository verwendet werden. Dadurch muss ein Projekt nicht jedes Mal von vorne begonnen werden. Dasselbe kann auch dafür genutzt werden, eine gemeinsame Basis für verschiedene Systemmodelle zu bilden (Anforderung *fP3*), wodurch Änderungen nur an einer Stelle vorgenommen werden müssen.

Authentifizierung & Autorisierung

Nr.	Anforderung	Kano
fA1	Die Software bietet eine externe Authentifizierung mit DLR-User-Daten an.	A
fA2	Die Software ermöglicht, dass bestimmte Projekte, Workspaces und Teile des Systemmodells nur für gewisse Benutzerrollen zugänglich sind.	O
fA3	Die Software ermöglicht es, verschiedene Projekte zu gruppieren.	A

Tabelle 7.5: Funktionale Anforderungen: Authentifizierung & Autorisierung

Im Rahmen der Arbeit wurde keine externe Authentifizierung mit DLR-User-Daten (Anforderung *fA1*) implementiert, da für die Umsetzung kein Zugriff auf DLR-Daten bestand. Die Anforderung *fA2* wird teilweise erfüllt. Durch die Verwendung von Gitlab können Projekte nur bestimmten Nutzerrollen zugänglich gemacht werden. Die Workspaces von Eclipse Che sind ebenfalls über eine Userverwaltung geschützt. Allerdings unterstützt weder EMF.Cloud noch Eclipse Che oder GitLab eine Zugriffskontrolle auf Systemmodellebene. Über Gitlab können Gruppen für Projekte erstellt werden, wodurch für mehrere Projekte dieselben Zugriffsrechte und Sichtbarkeit gesetzt werden können (Anforderung fA3).

Nicht-funktionale Anforderungen

Leistungseffizienz

Nr.	Anforderung	Kano
nfL1	Die Software skaliert mit der Last.	I
nfL2	Die Software ist günstiger oder gleich teuer als die aktuelle Lösung.	A

Tabelle 7.6: Funktionale Anforderungen: Leistungseffizienz

Die Anforderung *nfL1* wird durch den Einsatz von Kubernetes erfüllt. Kubernetes stellt den einzelnen Containern bei mehr Last mehr Ressourcen zur Verfügung. Eine Kostenbetrachtung wurde im Rahmen der Arbeit nicht durchgeführt, wodurch keine Aussage über die Kosten gemacht werden kann. Deswegen kann die Anforderung *nfL2* nicht erfüllt werden. Allerdings ist durch den Einsatz von cloud-native Technologien keine bestimmter Cloud-Anbieter notwendig.

Kompatibilität

Nr.	Anforderung	Kano
nfK1	Die Software kann mit anderer Software Daten austauschen.	A
nfK2	Die Software kann auf bereits existierender Hardware verwendet werden.	M
nfK3	Die Software kann von verschiedenen Geräten aus verwendet werden.	M

Tabelle 7.7: Funktionale Anforderungen: Kompatibilität

Der Austausch von Daten erfolgt über Standards wie REST und JSON-RPC, wodurch die Lösung Daten mit anderer Software austauschen kann (Anforderung *nfK1*). Des Weiteren lässt sich die Lösung durch Kubernetes auf bereits existierender Hardware verwenden, da keine Spezialhardware benötigt wird und die Lösung neben bereits vorhandener Software betrieben werden kann (Anforderung *nfK2*). Dadurch, dass die Lösung standardmäßig eine webbasierte Benutzeroberfläche bietet sowie eine API für das Systemmodell bereitstellt, kann von verschiedenen Geräten aus auf das Systemmodell zugegriffen werden (Anforderung *nfK3*).

Zuverlässigkeit

Nr.	Anforderung	Kano
nfZ1	Die Software ist bei hoher Last erreichbar.	M
nfZ2	Die Software ist bei Fehlern betriebsfähig.	M
nfZ3	Die Software kann bei einem Absturz die Daten wieder in den vorherigen Zustand wiederherstellen.	M

Tabelle 7.8: Funktionale Anforderungen: Zuverlässigkeit

Die Lösung ist auch bei hoher Last erreichbar (Anforderung *nfZ1*), da durch den Einsatz von Kubernetes bei Bedarf mehr Ressourcen allokiert werden. Des Weiteren sorgen Fehler in der Lösung innerhalb eines Workspaces nicht dafür, dass andere Workspaces in ihrem Betrieb gestört werden (Anforderung *nfZ2*). Außerdem werden durch Kubernetes abgestürzte Container automatisch neu gestartet. Das Systemmodell, das im Webbrowser bearbeitet wird, kann beim Absturz des Webbrowsers wiederhergestellt werden (nfZ3), da die Daten auf dem Server liegen. Kommt es zum Absturz des Servers bzw. Workspaces liegen diese weiterhin im Persistence Volume Claim innerhalb des Clusters. Wurden die Daten aber nicht explizit gespeichert, können diese nicht wiederhergestellt werden.

Sicherheit

Nr.	Anforderung	Kano
nfS1	Die Software speichert die Daten des Systemmodells verschlüsselt.	I
nfS2	Die Software verschlüsselt die Daten des Systemmodells auf dem Kommunikationsweg.	I
nfS3	Die Software speichert nicht das komplette Systemmodell lokal.	I
nfS4	Die Software protokolliert Zugriff und Änderungen am Systemmodell.	M
nfS5	Die Software stellt sicher, dass die Systemmodelle aus Geheimhaltungsgründen nur von berechtigten Ingenieuren genutzt werden können.	M
nfS6	Die Software kann auf eigener Hardware deployt werden.	I
nfS7	Die Software kann offline bzw. ohne externe Internetverbindung verwendet werden.	M

Tabelle 7.9: Funktionale Anforderungen: Sicherheit

Die Daten des Systemmodells werden im DLR GitLab abgelegt, in dem die Daten nicht verschlüsselt gespeichert werden. Dasselbe gilt für die Systemmodelle, die innerhalb der Workspaces liegen. Eclipse Che bzw. der Coffee Editor unterstützen das Verarbeiten von verschlüsselten Dateien nicht. Bei der Verschlüsselung der Daten im Repository auf GitLab besteht außerdem das Problem, dass Merge Requests die verschlüsselten Dateien sehen, wodurch ein Austausch über die Änderungen über GitLab nicht stattfinden kann. Ähnliche Probleme tauchen beim Forken eines Projekts auf. Im Rahmen der Arbeit wurde keine Lösung gefunden,

die es ermöglicht die Daten des Systemmodells zu verschlüsseln und dennoch die Funktionen von GitLab für das Concurrent Engineering zu nutzen.

Die Lösung verschlüsselt die Kommunikation über HTTPS (Anforderung nfS2). Die Daten werden nicht lokal gespeichert (Anforderung *nfS3*), sondern liegen entweder im Gitlab oder während der Bearbeitung im Workspace. Die Anforderung wurde von den Ingenieuren als unerhebliches Merkmal deklariert. Allerdings ist es zum Beispiel für den Kunden von großem Interesse, dass geheime Informationen nicht abhandenkommen. Das Protokollieren von Zugriff und Änderungen (Anforderung *nfS4*) wird teilweise erfüllt, da über GitLab nur schreibende Zugriffe protokolliert werden. Die Lösung stellt sicher, dass das Systemmodell nur von berechtigten Ingenieuren genutzt wird (Anforderung *nfS5*), indem das Gitlab Projekt sowie dem Eclipse Che Workspace nur mit gültigen Accountdaten zugegriffen werden kann. Dadurch, dass die Lösung auf Kubernetes aufbaut, kann sie auch auf eigener Hardware deployt werden (Anforderung *nfS6*), wodurch bei besonders kritischen Projekten die Hoheit über die Daten nicht auf Dritte übergeben werden muss. Die Lösung kann teilweise externe Internetverbindung eingesetzt werden (Anforderung *nfS7)*, nachdem sie aufgesetzt wurde und alle externen Container-Images geladen wurden sowie das VCS bzw. GitLab sich im selben lokalen Netzwerk befindet.

Wartbarkeit

Nr.	Anforderung	Kano
nfW1	Die Software ist Open-Source und hat eine freie Lizenz.	I
nfW2	Die Software kann durch Plugins/Extensions erweitert werden.	A

Tabelle 7.10: Funktionale Anforderungen: Wartbarkeit

Die Anforderung *nfW1* wird erfüllt, indem nur Open-Source Software verwendet wird, die eine freie Lizenz besitzt. Die Lösung selbst basiert auf dem Coffee Editor von EMF.Cloud, was den Quellcode über eine Open-Source Lizenz bereitstellt, der im Rahmen der Arbeit erweitert wurde. Des Weiteren kann die Lösung auf mehreren Ebenen erweitert werden, ohne dass Quellcode geschrieben werden muss (Anforderung *nfW2)*. Dazu zählt das Deployen von weiteren Diensten in Cluster, die Erweiterung der Devfile eines Workspaces sowie die Installation von Erweiterungen in der Entwicklungsumgebung. Allerdings funktionieren nicht alle Erweiterungen, die für vscode entwickelt wurden, da die Kompatbilitäts-API nicht die komplette API von vscode abdeckt. Die Inkompatibilität ist auch teilweise auf die genutzte Version Eclipse Theia zurückzuführen, die von dem Coffee Editor verwendet wird.

