INTERLIS Modell Composer (IMC)

Anforderungsspezifikation

Version 21 vom 5. Mai 2026

Auftraggeber:	Geostandards.ch
Ersteller:	Prelogic AG, Winterthur
Status:	Abgenommen durch Steuerungsausschuss
Klassifizierung:	öffentlich

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Zum Dokument

1.1 Zweck und Abgrenzung

Dieses Dokument beschreibt die fachlichen Anforderungen und technischen Rahmenbedingungen an ein Werkzeug für die Erstellung, Dokumentation und Änderung von INTERLIS Modellen.

Das Dokument beinhaltet eine Einführung in die Thematik, die Beschreibung der Ausgangslage und der Zielsetzungen, die Abgrenzung des Untersuchungsbereichs, eine Analyse der Ist-Situation, die Beschreibung der Soll-Situation und die Anforderungen, die sich daraus an eine mögliche Lösung ergeben.

1.2 Vollständigkeit und Detaillierungsgrad

Mit dem vorliegenden Dokument wird bezüglich Vollständigkeit und Detaillierungsgrad angestrebt, die fachlichen Anforderungen an das Werkzeug so weit zu definieren, dass eine genaue und verbindliche Aufwandschätzung für die Realisierung des Werkzeugs vorgenommen werden kann.

1.3 Referenzen

[1] KSTEC GmbH (2024): Konzept – INTERLIS-Workbench Funktionalität: Phase 1 Bedarfsabklärung der Funktionalität. [1]
Auftraggeber: Strategie Geoinformation Schweiz / GeoStandards.ch
Version 1.0, November 2024

1.4 Änderungskontrolle

Version	Datum	Beschreibung
1 – 8	22.10. – 07.11. 2025	Arbeitsversionen
9	10.11.2025	Arbeitsversion, Überarbeitung Kapitel 6.8 Versionsverwaltung/ Kollaboration, Version für Review Kernteam
10 – 11	10.11. – 15.11.2025	Arbeitsversionen, Einarbeitung Befunde aus Review Kernteam
12 – 13	28.11.2025	Ergänzung Kapitel Architektur, Befunde aus Walkthrough vom 28.11.2025
14 – 15	04.12.2025	Einarbeitung Reviewresultat
16 – 18	16.12.2025 – 23.01.2026	Einarbeitung finale Änderungen
19	25.02.2026	Einarbeitung Befunde Reviewer
20	18.03.2026	Grammatische Korrekturen, kleinere Präzisierungen.

1.5 Weitere Hinweise

Aus Gründen der Lesbarkeit und Übersichtlichkeit wird in den Zeichnungen, Darstellungen und Texten dieses Dokuments teilweise auf eine geschlechterspezifische Differenzierung verzichtet. Sämtliche Personenbezeichnungen und Darstellungen gelten gleichermassen für alle Geschlechter. Die gewählte Form umfasst immer alle Menschen – unabhängig von ihrem Geschlecht oder ihrer geschlechtlichen Identität.

[1] Online verfügbar unter: https://www.interlis.ch/download_file/view/05e2024b-27d3-4cb5-9f4d-e5901e74f2c6/782 (Zugriff am: 22.09.2025).

2 Glossar

Dieses Kapitel stellt keine Anforderungen an das Werkzeug dar. Es dient lediglich der Übersicht und der Einführung.

2.1 Begriffe/ Abkürzungen

Die nachfolgende Tabelle führt im Dokument verwendete Begriffe und Abkürzungen auf.

Abkürzung	Bedeutung
FIG	Fachinformationsgemeinschaft
GeoIG	Geoinformationsgesetz
GeoIV	Geoinformationsverordnung
IDE	Integrated Development Environment, auf Deutsch Integrierte Entwicklungsumgebung
ili2c[2]	INTERLIS Compiler (Eine bestehende Software, welche INTERLIS Datenmodelle auf korrekte Syntax prüft)
ilivalidator[3]	Mit dem ilivalidator lassen sich Daten im INTERLIS Transferformat (*.itf/*.xtf/ xml) darauf überprüfen, ob sie zum zugehörigen Modell (*.ili) konform sind.
IMC	INTERLIS Model Composer ist die Bezeichnung des Werkzeugs, das in dieser Anforderungsspezifikation beschrieben wird.
NGDI	Nationale Geodateninfrastruktur
Prosatext	Als Prosatext werden alle beschreibenden Texte, Freitexte und Grafiken bezeichnet, die einem INTERLIS-Modell beigefügt sind, jedoch keinen Bestandteil des formalen INTERLIS-Modells selbst bilden. Prosatexte ergänzen das Modell um menschenlesbare Erläuterungen, die weder in der INTERLIS-Syntax kodiert noch durch einen Compiler verarbeitbar sind. Sie können jedoch Bestandteil der Modelldokumentation sein.
SSOT	Single-Source-of-Truth bezeichnet im IT-Kontext das Prinzip, dass es für eine bestimmte Information genau eine zentrale, verbindliche Datenquelle gibt, auf die alle Systeme, Prozesse und Personen zugreifen.
VCS	Version Control System (Ein System, das Änderungen an Dateien (meist Quellcode) verfolgt, Versionen verwaltet und ermöglicht, zu früheren Ständen zurückzukehren)
Externer Katalog[4]	Externe Kataloge sind dynamische Wertelisten, deren Einträge aus der starren Datenmodelldefinition herausgelöst sind. Die Struktur des Katalogs wird im Datenmodell definiert, die Katalogdaten sind die zugehörigen INTERLIS 2-XML-Transferdaten (XTF, meist als .xml-Datei gespeichert), die typischerweise zusammen mit dem Datenmodell auf einem Online-Repository publiziert werden. Aus Sicht des Datenmodells kann ein Katalog auch als «generische Aufzählung» erachtet werden.
xls2xtf[5]	xls2xtf liest Codelisten aus einer Excel-Datei (.xls) und wandelt sie in eine Transferdatei im Format INTERLIS 2 (.xtf) um.

[2] Vgl. https://www.interlis.ch/downloads/ili2c, Zugriff 05.12.2025

[3] Vgl. https://www.interlis.ch/downloads/ilivalidator, Zugriff 05.12.2025

[4] Vgl. https://www.geo.admin.ch/dam/de/sd-web/528vKWlohxZJ/Empfehlungen%20Kataloge.pdf, Zugriff am 05.12.2025

[5] Vgl. https://github.com/claeis/xls2xtf, Zugriff am 05.12.2025

3 Einführung

3.1 Ausgangslage

Gestützt auf das Geoinformationsgesetz (GeoIG) und die Geoinformationsverordnung (GeoIV) hat das Bundesamt für Landestopografie (swisstopo) INTERLIS als verbindliche Beschreibungssprache für Geodatenmodelle festgelegt. Die modellbasierte Methode mit INTERLIS und den dazugehörigen Werkzeugen ist in der Schweiz etabliert und weit verbreitet im Einsatz.

In den letzten Jahren hat sich rund um INTERLIS und die nationale Geodateninfrastruktur (NGDI) eine hohe Dynamik entwickelt. Gleichzeitig wurde die Weiterentwicklung der Standards und Werkzeuge als unkoordiniert, wenig nutzerorientiert und intransparent wahrgenommen.

Zur gezielten Steuerung dieser Entwicklungen wurde die Organisation GeoStandards.ch ins Leben gerufen. Sie hat den Auftrag, die Standardisierung im Geoinformationsbereich nachhaltig und bedarfsorientiert zu gestalten, sowie die Weiterentwicklung von Werkzeugen und Bildungsangeboten sicherzustellen.

Im Rahmen dieser Strategie wurde das Projekt «Initialisierung INTERLIS-Suite» gestartet, um die künftige Werkzeugentwicklung zu koordinieren. Ein zentrales Ergebnis daraus ist die Bedarfsabklärung (vgl. [1]) für ein neues INTERLIS-Werkzeug, das unter dem Arbeitstitel «INTERLIS-Workbench» geführt wird. Diese Bedarfsabklärung dient als Grundlage für die Planung konkreter Software-Entwicklungsprojekte und bezieht gezielt Rückmeldungen von Anwenderinnen und Anwendern mit ein.

3.2 Ziele

3.2.1 Gesamtprojekt

Das übergeordnete Ziel des Projekts ist die schrittweise Entwicklung eines umfassenden, modernen und nutzerzentrierten INTERLIS-Werkzeugs – des sogenannten INTERLIS Model Composer (IMC). Dieses soll die bestehenden Werkzeuge konsolidieren, erweitern und nachhaltig verbessern. Die Lösung soll langfristig die Bedürfnisse von Modellierenden und Datenverantwortlichen abdecken.

Der Ausbau erfolgt etappenweise und nutzungsorientiert, wobei jede Etappe so gestaltet ist, dass sie auf klar identifizierten Bedürfnissen basiert und zu einer konsistenten, leistungsfähigen Gesamtlösung beiträgt.

3.2.2 Anforderungsanalyse

Ziel der Anforderungsanalyse ist die systematische Erhebung und Analyse der Anforderungen (Requirements Engineering) als Grundlage für die initiale Bereitstellung des IMC und dessen etappierter Ausbau.

Diese Phase umfasst:

- Die konkrete Definition und Spezifikation des Scopes der ersten Ausbaustufe mit Blick auf Funktionalität, Architektur, Integration und Zielgruppen.
- Die Erarbeitung eines vollständigen Anforderungskatalogs der ersten Ausbaustufe durch Einbezug relevanter Nutzergruppen.
- Die Identifikation und Priorisierung möglicher Ausbaustufen im Sinne eines phasenweisen Entwicklungsplans.

Das Ergebnis dieser Initialphase ist das vorliegende Dokument, welches als belastbare und abgestimmte Entscheidungsgrundlage für die Realisierung der ersten Etappe und dem strukturierten weiteren Ausbau des IMC dient.

4 Abgrenzung

4.1 Workshop zur Orientierung und Priorisierung der Anforderungen

Ausgehend von der bestehenden Bedarfsabklärung (vgl. [1]) wurde ein Workshop durchgeführt mit verschiedenen Personen, die bereits an der Bedarfsabklärung beteiligt waren. Ziel des Workshops war es, Orientierung für die künftige Stossrichtung der Applikation zu schaffen.

Dabei wurde ein gemeinsames Verständnis der primären Nutzer:innen und ihrer wichtigsten Anforderungen entwickelt und eine erste Priorisierung des bestehenden Anforderungskatalogs vorgenommen.

Folgende Personengruppen wurden definiert:

- FIG Leiter:in
- Modellierer:in Fachspezialist:in
- Modellierer:in Entwickler:in
- (INTERLIS-) Anfänger:in/ Einsteiger:in
- Fachperson im Amt
- Datenproduzent:in

Für jede dieser Gruppen wurden die fünf wichtigsten Anforderungen erarbeitet und im Anschluss priorisiert. Die resultierenden Anforderungen wurden nach der Grösse der jeweiligen Nutzergruppe sowie nach ihrer Priorität gewichtet, um die zentralen Bedürfnisse zu identifizieren und daraus den inhaltlichen Fokus sowie die Stossrichtung für eine erste Umsetzungsetappe abzuleiten.

Dabei war die Gewichtung der Anforderungen nicht als wissenschaftlich fundierte Ableitung konzipiert, sondern diente ausschliesslich dazu, eine Orientierung hinsichtlich der inhaltlichen Schwerpunkte und der strategischen Stossrichtung zu gewinnen.

Die nachfolgende Grafik zeigt die gewichteten Anforderungen, gruppiert nach den Kapiteln aus der Bedarfsabklärung. Am höchsten bewertet wurden die Anforderungen zur Modelldokumentation sowie zu editierbaren Sichten auf das Datenmodell, gefolgt von Erstellen, Editieren und Übersetzen von Katalogen. Anforderungen wie Darstellungskatalog, Modellprüfung oder Unterstützung von Validierungsmodellen wurden hingegen nur gering gewichtet.

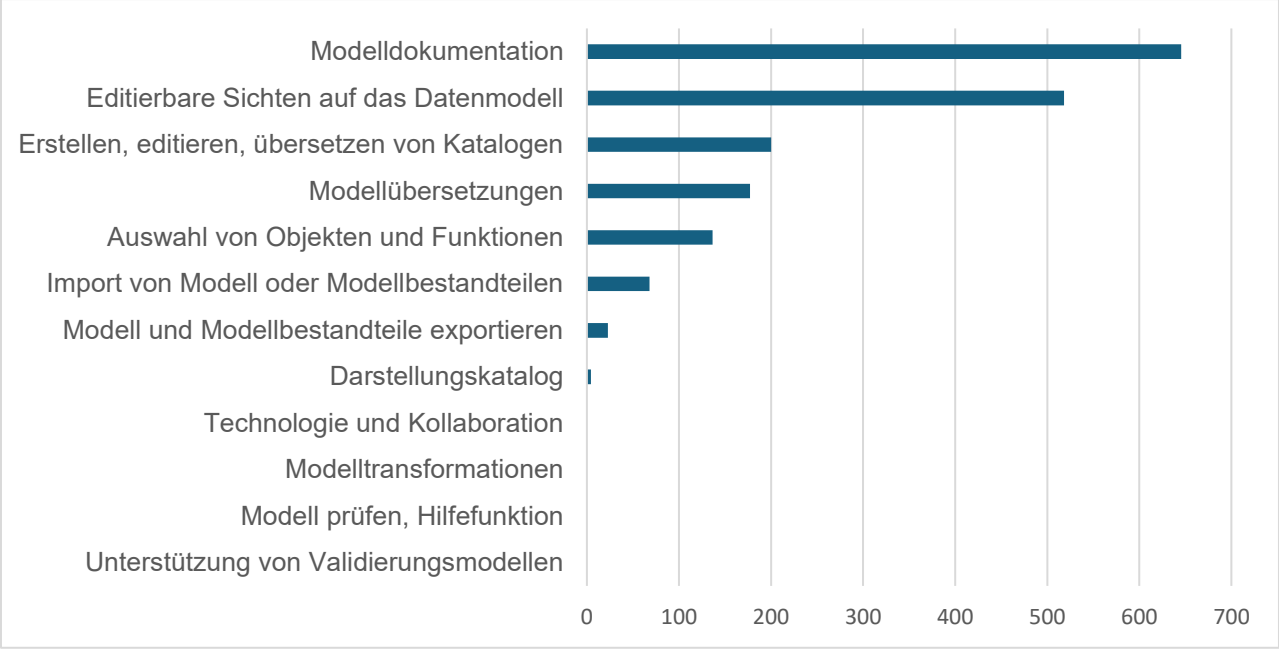

Abbildung 1. Gewichtete und gruppierte Anforderungen aus der Bedarfsanalyse

4.2 Entscheid zur Fokussierung der ersten Umsetzungsetappe

Auf Basis der gewichteten Anforderungen und der daraus abgeleiteten Stossrichtung wurde entschieden, dass in einer ersten Umsetzungsetappe der Schwerpunkt auf der Unterstützung der Arbeit der Fachinformationsgemeinschaften (FIG) liegt. Können die FIG effizienter arbeiten, dann profitieren alle Nutzenden davon und z.B. die Hürde und Kosten für eine Dokumentation auch in einer zweiten oder dritte Landessprache sinken. Dadurch profitieren alle Nutzergruppen und Anwender:innen der erarbeiteten Modelle, auch die Kantone, Gemeinden und Private als Auftraggeber von einer klaren und einfach verfügbaren und integrierbaren Dokumentation.

Ziel ist die Entwicklung des IMC, ähnlich einer integrierten Entwicklungsumgebung (IDE), welcher primär die Fachinformationsgemeinschaften (FIG) bei der Erstellung, Pflege, Prüfung und Dokumentation von INTERLIS-Datenmodellen unterstützt.

Der IMC soll den gesamten Arbeitsprozess – von der Modellierung über die Validierung bis zur Bereitstellung von Katalogen und Dokumentationen – in einer einheitlichen, modular erweiterbaren Umgebung ermöglichen.

Die Lösung soll auf einer modernen, webbasierten IDE-Architektur basieren, welche die parallele Entwicklung, Erweiterbarkeit und Integration externer und teilweise bereits bestehender Werkzeuge unterstützt.

4.3 Rahmenbedingungen

ID	Beschreibung
R1	Bestehende INTERLIS-Werkzeuge (z. B. ili2c) sollen – soweit möglich und sinnvoll – eingebunden werden.
R2	Der IMC muss im Browser (Priorität 1) und plattformunabhängig als Desktopanwendung (Priorität 2) nutzbar sein.
R3	Es werden ausschliesslich Open-Source-Software oder quelloffene Komponenten mit klar definierten Lizenzen eingesetzt. Wo immer möglich und sinnvoll, sollen für die Realisierung der geforderten Funktionalitäten bestehende und weit verbreitete Software-Komponenten eingesetzt werden.
R4	Die Benutzeroberfläche ist in den Sprachen Englisch, Deutsch, Französisch und Italienisch bereitzustellen. Mehrsprachigkeit erfolgt mittels Lokalisierungs-Dateien.
R5	Die Lösung muss auf einer bestehenden und weit verbreiteten Plattform modular aufgebaut sein und eine spätere Erweiterung ohne grundlegende Architekturänderungen ermöglichen.

5 Nutzergruppen und Nutzungsszenarien

5.1 Nutzergruppen

Nachfolgend werden die Nutzergruppen aufgeführt, die im Workshop identifiziert und in Form von Personas beschrieben wurden. Wie in Kapitel 4.2 beschrieben, wird im weiteren Verlauf aber v.a. auf die FIG-Nutzergruppen fokussiert.

5.1.1 N1 – Modellierer:in Fachspezialist:in

Rolle/ Hintergrund	Fachperson mit vertieftem Wissen zum Modellinhalt (Kataloge, Wertebereiche, Objektstrukturen)
Teil einer FIG	Ja
Nutzung IMC	Ja
Ziele	Fachinhalte effizient erfassen und in das Modell überführen. Zusammenhänge visualisieren.
Herausforderungen	Fehlende Werkzeuge für einfache, fachorientierte Modellierung
Technikaffinität	Mittel, offen für neue Tools

5.1.2 N2 – Modellierer:in Entwickler:in

Rolle/ Hintergrund	Spezialist:in für semantische Modellierung in INTERLIS und UML
Teil einer FIG	Ja
Nutzung IMC	Ja
Ziele	Technisch korrekte und semantisch konsistente Umsetzung von Fachmodellen.
Herausforderungen	Bestehende UML-Editoren sind unvollständig und umständlich.
Technikaffinität	Hoch (Programmier- und Modellierungserfahrung)

5.1.3 N3 – FIG-Leiter:in

Rolle/ Hintergrund	Verantwortlich für Abstimmung, Qualitätssicherung und Kommunikation innerhalb der FIG
Teil einer FIG	Ja
Nutzung IMC	Ja
Ziele	Koordination von Anforderungen, Dokumentation von Prozessen und Entscheidungen.
Herausforderungen	Sicherstellung eines verständlichen, konsistenten und wartbaren Modells
Technikaffinität	Mittel

5.1.4 N4 – Anfänger:in/ Einsteiger:in

Rolle/ Hintergrund	Fachperson mit Kenntnissen in Geoinformation, jedoch begrenzter INTERLIS-Erfahrung
Teil einer FIG	Ja
Nutzung IMC	Ja
Ziele	Erstellen einfacher Datenmodelle mit UML-/INTERLIS-Unterstützung.
Herausforderungen	Komplexe Handbücher, hohe Einstiegshürden.
Technikaffinität	Mittel bis gut