Portabilität

Nr.	Anforderung	Kano
nfP1	Die Software bietet einen leichten Zugriff auf das Systemmodell.	M
nfP2	Die Software ist auf Standard- Hard- und --Software lauffähig.	M

Tabelle 7.11: Funktionale Anforderungen: Portabilität

Die Lösung bietet über das Bereitstellen einer Webanwendung sowie API für das Systemmodell leichten Zugriff auf das Systemmodell, weil keine Installation von Software nötig ist (Anforderung *nfP1*). Des Weiteren ist die Lösung auf Standard-Hard- und -Software lauffähig, da keine herstellerspezifischen Lösungen verwendet wird, wodurch die Lösung nicht von einem Anbieter abhängig ist (Anforderung *nfP2)*.

Bei der ersten Befragungsrunde äußerten viele der befragten Ingenieure Bedenken, dass diese unsicher seien, ob sie wertvollen Input liefern können. Die Abbildung 7.1 sowie im Folgenden die Abbildung 7.7 zeigen, dass ein Großteil der Anforderungen von allen Ingenieuren als sinnvoll bzw. notwendig bewertet haben.

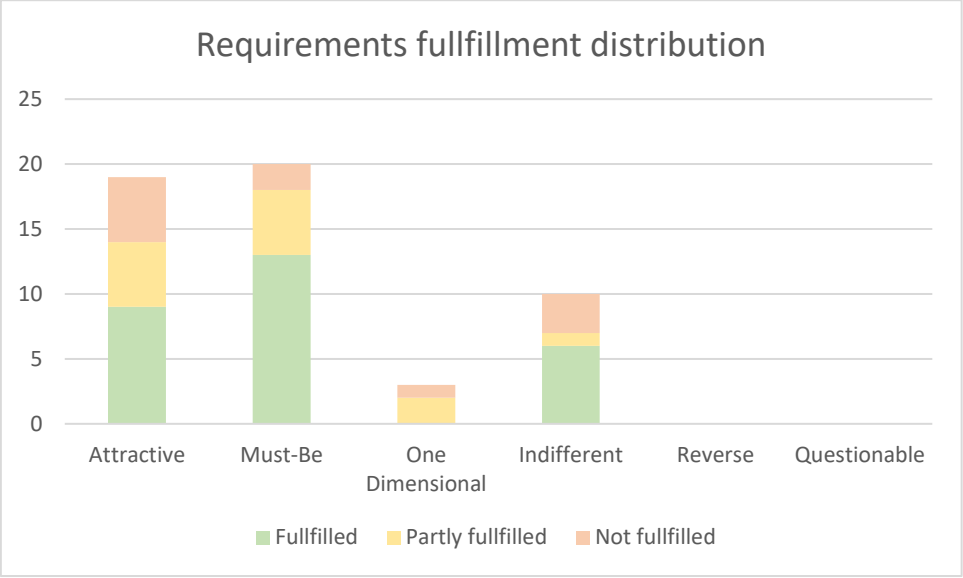


Abbildung 7.7: Erfüllung der Anforderungen nach Kano-Kategorisierung

Insgesamt wurden 52 Anforderungen identifiziert, wovon 28 (53,85 %) voll, 11 (21,15%) teilweise und 13 (25 %) nicht erfüllt wurden. Der hohe Anteil an relevanten Anforderungen (42 bzw. 84%) wurde auch während der zweiten Runde deutlich, in der die Ingenieure die Anforderungsliste generell positiv kommentierten. An dieser Stelle sei allerdings angemerkt, dass nur sieben Ingenieure an der Befragung teilgenommen haben, wodurch das Ergebnis nur als Tendenz gesehen werden kann.

Bei den Begeisterungsmerkmale wurden 9 (47,36%) der Anforderungen erfüllt, 5 (26,31%) konnten nicht erfüllt und 5 (26,31%) konnten teilweise erfüllt werden. 13 (65%) der Basismerkmale konnten erfüllt werden, während 2 (10%) nicht und 5 (25%) der Basismerkmalen teilweise erfüllt werden konnten. Bei den Leistungsfaktoren wurden zwei Anforderungen (66,66%) teilweise erfüllt und eine Anforderung (33,33%) nicht erfüllt. Von den unerheblichen Merkmalen wurden 6 Anforderungen (60%) komplett und 1 Anforderung (10%) teilweise erfüllt sowie 3 (30%) Anforderungen nicht erfüllt.

An dieser Stelle sei angemerkt, dass nicht alle Anforderungen zu jeder Zeit erfüllt werden. Manchmal kommt es vor, dass das Systemmodell nicht bearbeitet werden kann. Des Weiteren werden die Dateien nach dem Wechsel des Branches nicht aktualisiert. Die Validierungsfunktionen der formularbasierten Editoren erkennen Zahlen manchmal als String und zeigen deswegen eine ungültige Eingabe an. Diese Fehler lassen sich durch das Neuladen der IDE im Browser beheben. Fehler, die sich nicht so leicht beheben lassen, sind, dass ein Workspace sich nicht mehr mit dem GitLab authentifizieren kann, nachdem dieser nach Inaktivität gestoppt wurde. Beim erneuten Starten des Workspaces funktioniert zwar die IDE, aber die hinterlegten SSH-Keys werden nicht mehr akzeptiert.

7.3.2 Prozessunterstützung für Raumfahrtsysteme

Im Folgenden werden die Ergebnisse der zweiten Forschungsfrage diskutiert.

Produktlinien, Versionierung, Synchronisierung

Die Nutzung eines VCS wie GitLab ermöglicht zwar die Erstellung von Produktlinien, Versionierung und das Synchronisieren, allerdings sind diese für Quellcode ausgelegt. Das Systemmodell wird bei der implementierten Lösung in XML gespeichert, was bei großen Modellen das Auflösen von bei Merge-Konflikten erschwert. Wenn ein Merge-Konflikt zwischen zwei XML oder CSV-Dateien auftritt, wird weitere Software benötigt, die den Vorgang unterstützt. Das VCS an sich erfüllt damit die Minimalanforderungen. Des Weiteren besteht die Vermutung, dass vor allem Ingenieure, die eine Bearbeitung über Kalkulationstabellen gewohnt sind, durch ein VCS überfordert werden. Deswegen sollte bei der Einführung eines solchen Systems darauf geachtet werden, dass die Ingenieure entsprechend geschult werden. Wenn so das VCS angenommen wird, können die Ingenieure von weiteren Funktionen wie dem Aufgabenmanagement oder der Merge-Request-Funktion profitieren.

Privacy

Der gewählte Ansatz für das Concurrent Engineering erwies sich als inkompatibel mit der Anforderung, dass die Daten des Systemmodells durch zum Beispiel Verschlüsselung vor unbefugten Zugriff geschützt werden. Lediglich die Kommunikation erfolgt verschlüsselt durch den Einsatz von HTTPS. Projekte wie git-crypt oder git-secret sind zum Beispiel dafür ausgelegt, einzelne Dateien und nicht das ganze Repository zu verschlüsseln. Projekte, die dafür ausgelegt sind, ganze Repositories zu verschlüsseln und Open-Source sind, konnten nicht gefunden werden.

Abbildung von verschiedenen Sichten auf das Systemmodell

Durch die Verwendung von EMF.Cloud mit der Verbindung mit Engineering Categories ist es möglich, verschiedene Sichten auf das Systemmodell abzubilden. Allerdings fehlen EMF.Cloud noch viele Funktionen von EMF. In EMF.Cloud unterstützt zwar immer mehr Funktionen von EMF, allerdings fehlt die Generierung von Standard-Editoren komplett. Dadurch muss die Benutzeroberfläche auf der Clientseite manuell entwickelt werden.

Abbildung von verschiedenen Abstraktionsebenen auf das Systemmodell

Wie in dem vorherigen Punkt bereits angesprochen, unterstützt EMF.Cloud nicht alle Funktionen von EMF. Allerdings passiert das Definieren der Abstraktionsebenen selbst auch mit EMF.Cloud in EMF selbst, wodurch bei den Abstraktionsebenen dieselben Möglichkeiten wie bei EMF bestehen. Einzig bei dem Editor muss, wie oben bereits beschrieben, der komplette Editor implementiert werden.

7.3.3 Skalierung von Simulationen und Berechnungen

Bei der Berechnung der Gesamtmasse als auch bei der Deadlock-Überprüfung zeigt sich bei der Ausführung im Cluster ein ähnliches Bild. Bei den sequenziellen Anfragen bleiben beide Graphen bis zu einer kritischen Stelle vom Aufwand konstant. Ab einer bestimmten Menge wächst die benötigte Zeit in Abhängigkeit der MassCategories bzw. Workflows an. Bei den Versuchen mit mehreren Anfragen gleichzeitig zeigt sich, dass die benötigte Zeit direkt mit der Anzahl der MassCategories bzw. Workflows erhöht.