5.1.5 N5 – Fachperson im Amt (Datensatzverantwortliche:r)

Rolle/ Hintergrund	Expertenwissen über Domäne aber nicht zwingend Modellierungs-Know-how. Bringt fachlichen Informationsgehalt. Zuständige Fachperson im Amt, welche fachlich für den Datensatz zuständig ist. Nutzt Modell-Dokumentation (Erfassungsrichtlinie) Fall 1: für eigene Datenbearbeitung Fall 2: für Auftragsvergabe an Dritte
Teil einer FIG	Ja
Nutzung IMC	Nein
Ziele	Zugriff auf aktuelle und historische Modell-Dokumentationen
Herausforderungen	Unklare Ablage und Versionierung von Dokumentationen
Technikaffinität	Gut, GIS- und Datenmanagement-Kenntnisse

5.1.6 N6 – Datenproduzent:in

Rolle/ Hintergrund	Erfassung und Pflege von Daten gemäss Modell
Teil einer FIG	Nein
Nutzung IMC	Nein
Ziele	Modellkonforme Datenerfassung; Verständnis der Strukturen durch visuelle Darstellung
Herausforderungen	Komplexität der Datenmodelle erschwert Verständnis; fehlende Visualisierung
Technikaffinität	Hoch, praxisorientiert

5.2 Nutzungsszenarien

5.2.1 S1 – Fachorientierte Modellierung

Szenario	Fachspezialist:innen erfassen und pflegen Modellinhalte (Klassen, Attribute, Wertebereiche) in einer benutzerfreundlichen, visuell unterstützten Oberfläche.
Ziel	Einfache Übertragung von Fachwissen ins Datenmodell
Beteiligte Nutzergruppe	N1 – Modellierer:in Fachspezialist:in N4 – Anfänger:in Einsteiger:in

5.2.2 S2 – Modellierung komplexer INTERLIS-Modelle

Szenario	Modellierungsexpert:innen erstellen und pflegen INTERLIS-Modelle, validieren Syntax und Struktur, und führen Versionsvergleiche durch.
Ziel	Effiziente, technisch saubere Modellierung mit direkter Rückmeldung und Validierung
Beteiligte Nutzergruppe	N2 – Modellierer:in Entwickler:in

5.2.3 S3 – Modell-Dokumentationen

Szenario	Fachpersonen in Ämtern greifen auf aktuelle und historische Modellversionen zu, vergleichen Änderungen und nutzen diese zur Datenpflege oder Auftragsvergabe.
Ziel	Einheitliche, versionierte Dokumentationsquelle
Beteiligte Nutzergruppe	N3 – FIG-Leiter:in N5 – Fachperson im Amt

	N6 – Datenproduzent:in

5.2.4 S4 – Datenerfassung und -prüfung

Szenario	Datenproduzent:innen verwenden Modelle zur Datenvalidierung und -erfassung. Sie benötigen einfache Visualisierungen der Modellstrukturen, um Datensätze korrekt zu erfassen. Abgrenzung: Die Import-/ Export-Funktionen für xtf-Dateien ist primär für den Transfer von «Externen Katalogen» vorgesehen (vgl. Kapitel 6.13) und beinhaltet auch deren Validierung. Die Validierungs-Funktion wird nicht ausgelegt für die Validierung umfangreicher Datensätze von «Datenproduzent:innen». Somit beschränkt sich dieses Szenario in den nachstehenden Anforderungen auf die Visualisierung der Modellstrukturen.
Ziel	Sicherstellung der Modellkonformität von Daten
Beteiligte Nutzergruppe	N1 – Modellierer:in Fachspezialist:in N2 – Modellierer:in Entwickler:in N5 – Fachperson im Amt (Datensatzverantwortliche:r) N6 – Datenproduzent:in

5.2.5 S5 – Koordination und Qualitätssicherung

Szenario	FIG-Leitende überwachen Modellfortschritte, steuern Prioritäten, dokumentieren Entscheidungen und stellen Konsistenz sicher.
Ziel	Koordinierte, nachvollziehbare Modellentwicklung
Beteiligte Nutzergruppe	N1 – Modellierer:in Fachspezialist:in N2 – Modellierer:in/ Entwickler:in N3 – FIG-Leiter:in

5.2.6 S6 – Einstieg und Schulung

Szenario	Einsteiger:innen oder GIS-Fachpersonen nutzen Vorlagen, Assistenten und Validierungen, um erste INTERLIS-Modelle zu erstellen. Abgrenzung: Für die Unterstützung von Einsteiger:innen sollen ausserhalb des IMC Hilfsmittel zur Verfügung gestellt werden (z. B. E-Learnings, Benutzeranleitungen). Es werden in einem ersten Ausbauschritt keine Assistenzfunktionen (z. B. Assistent für die Modellerstellung) innerhalb des IMC zur Verfügung gestellt. Dieses Szenario wird in den nachstehenden Anforderungen auf die Visualisierung der Modellstrukturen und Bearbeitung in Form von Objektkatalogen beschränkt.
Ziel	Reduktion der Einstiegshürde und Förderung der Selbstständigkeit
Beteiligte Nutzergruppe	N1 – Modellierer:in Fachspezialist:in N4 – Anfänger:in Einsteiger:in

6 Funktionale Anforderungen

Das Kapitel beschreibt die funktionalen Anforderungen an die erforderliche Softwareunterstützung auf Basis der in Kapitel 5 aufgeführten Nutzungsszenarien und den am Workshop ausgewählten Anforderungen (vgl. Kapitel 4.1) aus der Bedarfsanalyse [1]. Weitere Funktionen wurden ergänzt, damit der IMC als Gesamtlösung funktioniert und die definierten Rahmenbedingungen (vgl. Kapitel 4.3) erfüllt.

6.1 Konzeptioneller Aufbau und Begriffe

Dieses Kapitel definiert den konzeptionellen Aufbau der Applikation, damit für die nachfolgende Beschreibung der Anforderungen eine nachvollziehbare Struktur geschaffen werden kann und Begriffe zentral definiert sind.

Die Grafik zeigt den konzeptionellen Aufbau im Sinne einer Software-Architektur des IMC als mehrschichtige Modellierungs- und Verarbeitungskomponente.

Die Architektur ist in vier logisch getrennten Layer gegliedert:

- Presentation Layer: Benutzeroberflächen zur Datenmodellbearbeitung
- Core Model Layer: zentrale, interne Datenrepräsentation
- Service / Processing Layer: Validierungs- und Generierungsservices
- Persistence Layer: Speicherung, Wiederverwendung und Versionierung

Jede Komponente im Presentation-Layer interagiert ausschliesslich über das zentrale Core Model, wodurch Konsistenz und Single-Source-of-Truth gewährleistet werden. Über das Core Model erfolgt die Synchronisierung aller Komponenten im Presentation Layer.

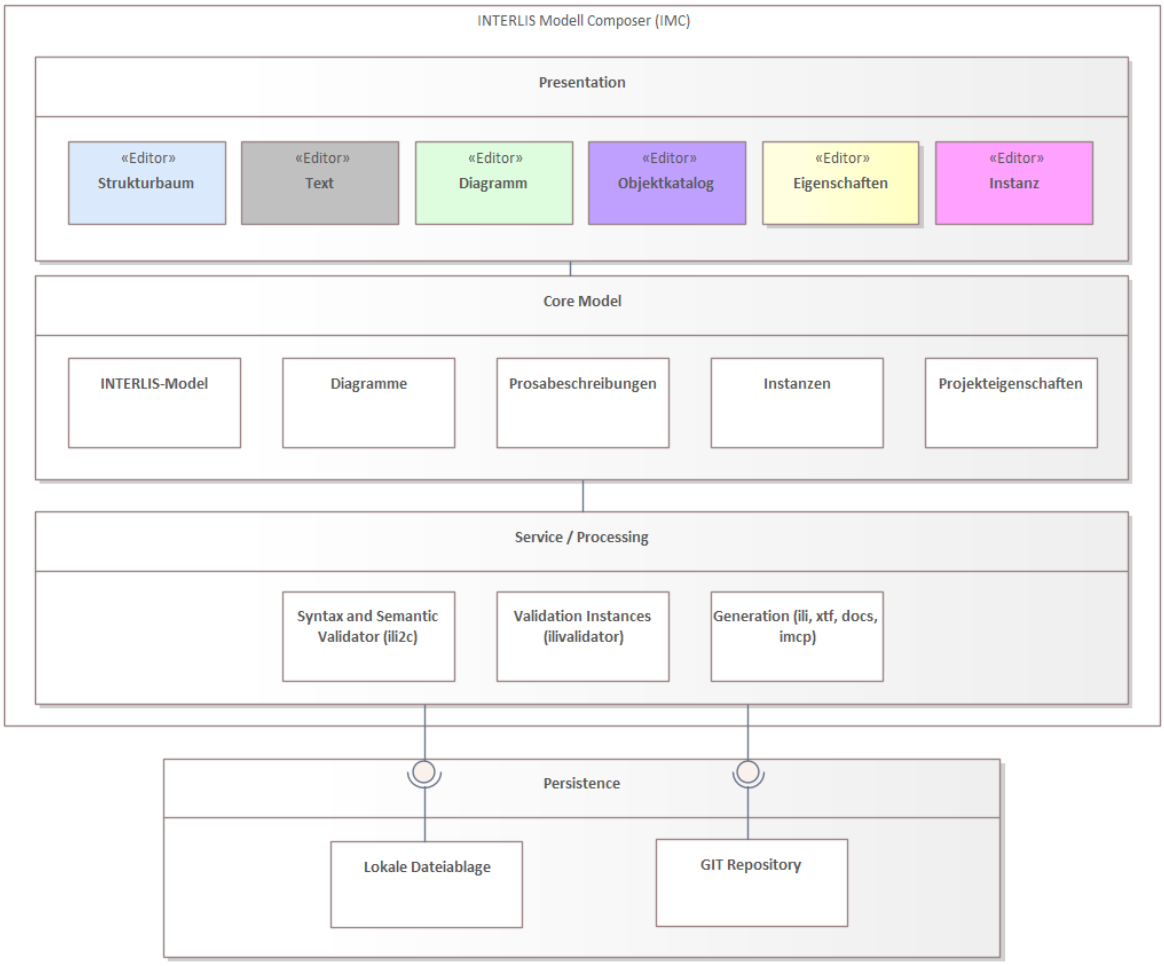

Abbildung 2. Software-Architektur IMC

6.1.1 Presentation Layer

Begriff	Definition (IT-fachlich)
Strukturbaum-Editor	Ansicht und Bearbeitung der Struktur des Hierarchiebaums mit allen Modell- und Projekt-Artefakten.
Text-Editor	Editor zur direkten textuellen Bearbeitung des INTERLIS-Codes.
Diagramm-Editor	Visueller, modellbasierter Editor zur grafischen Darstellung und Bearbeitung der Modellstruktur. Vergleichbar mit UML-Class-Diagrammbearbeitung.
Objektkatalog-Editor	Eingabekomponente zur Erstellung und Bearbeitung von Klassen und Attributen.
Eigenschaften-Editor	Editor zur Bearbeitung der Eigenschaften der verschiedenen Modell-Elemente (z. B. Klassen, Attribute, Diagramme).
Instanz-Editor	Eingabekomponente zur Erfassung von (Objekt)-Instanzen auf Basis einer Klassendefinition, die z. B. für die Erstellung von «externen Katalogen» genutzt werden kann.