Beim Vergleich von der Ausführung im Cluster und zur lokalen Ausführung fällt auf, dass die Unterschiede bei der Berechnung der Gesamtmasse höher ist als bei dem Modelchecker. Als Ursache wird vermutet, dass die Kommunikation bei der Massenberechnung einen größeren Anteil an der Gesamtzeit ausmacht als beim Modelchecker.

Sowohl bei den Messungen, bei der Massenberechnung als auch beim Modelchecker enthielten die Daten einige Ausreißer. Als Ursache wird vermutet, dass trotz Beenden aller nicht notwendigen Programme Prozesse des Betriebssystems Einfluss auf das Zeitverhalten genommen haben. Des Weiteren wurden die Messungen innerhalb einer virtuellen Maschine aufgenommen, wodurch zwei Betriebssysteme Einfluss auf das Zeitverhalten ausüben konnten. Die Messungen pro Datenpunkt dauerten bis zu 16 Stunden (Modelchecker mit 7 Workflows), in denen der Laptop ausgelastet war und durch die Wärmeentwicklung eventuell den CPU-Takt drosseln musste.

Es wird vermutet, dass das lokale Cluster eine bessere Latenz aufweist, als ein Cluster das in einer Cloud deployt wurde. Dadurch könnten bei der Evaluierung in einer Cloud die Unterschiede zur lokalen Ausführung noch stärker anwachsen.

7.3.4 Gefährdung der Validität

Die in diesem Kapitel dargestellten Punkte können die Validität der Ergebnisse gefährden. Nach der eigenen Einschätzung eignen sich User Stories mit nur einer Rolle nicht, um alle Anforderungen an die Lösung zu identifizieren. Einem Ingenieur könnten Sicherheitsanforderungen nicht bewusst oder nicht wichtig sein, während diese Anforderungen bei der IT-Abteilung von besonderem Interesse sind. Es gibt neben den Ingenieuren im Rahmen von Designstudien auch andere Rollen wie einen Moderator oder den Kunden, für den das Design erstellt wird.

Bei der Akquisition der Befragungsteilnehmer fiel auf, dass viele Ingenieure Erfahrungen mit dem Entwurf von Raumfahrtsystemen besitzen, jedoch nicht mit MBSE und umgekehrt.Das erschwerte die Akquisition von Teilnehmern, was in einer geringen Anzahl von sieben Teilnehmern für die erste Runde resultierte. Von einem Teilnehmer wurde der Fragebogen in der zweiten Runde nicht ausgefüllt, wodurch die finale Teilnehmerzahl auf sechs gesunken ist.

Bei der Befragung zur Kano-Kategorisierung fiel auf, dass die Anforderung *fC6* nicht atomar war, was einem Teilnehmer die Beantwortung erschwerte. Des Weiteren konnten keine technischen Anforderungen kategorisiert werden, da Endanwender befragt wurden.

Bei der Messung der Performanz auf der zur Verfügung stehenden Hardware fiel auf, dass es regelmäßig Ausreißer bei den Messwerten gab, die ein Vielfaches des Medians betrugen. Das könnte dafür sprechen, dass das Messen innerhalb einer virtuellen Maschine nicht optimal ist. Des Weiteren wurde die Messung vom Cluster Internal Service über einen Webbrowser angestoßen, was ebenfalls Einflüsse auf das Laufzeitverhalten haben kann.

8 Zusammenfassung und Ausblick

Der dokumentenbasierte Ansatz hat sich beim Systems Engineering für komplexe Systeme als nicht geeignet herausgestellt. MBSE muss eine große Vielfalt an Konfigurationen unterstützen, da es mit einer Vielfalt an Geräten, Software und Schnittstellen arbeitet. Der Ressourcenbedarf der verwendeten Software kann die Kapazitäten eines lokalen Computers überschreiten. Klassische Desktopanwendungen werden diesen steigenden Anforderungen nicht mehr gerecht. Cloud-Computing verspricht den steigenden Anforderungen gerecht zu werden, indem es zum Beispiel für Simulationen und Berechnungen nahezu unendliche Systemressourcen bereitstellen kann.

Es existieren MBSE-Tools für Raumfahrtsysteme, aber keines von denen ist cloud-native oder nutzt die Vorteile einer Cloud-Umgebung. Es existieren webbasierte MBSE-Tools, diese besitzen aber kein erweiterbares Datemodell. In den letzten Jahren wurde Open-Source Projekte wie Eclipse Theia, EMF.Cloud und Eclipse Che entwickelt, die es möglich machen eine cloud-native Entwicklungsumgebung für MBSE zu entwickeln. Es gibt aber noch keine Lösung die die aktuellen Technologien mit einem erweiterbaren Datenmodell und der Integration von Cloudressourcen für Berechnungen und Simulationen einsetzt.

Für die Ermittlung der Anforderungen wird ein Fragebogen mit zwei Runden durchgeführt. In der ersten Runde werden Anforderungen als User Stories gesammelt. In der zweiten Runde erfolgt eine Kategorisierung der Anforderungen nach dem Kano-Modell

Die implementierte Lösung wurde lokal auf einer virtuellen Maschine mit Minikube deployt. Als Workspace-Server wurde Eclipse Che verwendet. Die entwickelte Entwicklungsumgebung setzt auf den Coffee Editor von EMF.Cloud auf. Das erweiterbare Datenmodell wurde mithilfe eines Vererbungsansatzes implementiert. Das Concurrent Engineering wurde mithilfe von GitLab und einem Trunk-based Workflow implementiert.

Insgesamt wurden 52 Anforderungen identifiziert, wovon 19 als Begeisterungs-, 20 als Basis-, 3 als Leistungs- und 10 als unerhebliches Merkmal identifiziert wurden. Keine Anforderung wurde als Rückweisungsmerkmal oder fragwürdig kategorisiert. Von den Anforderungen wurden 28 (53,85 %) voll, 11 (21,15%) teilweise und 13 (25 %) nicht erfüllt.

Bei der Performanzanalyse zeigte sich, dass aufwendige Berechnungen ähnlich skalieren wie bei einer lokalen Ausführung. Werden simplere Berechnungen durchgeführt, ist die lokale Berechnung deutlich schneller.

Ein nächster möglicher Schritt wäre es, die implementierte Lösung in weiteren Arbeiten in einer Cloudumgebung zu deployen. In diesem Zuge wäre es möglich, die Performanzevaluierungen mit einer realistischen Latenz vorzunehmen. Neben REST ist eine Evaluierung mit gRPC, GraphQL, WebSockets und JSON-RPC für die Performanzevaluierungen denkbar. Neben dem Deployment in einer Cloud könnte auch ein Multi-Cloud Deployment oder das Deployment in einer GAIA-X-konformen Cloud durchgeführt werden. Bei dem Deployment in einer Cloud wäre ein weiterer Schritt eine Evaluierung in Bezug auf Kosten und Laufzeit von Serverless Functions für ressourcenintensive, aber seltene Berechnungen. Des Weiteren besteht die Möglichkeit, zu evaluieren, ob das NodeJS- und das Java-Backend sowie Services wie die Massenberechnung und der Modelchecker über WebAssembly in die Entwicklungsumgebung integrierbar sind. Dadurch könnte die Entwicklungsumgebung auch ohne eine Internetverbindung zum Cluster verwendet werden.

Außerdem könnten die Anforderungen erfüllt werden, die im Rahmen dieser Arbeit nicht erfüllt werden konnten. Für den Einsatz des VCS wird außerdem eine benutzerfreundlichere Möglichkeit benötigt, Merge-Konflikte aufzulösen. Die Freigabe eines Workspaces in Eclipse Che sollte aktuell vermieden werden, da sich in dem Workspace sensible Daten wie SSH-Schlüssel für Git befinden. Ein weiteres Problem beim Teilen des Workspaces ist es, dass die „last write wins policy“ fehleranfällig ist, weswegen eine Extension entwickelt werden könnte, die konfliktfreies Kollaborieren in einem Workspace ermöglicht. Außerdem könnten Erweiterungen des Datenmodells wie bei Virtual Satellite über Extensions oder Plugins installiert werden, um neben Datenmodellerweiterungen auch Softwarepakete zu installieren.

Das implementierte Datenmodell könnte durch ein Datenmodell auf Basis des Type-Of und Dynamic Template Pattern ersetzt werden, welches in z. B. Virtual Satellite genutzt wird. Dadurch besäße das Datenmodell weitreichenderer Funktionen zur Erweiterung des Datenmodells. Des Weiteren ist es nicht möglich, wie bei EMF einen Standard-Editor aus dem Datenmodell zu generieren, der den Einstieg erleichtern würde.

Zur Beschleunigung der Entwicklung wäre es eine Möglichkeit, die Entwicklungsumgebung über die Entwicklungsumgebung zu entwickeln wie es auch bei Che-Theia und Virtual Satellite möglich ist. Im Fall von Che-Theia kann die Entwicklungsumgebung direkt in einer produktionsähnlichen Umgebung entwickelt werden.