6.1.2 Core Model Layer

Begriff	Definition (IT-fachlich)
Core Model	Interne konsolidierte Modellrepräsentation als logische Datenstruktur, die sämtliche Editor-Änderungen synchron integriert.
INTERLIS-Model	Formal definiertes INTERLIS 2-Modell. Beinhaltet die modellierten INTERLIS-Elemente.
Diagramm	Visualisierte Repräsentation eines Ganzen oder eines Teils eines INTERLIS-Modells als Klassendiagramm.
Prosabeschreibungen	Lesbare Fachdokumentation des Modells, zur semantischen Erläuterung und fachlichen Nachvollziehbarkeit.
Instanzen	Instanzen einer Klasse, primär der Inhalt von «externen Katalogen».
Projekteigenschaften	Projektspezifische Informationen (z. B. Struktur Hierarchiebaum, benutzerspezifische Projekteinstellungen).

6.1.3 Service/ Processing Layer

Begriff	Definition (IT-fachlich)
Syntax and Semantic Validator (ili2c)	Komponente zur Analyse der textbasierten Modellspezifikation, prüft formale Syntax sowie logische, semantische und regelbasierte Modellkonsistenz nach dem INTERLIS-Metamodell. Basis ist der bestehende INTERLIS-Compiler ili2c.
Validation Instance (ilivalidator)	Komponente zur Validierung von Instanzen gegenüber der Modelldefinition. Im Wesentlichen zur Prüfung beim Import von xtf/ xml-Dateien für «externe Kataloge». Basis ist der bestehende ilivalidator.
Generation (ili, xtf/ xml, docs, imcp)	Service zur Erstellung unterschiedlicher Artefakte: .ili (Modelldefinition), .xtf/ xml (Transfer-/Datenaustauschformat), Dokumentation (HTML/PDF), .imcp (Tool-eigenes Projektdateiformat zur vollständigen Speicherung des Modellierungsprojekts). Ggf. Nutzung von Funktionalität aus xls2xtf.

6.1.4 Persistance Layer

Begriff	Definition (IT-fachlich)
Lokale Dateiablage	Dateisystembasierte Ablage zur Persistierung von Modellartefakten ohne systemgestützte Versionskontrolle.
GIT Repository	Dezentralisiertes Versionsverwaltungssystem zur revisionssicheren Speicherung von Modellartefakten.

6.2 Übersicht

Nr.	Szenario	Bezeichnung	Beschreibung
F1	S1 – S6	Bearbeitung von INTERLIS-Projekten	Werkzeug zur Unterstützung der Modellierung in INTERLIS-Projekten. Das Werkzeug verfügt über grundlegende Funktionalitäten zur Erstellung und Bearbeitung von INTERLIS-Modellen und zum Import und Export von Modellen.
F2	S1, S2, S6	Synchronisierung von Modelländerungen zwischen allen Ansichten	Synchronisierung von Änderungen im Modell und allen zur Verfügung stehenden Ansichten / Editoren.
F3	S1 – S6	Datei-Import und -Export	Import und -Export von INTERLIS-Modellen (.ili) und Transferdateien (.xtf/ xml). Validierung Transferdateien mittels ilivalidator.
F4	S1 – S6	Flexibler und anpassbarer Arbeitsbereich	Grundlegend anpassbarer Arbeitsbereich der Benutzeroberfläche, der ein effizientes Arbeiten unterstützt.
F5	S1, S2	Text-Editor mit Syntaxprüfung	Text-Editor zur Bearbeitung von INTERLIS-Code. Unterstützung der INTERLIS-Syntax mit farblicher Hervorhebung, Autovervollständigung und direkter Validierung.
F6	S1, S2	Struktur- und Semantikprüfung	Prüfung von Modellbeziehungen, Referenzen und Multiplizitäten gemäss INTERLIS-Regeln.
F7	S1, S2	Strukturbaum-Editor	Hierarchische Darstellung der Modellstruktur (Modelle, Themen, Klassen, Beziehungen, Attribute) in einem interaktiven Strukturbaum. Unterstützung von Drag-and-drop zur Neuordnung, Kontextmenüs zur Bearbeitung von Elementen sowie direkter Navigation zwischen Baumansicht und anderen Editoren.
F8	S1, S3, S6	Diagramm-Editor	Grafische Darstellung von Klassen, Beziehungen und Attributen (ähnlich UML).
F9	S1, S6	Objektkatalog-Editor	Bearbeitung von Objektkatalogen in tabellarischer Form. Definition von Klassen und Attributen (Bezeichnung, Multiplizität, Typ, Wertebereiche, Beschreibung).
F10	S1 – S6	Eigenschaften-Editor	Bearbeitung der generellen und spezifischen Eigenschaften der Modell-Elemente, inkl. Übersetzungen.
F11	S1, S6	Instanz-Editor	Bearbeitung von Katalogeinträgen von «externen Katalogen» in tabellarischer Form.
F12	S1, S3, S5	Fachliche Prosabeschreibungen	Erfassung von Prosatext für die generelle fachliche Beschreibung eines INTERLIS-Modells oder spezifische Beschreibung von Elementen wie Klassen und Attribute.
F13	S1, S2, S5	Versionsmanagement und Kollaboration	Vergleich und Verwaltung unterschiedlicher Modellstände (Diff, Merge, Historie). Markieren und Kommentieren von Modelländerungen.
F14	S3	Dokumentationsgenerierung	Automatische Erstellung von Modelldokumentationen.
F15	S1–S6	Suche, Filterung und Navigation	Volltextsuche in Modellen, Filterung nach Klassen, Domains, Beziehungen.

6.3 Bearbeitung von INTERLIS-Projekten

Identifikation: F1

6.3.1 User Story

Als Modellierer:in möchte ich neue INTERLIS-Modellierungsprojekte anlegen, speichern und wieder öffnen können – sowohl lokal als auch aus einem Git-Repository –,

damit ich meine Modellierungsarbeit strukturiert, nachvollziehbar und kontrolliert über mehrere Versionen durchführen kann und das Projekt mit anderen Beteiligten austauschen kann.

6.3.2 Ziel und Beschreibung

Der IMC verwaltet Modellierungsprojekte in einer eigenen Projektdatei. Für dieses Konzept wird eine Projektdatei mit der Endung .imcp (INTERLIS Model Composer Project) angenommen.

Eine Projektdatenstruktur umfasst alle für die Modellierung notwendigen Informationen:

- INTERLIS-Modelle (Textrepräsentation)
- Diagramme
- Instanzen (z. B. für «externe Kataloge»)
- Prosabeschreibungen
- Metadaten (z. B. Editorzustände, Darstellungsoptionen, Layouts)

Ein Projekt kann entweder

- im lokalen Dateisystem gespeichert/geladen werden oder
- aus einem Git-Repository geöffnet und wieder dorthin synchronisiert werden.

Die Projektverwaltung (z. B. über einen Menüpunkt „Project“ erreichbar) stellt folgende Grundfunktionen zur Verfügung:

- New Project…
- Open Project from file…
- Open Project from repository…

Beim Speichern werden alle zum Projekt gehörenden Daten in einer einzigen .imcp-Datei abgelegt, sodass der Projektzustand vollständig und konsistent bleibt.

6.3.3 Akzeptanzkriterien

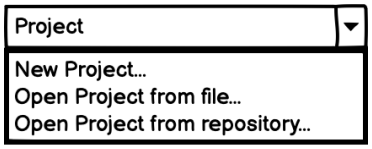

Abbildung 3. Projektverwaltung (Skizze)

Projekt erstellen

F1-1 Die Benutzer:in kann über eine Projektverwaltung ein neues Projekt anlegen.

F1-2 Ein neues Projekt enthält eine leere Struktur (keine Modellelemente).

F1-3 Beim Anlegen wird die Benutzer:in aufgefordert, Speicherort und Dateinamen zu bestimmen.

F1-4 Direkt nach dem Anlegen ist das Projekt gespeichert und einsatzbereit.

F1-5 Das gerade erstellte Projekt erscheint im Hauptfenster und im Titel-/Pfadbereich der Anwendung.

Projektdateiformat

F1-6 Alle projektbezogenen Daten werden in einer einzigen Datei gespeichert. Z. B. Datei mit Endung .imcp.

F1-7 Die Projektdatei ist portabel und kann ohne Zusatzdateien auf anderen Systemen geöffnet werden.

Projekt speichern

F1-8 Ein Projekt kann jederzeit gespeichert werden über die Projektverwaltung.

F1-9 Ein Projekt muss auch gespeichert werden können, wenn das zugrunde liegende INTERLIS-Modell syntaktische Fehler enthält.

F1-10 „Save“ überschreibt die bestehende .imcp-Datei.