Dadurch, dass die Server-Implementierung von vscode veröffentlicht wurde und Che-Theia aktuell nicht weiterentwickelt wird [105], könnte in einer zukünftigen Arbeit evaluiert werden, ob EMF.Cloud für vscode angepasst werden kann.

I. Anhang

A1. Tabellen Anforderungsermittlung

Technik	Bewertung oder Sortierung	Anzahl der Kriterien	Anzahl der Schätzungen pro Anforderung	Anzahl Schätzungen für neue Anforderung	Skalentyp	Konsolidierung
Planning Poker mit Story Points oder T-Shirt-Größen	B	1	2	2	kardinal	Diskussion bis Konsens oder Mittelwert
Bücket Estimation	B	1	1 bis 3	1 bis 3	kardinal	Mehrfachbegutachtung
Punktekleben, 100 €-Methode	B	1	0 bis 1	1 bis n	kardinal	Addieren der Punkte
Top-Ten-Methode	B & S	1	0 bis 1	1 bis 11	nominal	Addieren der Punkte bzw. Modus
Ein-Kriterium-Klassifikation	B	1	1	1	alle denkbar	je nach Skala
Zwei-Kriterien-Klassifikation	B & S	2	2	2	alle denkbar	je nach Skala
Kano	B	2	2	2	nominal	Modus, siehe auch [Klop12]
Wiegerssche Formel	S	4	4	4	kardinal	Mittelwert
Ranking	S	1	mindestens $n-1$	2 bis n	ordinal	Median
Bubble Sort	B & S	1	maximal n $(n-1)/2$	n	ordinal	Median
AHP	B & S	1	n $(n-1)/2$	n	kardinal	Mittelwert

Abbildung A.1: Vergleich verschiedener Bewertungs- und Sortierungstechniken für Anforderungen [95]

–		Wie wäre es, wenn die Anforderung erfüllt wird?				
		sehr gut	gut	egal	schlecht	sehr schlecht
Wie wäre es, wenn die Anforderung nicht erfüllt wird?	sehr gut	unerhebliches Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal
	gut	Begeisterungs-Merkmal	unerhebliches Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal
	egal	Begeisterungs-Merkmal	Begeisterungs-Merkmal	unerhebliches-Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal
	schlecht	Leistungs-Merkmal	Leistungs-Merkmal	Basis-Merkmal	unerhebliches-Merkmal	Rückweisungs-Merkmal
	sehr schlecht	Leistungs-Merkmal	Leistungs-Merkmal	Basis-Merkmal	Basis-Merkmal	unerhebliches-Merkmal

Abbildung A.2: Bewertungstabelle nach Puliot [100]

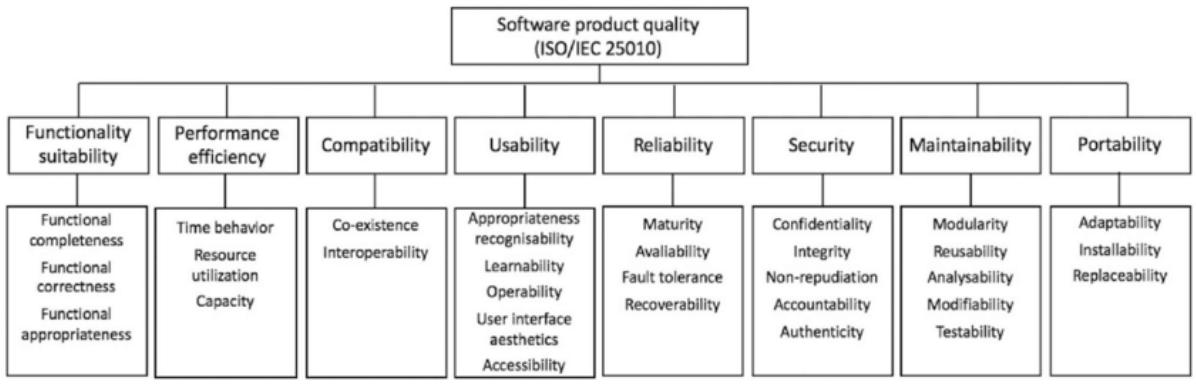


Abbildung A.3: ISO/IEC 25010 [106]

A2. Funktionale Anforderungen

A2.1 Modellierung

Anf.Nr.	Als	, möchte ich	, um	Erfüllt, wenn (Akzeptanzkriterien)
fM1	Ingenieur	ein System modellieren	Eine „single source of truth" zu haben und einfachen Zugriff auf alle Daten zu haben	- Das System hierarchisch aus Subsystemen mit Eigenschaften dargestellt werden kann - zusammengehörige Informationen gemeinsam gespeichert und verknüpft werden können - mit einer einfachen Modellierungssprache/Notationselementen das System beschrieben werden kann
fM2	Ingenieur	An einem eigenen Subsystem für meine Domäne arbeiten	Einen eigenen Arbeitsbereich zu haben und Konflikte zu minimieren	- Für eine Domäne ein eigenes Subsystem modelliert werden kann
fM3	Ingenieur	unterschiedliche Sichten auf die Daten haben oder selbst definieren können	die jeweilige Sicht übersichtlich zu halten und fokussieren zu können (z. B. nur CAN-Bus-Leitungen anzeigen)	- Unterschiedliche Sichten für das Systemmodell existieren - Daten aus den Sichten mit den Daten aus dem Systemmodell synchronisiert werden
fM4	Ingenieur	die Modellierungssprache bei Bedarf erweitern können	die Modellierungssprache einfach zu halten und an meine Anforderungen anzupassen	- In Abhängigkeit der Anforderungen Modellierungsmöglichkeiten hinzugefügt werden können
fM5	Ingenieur	Berechnungen und Referenzen definieren können	Bei Veränderungen von Eigenschaften Werte nicht erneut per Hand eingeben zu müssen	- Eigenschaften des Modells in Abhängigkeit von anderen Eigenschaften berechnet werden können - Eigenschaften des Modells referenziert werden können
fM6	Ingenieur	Anforderungen definieren und mit Elementen vom Systemmodell verknüpfen können	Die Anforderungen an das System im Blick zu haben und prüfen zu lassen	- Anforderungen als Teil des Systemmodells definiert werden können - Anforderungen mit anderen Elementen des Systemmodells verknüpft werden können - Anforderungen können automatisch geprüft werden

fM7	Ingenieur	Simulationen durchführen	Bereits während der Modellierung zu überprüfen ob das modellierte System den Anforderungen gerecht wird oder um es mit alternativen Modellierungen zu vergleichen	- Domänenspezifische Simulationen durchgeführt werden
fM8	Ingenieur	Das Systemmodell aus Konsistenz prüfen	Fehler in der Modellierung aufzudecken	- Das Systemodell (nach definierten Regeln) auf Konsistenz überprüft werden kann
fM9	Ingenieur	Eine Impact Analysis durchführen können	Auswirkungen von Änderungen darzustellen und abschätzen zu können	- Es werden die Auswirkungen von Änderungen dargestellt, die von einer Änderung betroffen sind
fM10	Ingenieur	Daten in weitere Lebensphasen des Raumfahrtlebenszyklus übernehmen	Beim Übergang in die nächste Lebensphase keine Informationen zu verlieren	- Modellierungen der späteren Lebensphasen Informationen aus den früheren Lebensphasen wiederverwenden

A2.2 Concurrent Engineering & Synchronisierung

fC1	Ingenieur	Gleichzeitig mit anderen Ingenieuren Zugriff auf das Systemmodell haben	Gleichzeitig am Systemmodell arbeiten zu können ohne sich mit anderen Nutzern zeitlich abzusprechen	- das Model/System von mehreren Personen bearbeitet werden kann ohne Inkonsistenzen oder Überschreibungen zu verursachen
fC2	Ingenieur	Bei auftretenden Merge-Konflikten zwischen zwei Versionen den Konflikt auflösen können	Änderungen von mehreren Ingenieuren mergen zu können	- die Unterschiede der beiden Versionen dargestellt werden - Das Werkzeug beim Zusammenführen der Versionen unterstützt
fC3	Ingenieur	Änderungen am Datenmodell in einer Kopie vornehmen können, ohne das gemeinsame Systemmodell zu überschreiben	Designvarianten durchzuspielen, ohne das gesamte Team damit zu stören.	- Eine Kopie des Systemmodells erstellt werden kann mit der Änderungen vorgenommen werden können, ohne die anderen Ingenieure zu stören
fC4	Ingenieur	Daten in einem Systemmodell als Favoriten markieren	Über Änderungen die für mir wichtig sind informiert zu werden	- Elemente und Eigenschaften einem Systemmodell als Favorit markiert werden können - Wenn ich per Benachrichtigung über Änderungen informiert werde
fC5	Ingenieur	über aktuell eingecheckte Modelländerungen von Kollegen	wichtige Änderungen sofort erhalten zu können und für mein	- Wenn ich per Benachrichtigung über Änderungen informiert werde

		umgehend informiert werden und entscheiden können, ob ich das Modellupdate sofort oder später übernehmen möchte	Teilmodell kritische Updates später mit mehr Zeit machen zu können.	
fC6	Ingenieur	Den aktuellen Status des Systemmodells sehen, ohne auf alle Commits der anderen Ingenieure warten zu müssen	Ein möglichst aktuelles Datenmodel zu haben	- Alle Kalkulationen im Modell unabhängig vom einzelnen Nutzer und seinen Rechten laufen
fC7	Ingenieur	In dem Model Informationen/Daten als sichtbar und unsichtbar deklarieren können	Sicherzustellen, dass nur fertige Informationen weiter verwendet werden können, und um die Möglichkeit zu haben, „Draft“ Daten für weitere Bearbeitung zu speichern	- Eigenschaften des Modells als Draft oder unsichtbar deklarieren zu können - Eigenschaften des Modells als Release oder visible deklariert werden können
fC8	Ingenieur	Sehen, welche anderen Ingenieure derzeit das Model bearbeiten oder eingeloggt sind	gezielt eine Kommunikation mit anderen Ingenieuren aufbauen zu können und doppelte Arbeit und Überschreiben zu vermeiden	- In dem Systemmodel die Ingeneiure (analog zu der Teilnehmerliste in Videokonferenz-Tools) in einem dedizierten Frame/einer dedizierten Ansicht sichtbar sind - Das Systemmodel mir anzeigt (farblich, via Tag oder Zeichen) welche Daten aktiv bearbeitet werden
fC9	Ingenieur	Kommentare (an spezielle Personen) an Elemente im Systemmodell hinterlegen	Fragen zu stellen, Probleme aufzuzeigen, Entscheidungen zu dokumentieren oder Erklärungen zu hinterlegen	- Kommentare an Modellelemente anzuheften und wenn nötig Personen damit zu verknüpfen - Die verknüpfte Person eine Benachrichtigung über den Kommentar erhält
fC10	Ingenieur	Sehen was sich seit meiner letzten Session geändert hat	Änderungen von anderen Nutzern im Blick zu haben und Änderungen/Auswirkungen nachzuverfolgen	- bei Beginn der Arbeit ein Change log gezeigt bekommt. - Eine Versionshistorie mit den genauen Änderungen einsehen zu können
fC11	Ingenieur	Meinen Workspace mit einem anderen Ingenieur teilen	Gemeinsam an einem Problem zu arbeiten oder Hilfe zubekommen, ohne dass der andere Ingenieur das Projekt bei sich selbst aufsetzen/starten muss	- Anderen Ingenieuren Zugriff auf meinen Workspace gegeben werden kann - Der Workspace über einen Link mit einem anderen Nutzer geteilt werden kann

fC12	Ingenieur	einen toolgestützten Change-Management-Prozess für jegliche Systemmodellelemente einrichten können	um Änderungswünsche mit betroffenen Experten (Change-Control-Board) diskutieren und beantworten zu können.	- Eine Diskussionsplattform für Änderungen bereitgestellt wird
fC13	Ingenieur	ein toolgestütztes Task-Management einrichten können	Die Entwicklung des Modells zu steuern oder Aufgaben an die jeweiligen Experten zu delegieren	- Aufgaben an andere Nutzer (oder mich selbst) vergeben können - Aufgaben an einzelnen Bereichen des Modells hinterlegt werden können und mit Personen/Entwicklern verknüpft werden können. - Die verknüpfte Person eine Benachrichtigung über die neue Aufgabe erhält (zb. email)
fC14	Ingenieur	Die Möglichkeit haben auch offline am Datenmodell weiterarbeiten zu können	Auch lokal am Datenmodell weiterzuarbeiten zu können (z.B. auf Dienstreisen).	- Das Systemmodell als lokale Arbeitskopie offline bearbeitet werden kann - Die lokale Arbeitskopie bei Internetzugang mit dem zentralen Systemmodell synchronisiert werden kann

A2.3 Schnittstellen & Exportieren von Daten

fS1	Ingenieur	Das Systemmodell über eine Web-API bereitstellen	Die Daten der Systemmodelle anderer Software zur Verfügung zu stellen und von externer Software angetriggert werden zu können	- Das Modell über eine Web-API zur Verfügung gestellt wird, wodurch das Systemmodell auch ohne die Benutzeroberfläche bearbeitet werden kann
fS2	Ingenieur	Externe Software über deren Web-API ansteuern	Daten von anderer Software zu verwenden und proaktiv Daten an andere Software zu schicken	- Web-APIs von anderen Tools angesprochen werden können
fS3	Ingenieur	Daten aus dem Systemmodell exportieren	Um Daten mit Kollegen austauschen zu können oder in anderer Software verwenden zu können	- Daten können in typischen Formaten (z.B. csv) exportiert werden

A2.4 Produktlinien & Versionierung

fP1	Ingenieur	eine Versionsverwaltung für das Systemmodell nutzen	Änderungen anderer Ingenieure nachvollziehen zu können und zu einer älteren Version des Projekts gehen zu können	- alle eingecheckten Änderungen mit Autor, Datum und Kommentar in einem Log gelistet werden - Änderungen verworfen werden können
fP2	Ingenieur	Daten früherer Systemmodelle wiederverwenden können.	Nicht jedes Mal von vorne beginnen zu müssen und Zeit zu sparen	- eine Bibliothek mit bereits erstellten Modellen zur Verfügung steht
fP3	Ingenieur	Varianten eines Systems erstellen, die eine gemeinsame Basis haben	Änderungen in der Basis in den Varianten übernehmen zu können, ohne diese für jede Variante durchzuführen	- ausgehend von einem Basismodell Varianten abzweigen können, die die Eigenschaften des Basismodells erben

A2.5 Authentifizierung & Autorisierung

fA1	Ingenieur	Mich mit meinen normalen DLR-User-Daten über ein Drittsystem authentifizieren	meine Identität zu beweisen sowie damit von der Software keine Passwörter gespeichert werden müssen	- der Login mit DLR-Daten außerhalb der Software - Es werden keine Passwörter gespeichert
fA2	Ingenieur	Dass bestimmte Projekte, Workspaces und Teile des Systemmodells nur für gewisse Benutzerrollen zugänglich sind	Unbefugten Zugriff zu vermeiden	- Zugriffsrechte auf Projektebene vergeben werden können - Zugriffsrechte zum Modifizieren einzelner Bereiche des Systemmodells individuell vergeben werden können - Zugriffsrechte auf Workspaces der Ingenieure vergeben werden können
fA3	Ingenieur	Verschiedene Projekte gruppieren	Zugriffsrechte zu vererben.	- Projekte zu Gruppen hinzugefügt werden können, die die Sichtbarkeit und Zugriff regeln

A3. Nicht-funktionale Anforderungen

A3.1 Leistungseffizienz

nfL1	Ingenieur	Dass die Software mit der Last skaliert	Auch bei erhöhter Last keine Performanceprobleme vorkommen	- Bei Bedarf die Software mehr Ressourcen allokiert
nfL2	Ingenieur	Dass die Software günstiger oder gleichteuer als die aktuelle Lösung ist	Mit dem gleichen Budget wie zuvor auszukommen	- Wenn der Betrieb der Software bei gleicher Leistung günstiger oder gleichteuer im Vergleich zur aktuellen Lösung ist - Bei Preisänderungen auf Alternativen gewechselt werden kann

A3.2 Kompatibilität

nfK1	Ingenieur	dass die Software mit anderer Software Daten austauschen kann	Für die einzelnen Arbeitsschritte die beste Software zu verwenden oder den Einstieg in die Software zu erleichtern	- Daten mit anderer Software in Standardformaten ausgetauscht wird
nfK2	Ingenieur	Dass die Software auf bereits existierender Hardware verwendet werden kann	Neuanschaffungen zu vermeiden und bereits vorhandene Hardware weiterzuverwenden	- Die Software keine Spezialhardware benötigt - Die Software neben bereits vorhandener Software verwendet werden kann
nfK3	Ingenieur	Von verschiedenen Geräten aus auf das Systemmodell zugreifen können	Schnell und Flexibel Informationen erhalten und bereitstellen zu können sowie sich direkt und ohne Installation und Wartung um die Produktentwicklung zu kümmern	- Das Systemmodell via Log-in von einem Desktop PC oder mobilen Endgerät erreicht werden kann - Das Model ohne Installationsprozeduren von mir und anderen via Link geöffnet werden kann

A3.3 Usability

Wird im Rahmen der Arbeit nicht betrachtet, da ein Prototyp mit eingeschränktem Funktionsumfang entwickelt wird.