F1-11 „Save As…“ fragt nach einem neuen Speicherort und Dateinamen.

F1-12 Der IMC bestätigt erfolgreiches Speichern visuell (z. B. Statusmeldung oder Indikator im Tab).

F1-13 Wenn das Projekt ungespeicherte Änderungen hat, wird dies im UI sichtbar angezeigt (z. B. Sternsymbol im Projektnamen).

Projekt laden – lokale Datei

F1-14 Projekte können über die Projektverwaltung geladen werden.

F1-15 Der Dateidialog zeigt initial nur .imcp-Dateien an.

F1-16 Beim Öffnen werden alle Projektkomponenten wiederhergestellt: Strukturbaum, Editoren, Editorpositionen, Layouts, Diagramme.

F1-17 Fehler in der .imcp-Datei (z. B. korrupt) werden verständlich gemeldet.

F1-18 Beim Laden wird die aktuelle Arbeitsumgebung geleert.

Projekt laden – Git-Repository

F1-19 Projekte können über die Projektverwaltung aus einem Git-Repository geöffnet werden.

F1-20 Unterstützt werden SSH- und HTTPS-basierte Repositories.

F1-21 Die Benutzer:in kann Branch, Tag oder Commit auswählen.

F1-22 Der IMC klont (oder aktualisiert) das Repository lokal und öffnet daraus die relevante .imcp-Datei.

F1-23 Fehler beim Klonen oder beim Zugriff auf das Repository (Authentifizierung, Netzwerkfehler) werden angezeigt.

F1-24 Änderungen am Projekt können wieder per Git (Commit/Push) zurückgeschrieben werden, sofern das Repository Schreibzugriff erlaubt.

Fehler- und Konfliktverhalten

F1-25 Bei fehlenden Schreibrechten wird ein klarer Hinweis angezeigt.

F1-26 Wenn ein Projekt geöffnet werden soll, obwohl ein anderes ungespeichert ist, fordert der IMC zum Speichern oder Verwerfen auf.

F1-27 Bei Git-Projekten werden Merge-Konflikte eindeutig gemeldet und können nicht stillschweigend überschrieben werden.

Benutzerfreundlichkeit

F1-28 Zuletzt geöffnete Projekte werden in einer Liste angezeigt („Recent Projects“).

F1-29 Die Projektverwaltung läuft ohne merkbare Verzögerungen (< 200 ms beim Öffnen lokaler Projekte).

F1-30 Die Benutzeroberfläche friert nicht ein, auch wenn ein Git-Download länger dauert (asynchrone Verarbeitung).

F1-31 Barrierefreiheit (z. B. Farbsichtbarkeit bei Farbfehlsichtigkeit)

6.4 Synchronisierung von Modelländerung zwischen allen Editoren

Identifikation: F2

6.4.1 User Story

Als Modellierer:in möchte ich Änderungen, die ich in einer Ansicht erfasst habe (z. B. Strukturbaum) automatisch in das «Core Model» (vgl. Kapitel 6.1.2) übernehmen und von dort in den anderen Editoren nachführen,

damit das Modell jederzeit konsistent ist und beliebig zwischen verschiedenen Darstellungs- und Bearbeitungsformen gewechselt werden kann.

6.4.2 Ziel und Beschreibung

Die Funktionalität für die Synchronisation basiert auf einem zentralen «Core Model» als Dreh- und Angelpunkt. Jede Änderung wird in diesem Modell aufgenommen und die betroffenen Editoren werden über die Änderungen informiert.

Im IMC ist sichergestellt, dass alle Editoren jederzeit synchron sind. Sobald eine Änderung vorgenommen wird (z. B. im Text-Editor), wird diese automatisch in das zugrunde liegende «Core Model» übernommen und von dort in allen Ansichten aktualisiert.

Damit wird sichergestellt, dass alle Ansichten, bzw. Modellrepräsentationen jederzeit konsistent sind, unabhängig davon, in welchem Editor eine Änderung vorgenommen wurde.

6.4.3 Akzeptanzkriterien

Interaktion zwischen Modellen und Editoren

F2-1 Änderungen in einem Editor aktualisieren automatisch das «Core Model» und alle Editoren, bzw. Modellrepräsentationen.

F2-2 Wird in einem Editor auf ein Element geklickt, dann wird in den anderen Ansichten direkt zum ausgewählten Element gesprungen. Beispiel: Wird im Diagramm-Editor eine Klasse ausgewählt, dann springt die Auswahl im Strukturbaum-Editor direkt zur ausgewählten Klasse.

F2-3 Das Löschen oder Umbenennen eines Elements wird in allen Editoren reflektiert.

F2-4 Die Synchronisation erfolgt ohne manuelles Speichern.

F2-5 Die Synchronisation erfolgt ohne bemerkbare Verzögerung (i.d.R. unter einer Sekunde)

Persistenz und Datenhaltung

F2-6 Änderungen werden automatisch im «Core Model» gespeichert.

F2-7 Das «Core Model» kann in eine Projektdatei exportiert oder aus einer Projektdatei geladen werden (vgl. Anforderung F1 in Kapitel 6.3).

F2-8 Absturzsichere Zwischenspeicherung (Auto-Save) erfolgt in einem konfigurierbaren Intervall.

F2-9 Das «Core Model» soll auf einem eigens definierten INTERLIS-Modell basieren, bzw. als solches dokumentiert sein. Hinweis: Dieses INTERLIS-Model soll parallel mit dem «Core Model» entwickelt werden.

6.5 Datei-Import und -Export

Identifikation: F3

6.5.1 User Story

Als Modellierer:in möchte ich INTERLIS-Modelle (.ili) und Transferdaten (.xtf/ xml) importieren und exportieren können,

damit ich bestehende Modelle weiterverwenden, Katalogdaten austauschen und Dokumentationen generieren kann.

6.5.2 Ziel und Beschreibung

Der IMC unterstützt den Import und Export von Dateien, die in der INTERLIS-Sprache definiert sind.

Dabei werden zwei unterschiedliche Dateitypen unterschieden:

- ILI-Dateien (.ili): Modellbeschreibungen
- XTF-Dateien (.xtf/ xml): INTERLIS-Transferdateien, zum Austausch von Katalogen oder Referenzdaten

Für den Import / Export (z. B. realisiert durch ein Menu) stehen folgende Funktionen zur Verfügung:

- Import ili-File…
- Import Transfer-File (xtf/ xml)…
- Export ili-File…
- Export Transfer-File (xtf/ xml)…
- Export Documentation…

Exportfunktionen schreiben die jeweils aktuellen Projektinhalte in die ausgewählten Formate. Via IMPORTS referenzierte Datenmodelle müssen nicht importiert werden.

Abgrenzung:

Die Import-/ Export-Funktionen für xtf/ xml-Dateien ist primär für den Transfer von «Externen Katalogen» und die Bereitstellung von Testdaten vorgesehen (vgl. Kapitel 6.13) und beinhaltet auch deren Validierung. Die Funktion ist nicht vorgesehen für die Validierung umfangreicher Datensätze von «Datenproduzent:innen».

6.5.3 Akzeptanzkriterien

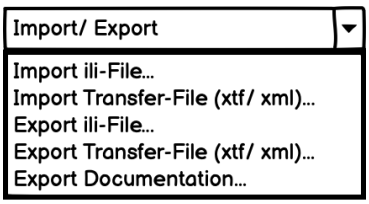

Abbildung 4. Datei-Import und Export (Skizze)

Allgemeines

F3-1 Die Funktionen sind z. B. über ein Menü Import / Export zugänglich.

F3-2 Alle Im- und Exportdialoge verwenden systemübliche Dateiauswahldialoge.

F3-3 Bei allen Vorgängen ist ein technisch erfolgreicher oder fehlerhafter Import/ Export für die Benutzer:in klar ersichtlich.

F3-4 Änderungen/ Überschreibungen werden erst nach expliziter Bestätigung der Benutzer:in vorgenommen.

Import ili-Datei (.ili)

F3-5 Der IMC erkennt beim Import automatisch, ob die .ili-Datei syntaktisch gültig ist.

F3-6 Ist die Datei syntaktisch ungültig, erscheint eine Fehlermeldung mit Angabe der Fehlerstelle.

F3-7 Enthält die Datei ein Modell, das im Projekt bereits existiert, wird der Benutzer:in ein Warndialog angezeigt, über den der Vorgang abgebrochen oder das bestehende Modell überschrieben werden kann.
Bei einem Import eines Modelles soll die Prüfung auf Modellnamen, Modellversion und md5-Wert erfolgen. Weicht einer der Werte ab von jenem des bereits vorhandenen Modells, soll dies dem Benutzer angezeigt werden. Der Benutzer hat dann die Wahl auf Import oder Ablehnung.

F3-8 Ein Überschreiben ersetzt das bestehende Modell vollständig (inkl. Diagrammen, Bindings, Properties, sofern unmittelbar davon abhängig).

F3-9 Existiert das Modell noch nicht, wird es automatisch in das Projekt aufgenommen.

F3-10 Nach erfolgreichem Import werden:

- der Strukturbaum aktualisiert und
- der Text-Editor geöffnet und synchronisiert.

Import Transfer-File (.xtf/ xml)

F3-11 Der IMC erkennt .xtf/ xml-Dateien und ordnet deren Inhalte (z. B. Katalogeinträge) den entsprechenden Modellstrukturen zu.

F3-12 Enthält die .xtf/ xml-Datei Daten für ein Modellelement, das nicht im Projekt existiert, erscheint eine Meldung, dass das Modell zu dieser Transferdatei im Projekt nicht vorhanden ist.

F3-13 Die importierten Daten werden mittels «ilivalidator» gegenüber der zugrundeliegenden Klasse validiert und können nur importiert werden, wenn sie valide sind.

F3-14 XTF-Importe erzeugen keine Modelländerungen, sondern nur Instanzen (Daten-/Katalogeinträge).