A3.4 Zuverlässigkeit

nfZ1	Ingenieur	dass die Software auch bei hoher Last erreichbar ist	Um bei meiner Arbeit nicht unterbrochen zu werden	- Die Software bei erhöhter Last mehr Ressourcen allokiert
nfZ2	Ingenieur	Dass die Software auch bei Fehlern betriebsfähig bleibt	Meine Arbeit in fehlerfreien Bereichen weiter fortzuführen	- Fehler in der Soft- oder Hardware sorgen nicht für den kompletten Ausfall der Software
nfZ3	Ingenieur	Dass beim Absturz der Software der vorherige Zustand wieder hergestellt werden kann	Keinen Fortschritt beim Absturz des Systems zu verlieren	- Nicht gespeicherte Informationen auch ohne explizites Speichern wiederhergestellt werden können

A3.5 Sicherheit

nfS1	Ingenieur	Dass die Daten des Systemmodells beim Speichern verschlüsselt werden	Die Daten vor unbefugten Zugriff zu schützen	- Die Daten des Systemmodells sind außerhalb der Software nach Stand der Technik verschlüsselt
nfS2	Ingenieur	Der Datenaustausch erfolgt verschlüsselt	Die Daten auch beim Abfangen der Datenpakete zu schützen	- Die Kommunikation ist nach dem Stand der Technik verschlüsselt
nfS3	Ingenieur	Dass das komplette Systemmodell nicht lokal abgespeichert wird	Zu verhindern, dass komplette Systemmodell durch einen Ingenieur entwendet werden kann	- Das Datenmodell extern zentral und nicht lokal gehalten und abgespeichert wird
nfS4	Ingenieur	Möchte ich, dass Zugriff und Änderungen am Systemmodell protokolliert werden	Nachzuvollziehen wer auf das Systemmodell zugegriffen hat und welche Änderungen durchgeüfhrt wurden	- Die Zugriffsart und -zeitpunkt mit Nutzernamen protokolliert wird
nfS5	Ingenieur	Sicherstellen können, dass die Datenmodelle aus Geheimhaltungsgründen ausschließlich von berechtigten Ingenieuren genutzt werden können	Datenkritische Projekte mit z.B. der Industrie durchführen zu können und auszuschließen, dass Unbefugte Zugriff auf die Datenmodelle haben	- Eindeutige Rechtevergabe, Zugriffskontrolle und Eingabeprotokoll. - Gaia-X Kompatibilität
nfS6	Ingenieur	Dass das System auf eigener Hardware deployt werden kann	Geheimhaltungsklauseln gerecht zu werden	- Das System benötigt keine exotische Hardware oder Software
nfS7	Ingenieur	dass die Software offline bzw. ohne externe Interverbindung verwendbar ist	Das System von Zugriff von außen zu schützen	- Die Software benötigt nach der Einrichtung keine externe Internetverbindung mehr

				- Es bestehen keine Abhängigkeiten zu externen Diensten - Ingenieure außerhalb des Netzwerks können sich über VPN mit der Software verbinden

A3.6 Wartbarkeit

nfW1	Ingenieur	Eine Software nutzen die Open-Source ist und eine freie Lizenz hat	Funktionen selber beitragen zu können und die Software überprüfbar zu machen	- Der Quellcode für den Nutzer einsehbar ist - Der Quellcode unter einer freien Lizenz veröffentlicht ist
nfW2	Ingenieur	Dass die Software durch Plugins/Extensions erweiterbar ist	Erweiterungen auch ohne Änderungen am Quellcode vorzunehmen und die Software erst bei Bedarf mit weiteren Funktionen zu erweitern und damit nicht zu Beginn zu überladen	- Die Software unterstützt einen Mechanismus zur Erweiterung der Software - Die Software kann je nach projektspezifischen Anforderungen mit Plugins/Extensions erweitert werden

A3.7 Portabilität

nfP1	Ingenieur	Möchte ich leichten Zugriff auf das Systemmodell haben	Eine geringe Nutzungshürde zu haben und das Systemmodell schnell bearbeiten zu können	- Keine besondere Installation der Software nötig. (einfaches Entpacken oder Zugriff via Browser)
nfP2	Ingenieur	Dass die Software auf Standard-Hard- und Software lauffähig ist	Spezialanschaffungen oder Abhängigkeiten zu vermeiden	- Die Software funktioniert auf gängiger Soft- und Hardware - Die Software verwendet keine herstellerspezifischen Lösungen - Das System ist nicht von einem Anbieter abhängig

II. Literaturverzeichnis

[1] „Systems Engineering Definition“, *INCOSE*. https://www.incose.org/about-systems-engineering/system-and-se-definition/systems-engineering-definition (zugegriffen 2. Mai 2022).

[2] S. Friedenthal, A. Moore, und R. Steiner, Hrsg., *A Practical Guide to SysML*, 3. Aufl. Boston: Morgan Kaufmann, 2015. doi: 10.1016/B978-0-12-800202-5.01001-8.

[3] S. Sheard *u. a.*, „A complexity primer for system engineers“, *INCOSE Complex Systems Working Group White Paper*, Bd. 1, S. 1–10.

[4] ECSS Secretariat ESA Estec, „ECSS-M-ST-10C Rev.1 – Project planning and implementation (6 March 2009) | European Cooperation for Space Standardization“, 2009

[5] National Aeronautics and Space Administration, *NASA Systems Engineering Handbook (NASA SP-2016-6105 Rev2 supersedes SP-2007-6105 Rev 1)*. Washingthon, D.C., 2007.

[6] D. D. Walden, G. J. Roedler, und K. Forsberg, „INCOSE Systems Engineering Handbook Version 4: Updating the Reference for Practitioners“, *INCOSE International Symposium*, Bd. 25, Nr. 1, S. 678–686, 2015, doi: 10.1002/j.2334-5837.2015.00089.x.

[7] P. M. Fischer, M. Deshmukh, V. Maiwald, D. Quantius, A. Martelo Gomez, und A. Gerndt, „Conceptual Data Model - A Foundation for Successful Concurrent Engineering“, *Concurrent Engineering Research and Applications*, Nov. 2017.

[8] G. Sohlenius, „Concurrent Engineering“, *CIRP Annals*, Bd. 41, Nr. 2, S. 645–655, Jan. 1992, doi: 10.1016/S0007-8506(07)63251-X.

[9] L. Wang, W. Shen, H. Xie, J. Neelamkavil, und A. Pardasani, „Collaborative conceptual design—state of the art and future trends“, *Computer-Aided Design*, Bd. 34, Nr. 13, S. 981–996, 2002, doi: https://doi.org/10.1016/S0010-4485(01)00157-9.

[10] K. J. Gordon, „Spreadsheet or database: Which makes more sense?“, *J. Comput. High. Educ.*, Bd. 10, Nr. 2, S. 111–116, März 1999, doi: 10.1007/BF02948725.

[11] P. M. Fischer, D. Lüdtke, C. Lange, F.-C. Roshani, F. Dannemann, und A. Gerndt, „Implementing model-based system engineering for the whole lifecycle of a spacecraft“, *CEAS Space J*, Bd. 9, Nr. 3, S. 351–365, Sep. 2017, doi: 10.1007/s12567-017-0166-4.

[12] P. Winzer, „Die Systemmodellierung im GSE-Ansatz“, in *Generic Systems Engineering: Ein methodischer Ansatz zur Komplexitätsbewältigung*, P. Winzer, Hrsg. Berlin, Heidelberg: Springer, 2016, S. 115–194. doi: 10.1007/978-3-662-52893-8_3.

[13] W. Schönecker *u. a.*, „Arbeitsgemeinschaft angewandte Lehre und Forschung der GfSE Silos aufbrechen: Ist MBSE die Lösung?“, 26. Juli 2021. doi: 10.13140/RG.2.2.26326.98884.

[14] D. Knoll, C. Fortin, und A. Golkar, „Review of Concurrent Engineering Design practice in the space sector: state of the art and future perspectives“, in *2018 IEEE International Systems Engineering Symposium (ISSE)*, Rome, Okt. 2018, S. 1–6. doi: 10.1109/SysEng.2018.8544387.

[15] A. Braukhane, V. Maiwald, A. Martelo, D. Quantius, und O. Romberg, „BE AWARE OF THE SQUAD: LESSONS LEARNT FROM 50 CONCURRENT ENGINEERING STUDIES FOR SPACE SYSTEMS“, *th International Astronautical Congress*, S. 8.

[16] „Model Based Systems Engineering (MBSE) on AWS: From Migration to Innovation - Model Based Systems Engineering (MBSE) on AWS: From Migration to Innovation“. https://docs.aws.amazon.com/whitepapers/latest/model-based-systems-engineering/model-based-systems-engineering.html (zugegriffen 18. Mai 2022).