F3-15 Existieren für das Modellelement, auf das sich die .xtf/ xml-Datei bezieht, bereits Instanzen, erscheint eine Meldung und die Benutzerin kann wählen, ob sie die Transaktion abbrechen möchte, die Einträge ergänzen möchte oder die im Modell bestehenden Instanzen überschreiben möchte.

F3-16 Nach dem Import stehen die Daten im Instanz-Editor zur Verfügung

Export ili-Datei (.ili)

F3-17 Der IMC kann aktuell im Projekt enthaltene Modelle als .ili-Datei exportieren.

F3-18 Bei mehreren Modellen kann ausgewählt werden, welche Exportiert werden sollen.

F3-19 Der Dateiname schlägt automatisch den Modellnamen vor, sofern nicht mehrere Modelle im Projekt vorhanden sind.

F3-20 Der Export erzeugt INTERLIS-konformen Code basierend auf dem aktuellen Modellstand.

F3-21 Syntaxfehler werden vor dem Export geprüft. Bei Fehlern wird der Export blockiert und eine Meldung angezeigt: „Das Modell enthält Syntaxfehler. Export nicht möglich.“

Export Transfer-File (.xtf/ xml)

F3-22 Der IMC kann die zum Modell gehörenden Daten (z. B. Kataloginhalte) als .xtf/ xml exportieren.

F3-23 Der Export stellt über den «ilivalidator» sicher, dass die Transferdatei Modell-konform ist.

F3-24 Fehlende Pflichtangaben (MANDATORY) werden gemeldet.

F3-25 Ein Export einer fehlerhaften Transferdatei ist nicht möglich.

F3-26 Der Export schlägt einen Dateinamen basierend auf Modell und Datum vor.

Export Documentation…

Die allgemeinen Anforderungen an die Dokumentgenerierung sind in der Anforderung F14 im Kapitel 6.16 beschrieben.

F3-27 Der IMC erzeugt Dokumentationsdateien des Modells in einem auswählbaren Format.

F3-28 Die Benutzer:in kann die Bestandteile der Dokumentation auswählen

F3-29 Die Dokumentation kann folgende Bestandteile beinhalten:

- INTERLIS-Elemente (Modell, Topics etc.) mit Objektkatalog für Klassen-/ Attributbeschreibungen
- Prosatext
- Diagramme

- Instanzen (z. B. Kataloginhalte)
- INTERLIS-Code

F3-30 Die Benutzer:in kann das Ausgabeverzeichnis auswählen.

6.6 Flexibler und anpassbarer Arbeitsbereich

Identifikation: F4

6.6.1 User Story

Als Modellierer:in möchte ich eine klar strukturierte Arbeitsoberfläche des IMC flexibel an meine Bedürfnisse anpassen können,

damit ich INTERLIS-Modelle im Gesamtzusammenhang verstehen kann und ich genau jene Editoren aufblenden kann, die ich für meine aktuelle Modellierungsaufgabe benötige und sie so anordnen kann, dass ein effizienter Arbeitsfluss möglich ist.

6.6.2 Ziel und Beschreibung

Der grundsätzliche Aufbau der IMC-Benutzeroberfläche orientiert sich an modernen, modularen Entwicklungsumgebungen (z. B. Visual Studio Code).

Die Benutzeroberfläche besteht aus einem zentralen Fenster mit frei konfigurierbaren Bereichen, in denen verschiedene Editoren (Text-, Diagramm-, Objektkatalog-, Instanz-, Eigenschaften-Editor) angezeigt werden können.

Benutzer:innen entscheiden selbst, welche Editoren angezeigt werden, wie diese angeordnet werden, und in welcher Grösse sie dargestellt werden.

Editoren können angedockt, nebeneinandergelegt oder in Tabs organisiert werden. Der IMC merkt sich die zuletzt verwendete Anordnung und stellt diese beim nächsten Öffnen wieder her.

Alle Editoren werden synchronisiert und ermöglichen eine über das gesamte Projekt konsistente Modellierung.

Übergeordnete Funktionen wie Projektverwaltung, Toolbar-Aktionen und Ansichtseinstellungen stehen zentral zur Verfügung.

6.6.3 Benutzeroberfläche

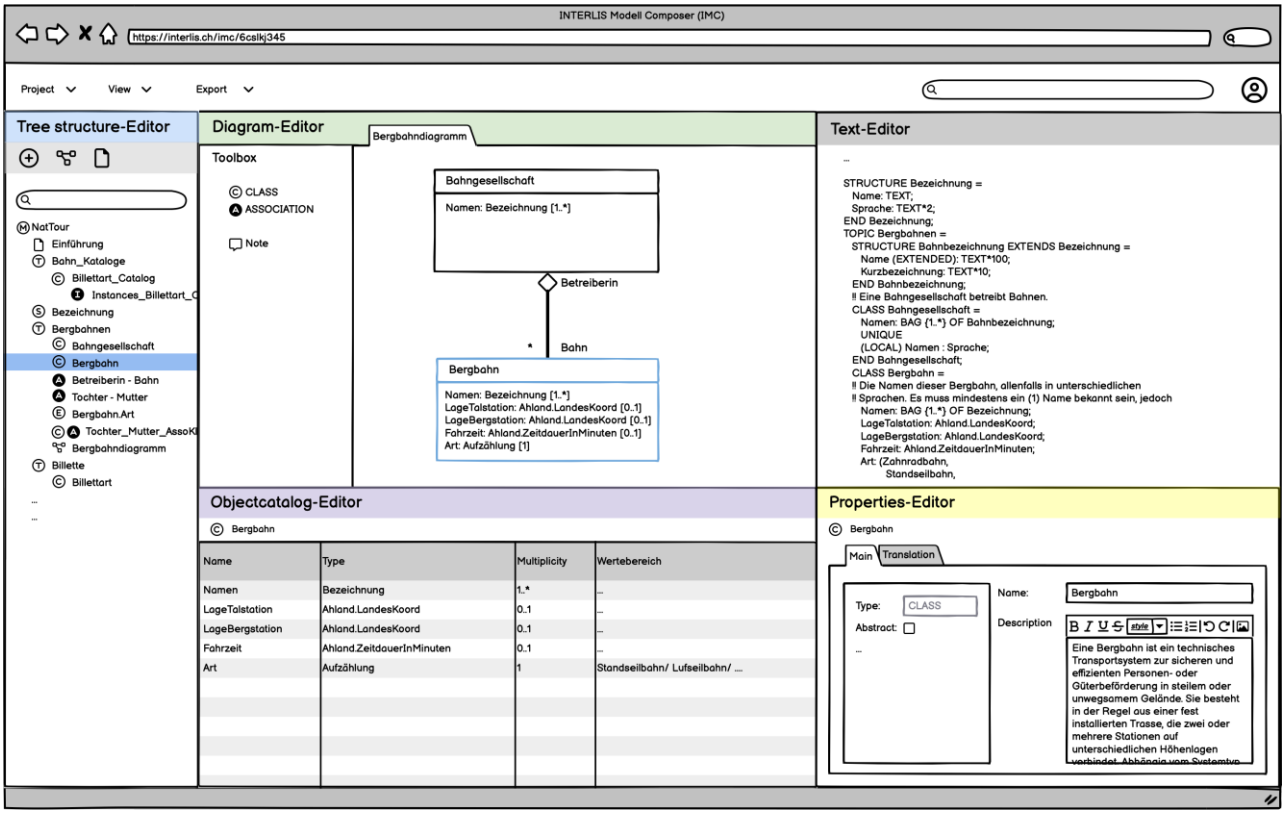

Abbildung 5. Benutzeroberfläche IMC (Skizze)

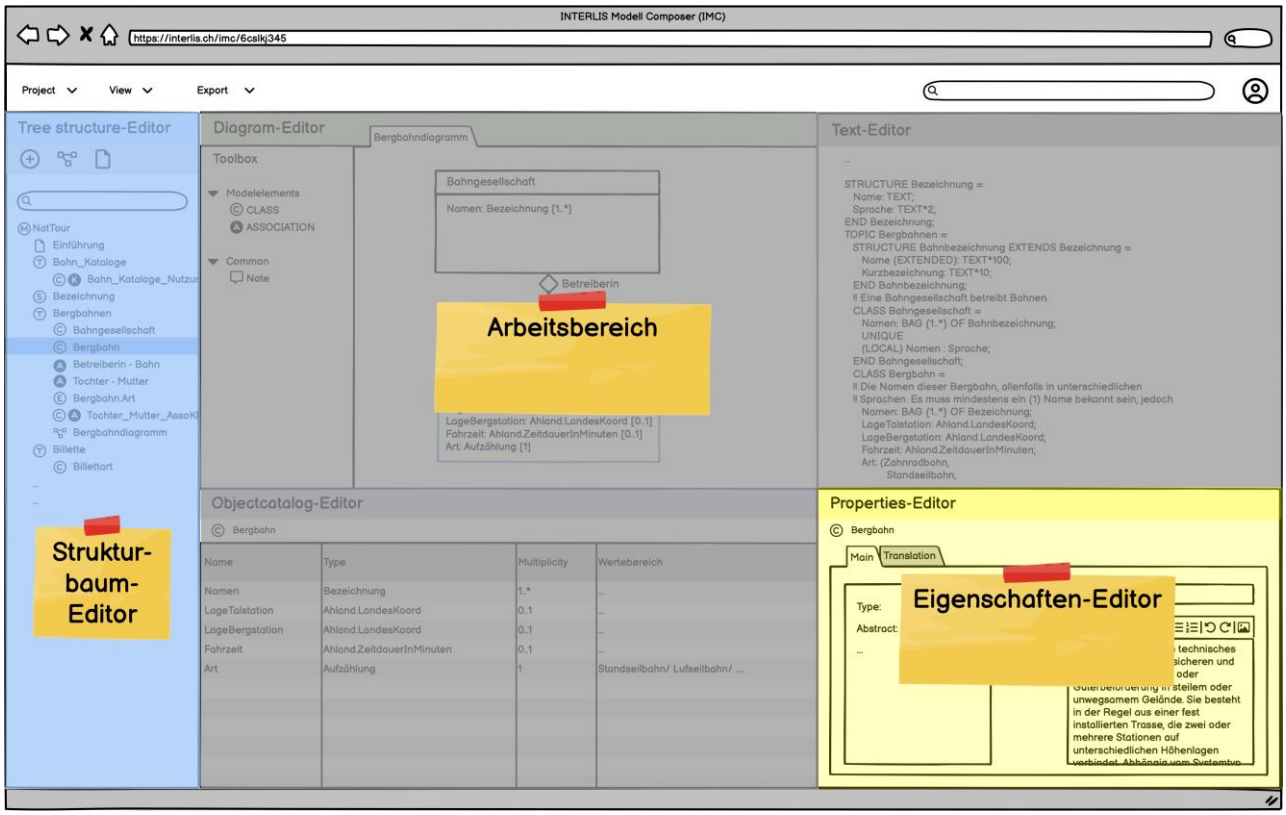

Abbildung 6. Benutzeroberfläche IMC - Bereiche (Skizze)