[17] A. Braukhane und T. Bieler, „The Dark Side of Concurrent Design: A Story of Improvisations, Workarounds, Nonsense and Success“, gehalten auf der 6th International Conference on Systems & Concurrent Engineering for Space Applications, Stuttgart, Germany, Okt. 2014. Zugegriffen: 14. September 2022. [Online]. Verfügbar unter: https://elib.dlr.de/92977/

[18] M. Lisi, M. Recchioni, und I. Roma, „HOW INDIVIDUAL LEARNING MODELS AND DIDACTIC METHODOLOGIES WILL CHANGE AFTER THE CORONAVIRUS PANDEMIC: THE CASE OF CONCURRENT ENGINEERING“, S. 5.
[19] M. H. Syed und E. B. Fernandez, „The Software Container Pattern“, S. 7.
[20] A. M. Potdar, N. D g, S. Kengond, und M. M. Mulla, „Performance Evaluation of Docker Container and Virtual Machine“, *Procedia Computer Science*, Bd. 171, S. 1419–1428, Jan. 2020, doi: 10.1016/j.procs.2020.04.152.
[21] Docker Inc., „Docker overview“, *Docker Documentation*, 2. November 2022. https://docs.docker.com/get-started/overview/ (zugegriffen 8. November 2022).
[22] A. Puliafito und K. S. Trivedi, Hrsg., *Systems Modeling: Methodologies and Tools*. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-319-92378-9.
[23] C. Carrión, „Kubernetes Scheduling: Taxonomy, ongoing issues and challenges“, *ACM Comput. Surv.*, S. 3539606, Juni 2022, doi: 10.1145/3539606.
[24] Sam newmann, *Building Microservices, 2nd Edition*, 2. Aufl. Sebastopol: O'Reilly Media inc.
[25] P. Mell und T. Grance, „The NIST Definition of Cloud Computing“, National Institute of Standards and Technology, NIST Special Publication (SP) 800-145, Sep. 2011. doi: 10.6028/NIST.SP.800-145.
[26] „CNCF Technical Oversight Committee (TOC)“. Cloud Native Computing Foundation (CNCF), 15. September 2022. Zugegriffen: 18. September 2022. [Online]. Verfügbar unter: https://github.com/cncf/toc/blob/b6d0cdaa7cf5bc32e8494f9738bc2a42d1648bb3/DEFINITION.md
[27] A. Scholz und J.-N. Juang, „Toward open source CubeSat design“, *Acta Astronautica*, Bd. 115, S. 384–392, Okt. 2015, doi: 10.1016/j.actaastro.2015.06.005.
[28] „VSD-Project | Virtual Spacecraft Design“. https://www.vsd-project.org/ (zugegriffen 27. Juli 2022).
[29] „VSD Licenses | VSD-Project“. https://www.vsd-project.org/vsd-licenses/ (zugegriffen 9. Oktober 2022).
[30] I. Ferreira, H. P. de Koning, S. Gerene, A. Pickering, J. Vennekens, und F. Beyer, „Open Concurrent Design Tool: ESA Community Open Source Ready to Go!“, Okt. 2014.
[31] „Simulation and EGSE for Space Programmes (SESP2012)“, *Indico at ESA / ESTEC (Indico)*. https://indico.esa.int/event/94/contributions/3608/ (zugegriffen 24. Juli 2022).
[32] „VSEE Overview | VSD-Project“. https://www.vsd-project.org/vsee-overview/ (zugegriffen 9. Oktober 2022).
[33] J. Rey, „Modeling with VSEE: Definition of Guidelines and Exploitation of the Models“, ESA, YGT Final Report, Aug. 2013. Zugegriffen: 24. Juli 2022. [Online]. Verfügbar unter: https://www.vsd-project.org/download/documents/YGT%20final%20report%20Rey%20V2.pdf
[34] „SSDE | VSD-Project“. https://www.vsd-project.org/ssde/ (zugegriffen 9. Oktober 2022).
[35] „SSRDB | VSD-Project“. https://www.vsd-project.org/ssrdb/ (zugegriffen 11. Oktober 2022).
[36] „VSEE Metamodel | VSD-Project“. https://www.vsd-project.org/vsee-metamodel/ (zugegriffen 6. Oktober 2022).
[37] P. M. Fischer *u. a.*, „Spacecraft Interface Management in Concurrent Engineering Sessions“, in *Cooperative Design, Visualization, and Engineering*, Cham, 2019, S. 54–63. doi: 10.1007/978-3-030-30949-7_7.
[38] „OCDT - Open Concurrent Design Tool“. https://ocdt.esa.int/ (zugegriffen 6. Oktober 2022).
[39] D. Knoll und A. Golkar, „A coordination method for concurrent design and a collaboration tool for parametric system models“, *Concurrent Engineering*, Bd. 26, Nr. 1, S. 5–21, März 2018, doi: 10.1177/1063293X17732374.

[40] C. Bach, C. Drobny, und M. Tajmar, „Trade-off between concurrent engineering software tools for utilisation in space education and beyond“, gehalten auf der 4th Symposium on Space Educational Activities, Apr. 2022. doi: 10.5821/conference-9788419184405.125.
[41] CDP4 Development Team, „CDP4® Integrated Modelling Environment – Version 7.0.1 User Manual“. 16. Februar 2021. Zugegriffen: 7. Oktober 2022. [Online]. Verfügbar unter: http://www.siangyu.com.cn/kbcms/ueditor/php/upload/file/20211027/1635328672784142.pdf
[42] R. Group, „COMET™: Explore the Full Potential of Collaborative Design“. https://products.rheagroup.com/comet (zugegriffen 7. Oktober 2022).
[43] „CDP 4 User Manual - Architecture“. http://cdp4docs.rheagroup.com/?c=B%20Concurrent%20Design%20Platform&p=Architecture.md (zugegriffen 7. Oktober 2022).
[44] „CDP 4 User Manual - Authentification“. http://cdp4docs.rheagroup.com/?c=F%20CDP4%20Web%20API&p=06_WEB_API_Authentication.md (zugegriffen 7. Oktober 2022).
[45] „CDP 4 User Manua - Authorization“. http://cdp4docs.rheagroup.com/?c=F%20CDP4%20Web%20API&p=07_WEB_API_Authorization.md (zugegriffen 7. Oktober 2022).
[46] T. Gateau, L. Senaneuch, S. S. Cordero, und R. Vingerhoeds, „Open-source Framework for the Concurrent Design of CubeSats“, in *2021 IEEE International Symposium on Systems Engineering (ISSE)*, Vienna, Austria, Sep. 2021, S. 1–8. doi: 10.1109/ISSE51541.2021.9582549.
[47] E. Caliò, F. D. Giorgio, und M. Pasquinelli, „Deploying Model-Based Systems Engineering in Thales Alenia Space Italia“, S. 7.
[48] C. Guindon, „Eclipse Sirius, the technology behind Capella | The Eclipse Foundation“. https://www.eclipse.org/community/eclipse_newsletter/2017/december/article2.php (zugegriffen 9. Oktober 2022).
[49] „Case_Study_ArianeGroup.pdf“. Zugegriffen: 11. Oktober 2022. [Online]. Verfügbar unter: https://www.eclipse.org/capella/resources/pdf/Case_Study_ArianeGroup.pdf
[50] „Case_Study_CNES.pdf“. Zugegriffen: 11. Oktober 2022. [Online]. Verfügbar unter: https://www.eclipse.org/capella/resources/pdf/Case_Study_CNES.pdf
[51] P. Roques, „MBSE with the ARCADIA Method and the Capella Tool“, in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, Jan. 2016.
[52] S. Bonnet, J.-L. Voirin, D. Exertier, und V. Normand, „Not (strictly) relying on SysML for MBSE: Language, tooling and development perspectives: The Arcadia/Capella rationale“, in *2016 Annual IEEE Systems Conference (SysCon)*, Apr. 2016, S. 1–6. doi: 10.1109/SYSCON.2016.7490559.
[53] „eclipse/capella“. Eclipse Foundation, 7. Oktober 2022. Zugegriffen: 9. Oktober 2022. [Online]. Verfügbar unter: https://github.com/eclipse/capella/blob/cf2f9445816a05f5c08aaf162bf0ac494f1465e5/doc/plugins/org.polarsys.capella.ui.doc/html/Installation%20Guide/How%20to%20install%20Capella%20and%20Addons.mediawiki
[54] „Capella MBSE Tool - Features“. https://www.eclipse.org/capella/features.html (zugegriffen 11. Oktober 2022).
[55] „MBSE Tool - Team for Capella - Obeo - Obeo“. https://www.obeosoft.com/en/team-for-capella (zugegriffen 11. Oktober 2022).
[56] „DLR - Institut für Softwaretechnologie - Virtual Satellite“. https://www.dlr.de/sc/desktopdefault.aspx/tabid-5135/8645_read-8374/ (zugegriffen 7. Oktober 2022).
[57] P. M. Fischer *u. a.*, „Enabling a Conceptual Data Model and Workflow Integration Environment for Concurrent Launch Vehicle Analysis“, in *Proceedings of the International Astronautical Congress, IAC*, Bremen, Germany, Okt. 2018. Zugegriffen: 7. Oktober 2022. [Online]. Verfügbar unter: https://iafastro.directory/iac/proceedings/IAC-18/