Die IMC-Benutzeroberfläche besteht aus den folgenden nennenswerten Elementen:

1	Bezeichnung:	Strukturbaum-Editor
	Elementtyp:	Bereich
	Bemerkungen:	Strukturbaum-Editor zur Navigation und zur Bearbeitung der Struktur (vgl. Kapitel 6.9)
2	Bezeichnung:	Arbeitsbereich
	Elementtyp:	Bereich
	Bemerkungen:	Arbeitsbereich, in dem verschiedene Editoren (Text-, Diagramm-, Objektkatalog-, Instanz-Editor) sowie ein Prüfprotokoll (Fehlerliste) angezeigt werden
3	Bezeichnung:	Eigenschaften-Editor
	Elementtyp:	Bereich
	Bemerkungen:	Eigenschaften-Editor zur Anzeige und Bearbeitung der Eigenschaften ausgewählter Modellelemente

6.6.4 Akzeptanzkriterien

Allgemeiner UI-Aufbau

F4-1 Die Benutzeroberfläche besteht aus einem Hauptfenster mit frei platzierbaren Editorbereichen.

F4-2 Ein grundlegendes Seitenpanel (Strukturbaum) kann unabhängig von Editoren ein- und ausgeklappt werden.

F4-3 Alle Editoren werden innerhalb desselben Fensters angezeigt, nicht in externen Fenstern.

Editorsteuerung

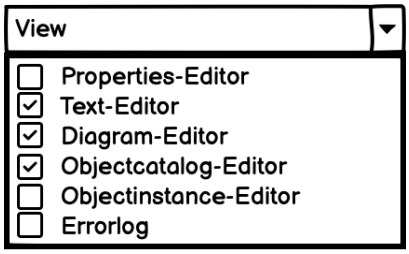

Abbildung 7. Anzeigen von Editoren (Skizze)

F4-4 Die Benutzer:in kann bestimmen, welche Editoren angezeigt werden.

F4-5 Editoren können frei angeordnet werden:

- Als Tabs im selben Bereich
- Nebeneinander (horizontal split)
- Untereinander (vertical split)
- Abgedockt innerhalb des Hauptfensters

F4-6 Die UI erlaubt es, mehrere Editoren gleichzeitig sichtbar zu machen.

F4-7 Die Grösse jedes Editorbereichs kann angepasst werden (interaktives Resizing).

Anordnung von Editorenbereichen

F4-8 Editorbereiche lassen sich per Drag-and-Drop verschieben, teilen und kombinieren.

F4-9 Tabs können zwischen Editorbereichen verschoben werden.

F4-10 Eine Editorgruppe kann mehrere Tabs enthalten.

F4-11 Die aktuell geöffnete Editoranordnung wird beim Schliessen des IMC gespeichert und beim nächsten Start wiederhergestellt.

Auswahl & Kontext

F4-12 Wird ein Diagramm geöffnet, dann öffnet sich automatisch der Diagramm-Editor, falls nicht bereits geöffnet.

F4-13 Wird ein Prosatext-Element geöffnet, dann öffnet sich automatisch der Eigenschaften-Editor, falls nicht bereits geöffnet.

F4-14 Die Benutzer:in kann Editoren auch unabhängig vom Strukturbaum öffnen oder schliessen.

F4-15 Die Sichtbarkeit der Editoren wird nicht durch die Reihenfolge im Strukturbaum eingeschränkt.

Arbeitsfluss & Konsistenz

F4-16 Die UI reagiert flüssig auf das Verschieben oder Umordnen von Editorbereichen.

F4-17 Die Benutzeroberfläche verhindert unzulässige Anordnungen (z. B. Editor ausserhalb des Fensters).

6.7 Text-Editor mit Syntaxüberprüfung

Identifikation: F5

6.7.1 User Story

Als Modellierer:in möchte ich INTERLIS-Modelle in einem textbasierten Editor mit Syntax-Highlighting, Autovervollständigung und direkter Syntaxvalidierung bearbeiten,

damit ich effizient und fehlerfrei strukturierte INTERLIS-Modelle erstellen und anpassen kann.

6.7.2 Ziel und Beschreibung

Der Text-Editor bietet Unterstützung der INTERLIS-Syntax im Umfang der Implementation des Compilers ili2c und überprüft Code auf syntaktische Fehler.

Neben Syntax-Highlighting und Einrückung bietet der Text-Editor Funktionen wie Autovervollständigung, Fehlerhinweise, kontextabhängige Hilfe und direkte Navigation zu Fehlerstellen.

Der Text-Editor ermöglicht die textuelle Bearbeitung von Modellen, steht neben anderen Editoren (z. B. Strukturbaum, Diagramm) und muss mit diesen synchronisiert sein.

6.7.3 Akzeptanzkriterien

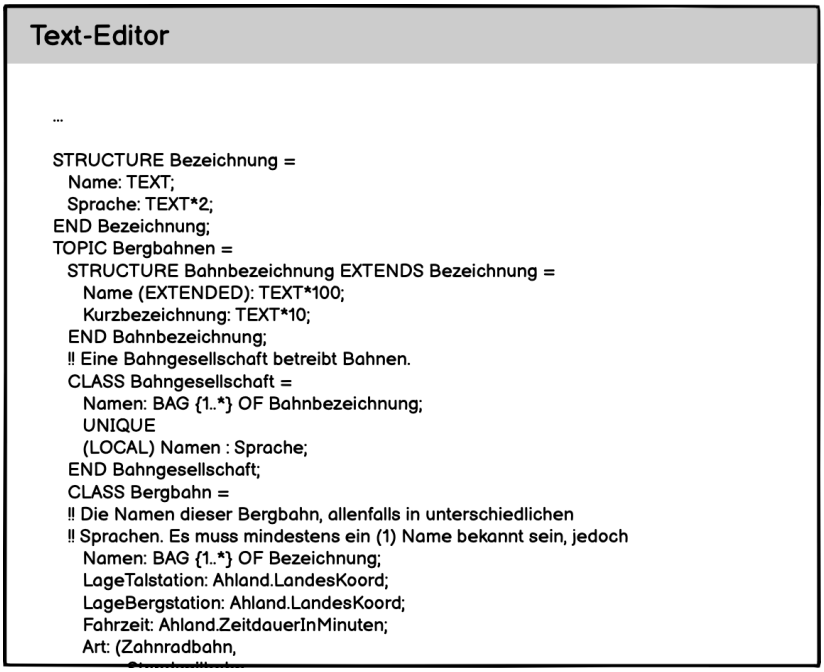

Abbildung 8. Benutzeroberfläche Text-Editor (Skizze ohne farbliche Hervorhebung)

Anzeige und Syntax-Highlighting

F5-1 Der Editor erkennt die INTERLIS-Schlüsselwörter[6] (z. B. MODEL, TOPIC, CLASS, ATTRIBUTE, DOMAIN, UNIT) und hebt sie farblich hervor.

F5-2 Kommentare werden in eigener Farbe dargestellt.

F5-3 Zahlen, Operatoren und INTERLIS-Wertebereiche (z. B. TEXT, NUMERIC, COORD) werden unterschiedlich farblich kodiert.

F5-4 Gross-/ Kleinschreibung wird unterschieden.

F5-5 Syntax-Farben können über ein sog. «Theme» angepasst werden.

[6] Vgl. RefHB Kapitel 3.2.7 https://geostandards-ch.github.io/doc_refhb24/#_sonderzeichen_und_reservierte_w%C3%B6rter

Syntaxprüfung (Parsing und Fehlererkennung)

F5-6 Der Editor überprüft die Syntax des aktuellen Modells nach dem Speichern.

F5-7 Syntaxfehler (z. B. fehlende Semikolons, unvollständige Klammern, falsche Schlüsselwörter) werden sofort erkannt.

F5-8 Fehler werden mit roter Wellenlinie markiert und in einer Fehlerliste mit Zeilennummer, Spalte und Fehlermeldung angezeigt.

F5-9 Die Fehlermarkierungen verschwinden automatisch, wenn ein Fehler korrigiert wird.

F5-10 Mehrfache oder verschachtelte Fehler werden einzeln behandelt.

F5-11 Für die Syntaxprüfung ist der bestehende Compiler «ili2c» zu benutzen.

F5-12 Die Sprachversion eines Modells wird über die allererste Zeile im Modell definiert. Dort kann stehen:

a) TRANSFER INTERLIS1;

b) INTERLIS 2.2;

c) INTERLIS 2.3;

d) INTERLIS 2.4;

Fall a) soll direkt detektiert werden und es soll ein Hinweis erfolgen, dass INTERLIS 1 durch das Werkzeug nicht unterstützt wird (da nicht objektorientiert, also auch nicht kompatibel zu UML).

Auf die Fälle b) bis d) soll der Compiler eingehen, d.h. wenn bei einem bestehenden 2.3er-Modell die erste Zeile auf 2.4 abgeändert wird, dann soll mit dieser Sprachversion geprüft und es sollen alle „Fehler“ detektiert werden.