[58] V. Schaus, „Concurrent Engineering Software Development at German Aerospace Center - Status and Outlook -“, S. 8.
[59] „Virtual Satellite 4 Core Server Manual“, *GitHub*. https://github.com/virtualsatellite/VirtualSatellite4-Core/releases/download/Release_4.15.0/VirSat_Core_Server_Manual.pdf (zugegriffen 21. Juli 2022).
[60] Y. Wang, P. Wagstrom, E. Duesterwald, und D. Redmiles, „New opportunities for extracting insights from cloud based IDEs“, in *Companion Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, Mai 2014, S. 408–411. doi: 10.1145/2591062.2591105.
[61] „Documentation for Visual Studio Code“. https://code.visualstudio.com/docs (zugegriffen 8. Oktober 2022).
[62] „Visual Studio Code - Open Source (‚Code - OSS‘)“. Microsoft, 8. Oktober 2022. Zugegriffen: 8. Oktober 2022. [Online]. Verfügbar unter: https://github.com/microsoft/vscode
[63] „Extension API“. https://code.visualstudio.com/api/index (zugegriffen 8. Oktober 2022).
[64] „Visual Studio Code Frequently Asked Questions“. https://code.visualstudio.com/docs/supporting/faq (zugegriffen 8. Oktober 2022).
[65] „VSCodium - Open Source Binaries of VSCode“. https://vscodium.com/ (zugegriffen 8. Oktober 2022).
[66] „code-server“. Coder, 8. Oktober 2022. Zugegriffen: 8. Oktober 2022. [Online]. Verfügbar unter: https://github.com/coder/code-server
[67] „OpenVSCode Server“. Gitpod, 8. Oktober 2022. Zugegriffen: 8. Oktober 2022. [Online]. Verfügbar unter: https://github.com/gitpod-io/openvscode-server
[68] „FAQ - code-server v4.7.1 docs“. https://coder.com/docs/code-server/latest/FAQ (zugegriffen 8. Oktober 2022).
[69] „Web IDE | GitLab“. https://docs.gitlab.com/ee/user/project/web_ide/ (zugegriffen 8. Oktober 2022).
[70] „Theia - Cloud and Desktop IDE Platform - Docs“. https://theia-ide.org/docs/ (zugegriffen 8. Oktober 2022).
[71] „eclipse-theia/theia“. Eclipse Theia™, 12. Oktober 2022. Zugegriffen: 12. Oktober 2022. [Online]. Verfügbar unter: https://github.com/eclipse-theia/theia/blob/807cf6d0ab0217a7742bd49cd794d591b5a8ee84/LICENSE
[72] „Theia - Cloud and Desktop IDE Platform - Project Goals“. https://theia-ide.org/docs/project_goals (zugegriffen 8. Oktober 2022).
[73] „Theia - Cloud and Desktop IDE Platform - Extensions“. https://theia-ide.org/docs/extensions (zugegriffen 8. Oktober 2022).
[74] „Theia - Cloud and Desktop IDE Platform - Architecture“. https://theia-ide.org/docs/architecture (zugegriffen 8. Oktober 2022).
[75] I. Gornev und T. Liakh, „RIDE: Theia-Based Web IDE for the Reflex Language“, in *2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM)*, Juni 2021, S. 503–506. doi: 10.1109/EDM52169.2021.9507678.
[76] „EMF.cloud“. https://www.eclipse.org/emfcloud/ (zugegriffen 8. Oktober 2022).
[77] „EMF.cloud - License“. eclipse-emfcloud, 12. September 2022. Zugegriffen: 12. Oktober 2022. [Online]. Verfügbar unter: https://github.com/eclipse-emfcloud/emfcloud/blob/20eb4acfa4632047bbb2e78b050100b3832aba3b/LICENSE
[78]. „gitpod.yml- Documentation Introduction“. https://www.gitpod.io/docs/introduction/learn-gitpod/gitpod-yaml (zugegriffen 9. Oktober 2022).
[79] „gitpod-io/gitpod“. Gitpod, 8. Oktober 2022. Zugegriffen: 8. Oktober 2022. [Online]. Verfügbar unter: https://github.com/gitpod-io/gitpod
[80]. „gitpod.yml - Reference Documentation“. https://www.gitpod.io/docs/references/gitpod-yml (zugegriffen 9. Oktober 2022).
[81] „Workspace Image“. https://www.gitpod.io/docs/configure/workspaces/workspace-image (zugegriffen 9. Oktober 2022).

[82] „IDEs & Editors“. https://www.gitpod.io/docs/references/ides-and-editors (zugegriffen 9. Oktober 2022).
[83] „Gitpod: Always ready to code.“ https://www.gitpod.io/ (zugegriffen 8. Oktober 2022).
[84] „Single-Cluster Reference Architecture“. https://www.gitpod.io/docs/configure/self-hosted/latest/reference-architecture/single-cluster-ref-arch (zugegriffen 9. Oktober 2022).
[85] „Gitpod Self-Hosted Requirements“. https://www.gitpod.io/docs/configure/self-hosted/latest/requirements (zugegriffen 9. Oktober 2022).
[86] „Install Gitpod in an Air-Gapped Network“. https://www.gitpod.io/docs/configure/self-hosted/latest/advanced/air-gap#install-gitpod-in-an-air-gapped-network (zugegriffen 9. Oktober 2022).
[87] „Introduction to Eclipse Che :: Eclipse Che Documentation“. https://www.eclipse.org/che/docs/stable/overview/introduction-to-eclipse-che/ (zugegriffen 9. Oktober 2022).
[88] „Eclipse Che Website“. https://www.eclipse.org/che/ (zugegriffen 12. Oktober 2022).
[89] „Installing Che in a restricted environment on OpenShift :: Eclipse Che Documentation“. https://www.eclipse.org/che/docs/stable/administration-guide/installing-che-in-a-restricted-environment/ (zugegriffen 12. Oktober 2022).
[90] „Che architecture :: Eclipse Che Documentation“. https://www.eclipse.org/che/docs/stable/administration-guide/architecture-overview/ (zugegriffen 9. Oktober 2022).
[91] „Devfile“. https://devfile.io/ (zugegriffen 9. Oktober 2022).
[92] „User workspaces :: Eclipse Che Documentation“. https://www.eclipse.org/che/docs/stable/administration-guide/user-workspaces/ (zugegriffen 9. Oktober 2022).
[93] R. Saini, S. Bali, und G. Mussbacher, „Towards Web Collaborative Modelling for the User Requirements Notation Using Eclipse Che and Theia IDE“, in *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, Mai 2019, S. 15–18. doi: 10.1109/MiSE.2019.00010.
[94] R. Saini und G. Mussbacher, „Towards Conflict-Free Collaborative Modelling using VS Code Extensions“, in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Okt. 2021, S. 35–44. doi: 10.1109/MODELS-C53483.2021.00013.
[95] A. Herrmann, *Grundlagen der Anforderungsanalyse: Standardkonformes Requirements Engineering*. Wiesbaden: Springer Fachmedien Wiesbaden, 2022. doi: 10.1007/978-3-658-35460-2.
[96] P. W. Szabo, *User Experience Mapping*. Packt Publishing Ltd, 2017.
[97] „Was sind Akzeptanzkriterien? - Wissen kompakt - t2informatik“, 24. März 2020. https://t2informatik.de/wissen-kompakt/akzeptanzkriterien/ (zugegriffen 13. Oktober 2022).
[98] *User Story Mapping*. Zugegriffen: 13. Oktober 2022. [Online]. Verfügbar unter: https://learning.oreilly.com/library/view/user-story-mapping/9781491904893/
[99] Mike Cohn, *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, 2004.
[100] M. Latta, „Reverse and Questionable Responses Using the Kano Method in International Surveys“, *Association of Marketing Theory and Practice Proceedings 2019*, Jan. 2019, [Online]. Verfügbar unter: https://digitalcommons.georgiasouthern.edu/amtp-proceedings_2019/30
[101] „Die Kano Methode“. https://www.eric-klopp.de/texte/die-kano-methode.php (zugegriffen 13. Oktober 2022).
[102] C. Högström, M. Rosner, und A. Gustafsson, „How to create attractive and unique customer experiences: An application of Kano’s theory of attractive quality to recreational

tourism“, *Marketing Intelligence & Planning*, Bd. 28, S. 385–402, Juni 2010, doi: 10.1108/02634501011053531.
[103] R. Potvin und J. Levenberg, „Why Google stores billions of lines of code in a single repository“, *Commun. ACM*, Bd. 59, Nr. 7, S. 78–87, Juni 2016, doi: 10.1145/2854146.
[104] P. Shanmugapriya und R. M. Suresh, „Software Architecture Evaluation Methods A Survey“, *IJCA*, Bd. 49, Nr. 16, S. 19–26, Juli 2012, doi: 10.5120/7711-1107.
[105] „Sunset Che Theia as a built-in Che editor · Issue #21771 · eclipse/che“. https://github.com/eclipse/che/issues/21771 (zugegriffen 8. November 2022).
[106] J. Franca, A. Junior, und M. Soares, „Architecture-Driven Development of an Electronic Health Record Considering the SOAQM Quality Model“, *SN Computer Science*, Bd. 1, Apr. 2020, doi: 10.1007/s42979-020-00150-x.

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Hausarbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Datum, Unterschrift