F5-13 Modelle können unabhängig von Syntaxfehlern geöffnet und bearbeitet werden.

F5-14 Kann ein Diagramm aufgrund von Syntaxfehlern nicht angezeigt werden, dann ist ein entsprechender Hinweis anzuzeigen.

Autovervollständigung

F5-15 Autovervollständigung schlägt INTERLIS-Schlüsselwörter, Modellnamen, Klassen, Attribute und Domains kontextsensitiv vor.

F5-16 Vorschläge erscheinen nach Drücken von Strg+Leertaste oder automatisch nach bestimmten Eingaben (z. B. nach „CLASS “).

F5-17 Die Benutzer:in kann zwischen Vorschlägen navigieren und per Enter oder Tab übernehmen.

F5-18 Autovervollständigung berücksichtigt vorhandene Modellkontexte (z. B. TOPIC-Abhängigkeiten und mittels IMPORTS referenzierte Datenmodelle).

F5-19 Vorschläge für Modellobjekte werden alphabetisch sortiert.

F5-20 Die Autovervollständigung kann deaktiviert werden (Benutzereinstellung).

Formatierung und Layout

F5-21 Der Editor unterstützt automatische Einrückung und Ausrichtung von Codeblöcken (z. B. Klassenkörper, Attribute).

F5-22 Für das Einrückungen soll die «pretty-print-Funktion» des Compilers genutzt werden. Der Benutzer soll dabei die Funktion explizit auslösen, d.h. der Code soll nicht automatisch ausgerichtet werden.

F5-23 Zeilennummern, Einrückungslinien und visuelles Klammern-Matching sind sichtbar.

F5-24 Der Editor unterstützt das Auf- und Zuklappen von TOPICS und CLASSes (folding)
Hinweis: Das Auf-/ Zuklappen ist nicht mit dem Strukturbaum-Editor zu synchronisieren.

Interaktion mit anderen Editoren und referenzierten Modellen (IMPORTS)

F5-25 Die Interaktion und Synchronisation mit anderen Editoren sind in Kapitel 6.12 festgehalten.

F5-26 Vom referenzierenden Element kann direkt zum entsprechenden Element im importierten Modell navigiert werden (analog zur Basisklassen-Navigation in IDEs).

F5-27 Importierte Modelle werden im IMC als schreibgeschützt (read-only) angezeigt; eine Bearbeitung ist nicht möglich.

F5-28 Die Rücknavigation zum Ausgangselement im eigenen Modell ist jederzeit möglich.

Fehler- und Warnungsdarstellung

F5-29 Alle Prüfresultate werden in der „Prüfprotokoll“-Anzeige (Errorlog) mit Spalten für Typ (Fehler/Warnung), Ort, Zeile und Beschreibung angezeigt.

F5-30 Klick auf einen Eintrag springt im Editor zur Fehlerstelle.

F5-31 Fehlerliste ist sortier- und filterbar.

F5-32 Eine Statusleiste zeigt die Gesamtanzahl an Fehlern und Warnungen.

F5-33 Die Prüfergebnisse bleiben sichtbar, bis sie manuell gelöscht oder überschrieben werden.

F5-34 Prüfberichte können kopiert oder exportiert werden (z. B. als TXT, CSV oder JSON).

Erweiterbarkeit und Zukunftssicherheit

F5-35 Syntaxprüfung wird automatisch nach Eingabe (debounced 300ms) ausgelöst und Fehler unmittelbar angezeigt (≤ 1s Verzögerung).

F5-36 Syntaxprüfung funktioniert auch bei grösseren Dateien (> 10 000 Zeilen (inkl. IMPORTS)).

F5-37 Parser und Highlighter sind modular aufgebaut, um neue INTERLIS-Versionen leicht zu unterstützen.

Usability

F5-38 „Undo/Redo“ ist soweit wie möglich zurück unterstützt, jedoch mindestens zwischen dem aktuellen Zustand und der letzten Speicherung.

6.8 Struktur- und Semantikprüfung

Identifikation: F6

6.8.1 User Story

Als Modellierer:in möchte ich sicherstellen, dass INTERLIS-Modelle strukturell und semantisch korrekt sind,

damit alle Modelle den INTERLIS-Regeln, Beziehungen, Multiplizitäten und Referenzen entsprechen und keine logischen Inkonsistenzen enthalten.

6.8.2 Ziel und Beschreibung

Die Struktur- und Semantikprüfung überprüft alle Modellkomponenten gemäss der INTERLIS-Spezifikation.

Sie ergänzt die reine Syntaxprüfung (F5) durch tiefgehende logische und regelbasierte Validierungen:

Vererbungshierarchien, Beziehungen, Multiplizitäten, Typkompatibilität, Referenzen, Domains, Units, Constraints und Konsistenz über Modellgrenzen hinweg.

Die Prüfung kann automatisch (bei Modelländerung) oder manuell (durch Befehl „Modell prüfen“) ausgelöst werden.

Ergebnisse werden in einer Fehler- und Warnliste dargestellt und visuell im IMC markiert (z. B. im Strukturbaum-Editor und Text-Editor).

6.8.3 Akzeptanzkriterien

Allgemeines Verhalten

F6-1 Der IMC führt eine vollständige Struktur- und Semantikprüfung durch, sobald ein Modell gespeichert oder explizit validiert wird.

F6-2 Die Prüfung erfolgt gemäss INTERLIS-Referenzhandbuch, indem der Compiler «ili2c» eingebunden wird. Damit werden folgende Aspekte geprüft:

- Modellstruktur
- Beziehungen und Multiplizitäten
- Typen, Domains und Units
- Constraints und Bedingungen
- Referenzen und Vererbung

F6-3 Alle Fehler, Warnungen und Hinweise werden in einer strukturierten Ergebnisliste ausgegeben.

F6-4 Benutzer:innen können die Prüfung manuell starten.

F6-5 Die Prüfung läuft asynchron, damit der IMC weiterhin bedienbar bleibt.

F6-6 Prüfergebnisse werden im Projektkontext gespeichert für spätere Vergleiche.

F6-7 Die automatische Syntaxprüfung kann ein- und ausgeschaltet werden.

Fehler- und Warnungsdarstellung

F6-8 Analog Akzeptanzkriterien F5.

Integration mit anderen Funktionen

F6-9 Bei Korrekturen im Editor (F5) aktualisiert sich das Prüfergebnis automatisch.

F6-10 Bei Dokumentationsgenerierung (F14) werden nur geprüfte Modelle zugelassen.

F6-11 Die Semantikprüfung bezieht importierte Modelle mit ein.
Hinweis: Der Einbezug importierter Modelle erfolgt über den Compiler ili2c, der bereits in der Lage ist die entsprechenden Modelle in den öffentlich zugänglichen Repositories mit einzubeziehen.

Referenzierte Modelle (IMPORTS)

F6-12 Der IMC löst alle via IMPORTS deklarierten Modelle automatisch auf und stellt sicher, dass referenzierte Elemente (z.B. Klassen, Attribute) im Zielmodell tatsächlich vorhanden sind.

F6-13 Referenzen auf Elemente importierter Modelle werden auf Korrektheit validiert. Ungültige Referenzen werden mit Fehlermeldung ausgewiesen.

Performance und Stabilität

F6-14 Prüfung grosser Modelle (bis 10 000 Zeilen, Imports miteingerechnet) erfolgt innerhalb von 3 Sekunden.

F6-15 Bei Unterbrechung bleibt letzter Prüfstatus erhalten.

6.9 Strukturbaum-Editor

Identifikation: F7

6.9.1 User Story

Als Modellierer:in möchte ich im IMC einen hierarchischen Strukturbaum sehen und bearbeiten,

damit ich INTERLIS-Modelle (Modelle, Topics, Klassen, Strukturen, Attribute, Beziehungen usw.) effizient erstellen, verstehen und pflegen kann.

6.9.2 Ziel und Beschreibung

Der Strukturbaum ist eine hierarchische Darstellung aller INTERLIS-Modellelemente (Modelle, Topics, Klassen, Attribute etc.) und weiteren Projekt-Artefakten (Prosabeschreibungen, Instanzen) und dient als zentrale Navigations- und Bearbeitungshilfe. Änderungen im Baum werden direkt ins «Core Model» übernommen und mit den anderen Editoren synchronisiert.

6.9.3 Akzeptanzkriterien

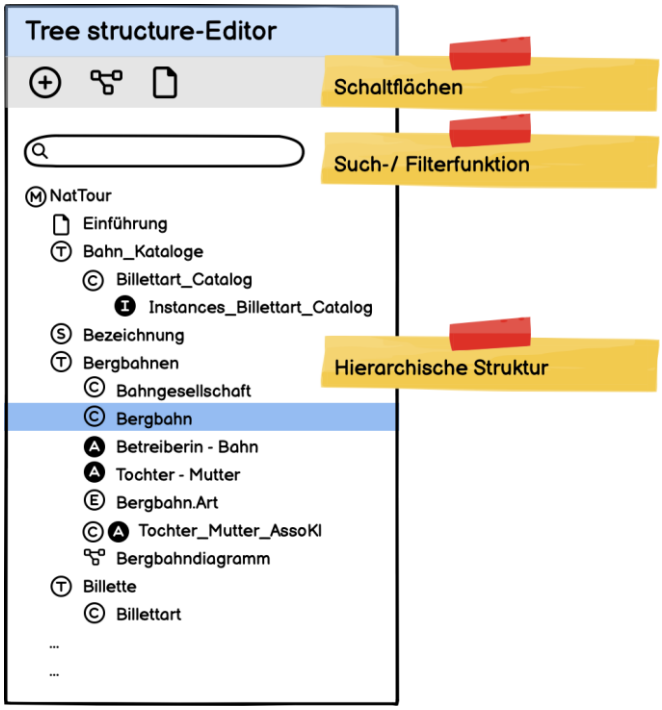

Abbildung 9. Benutzeroberfläche Strukturbaum-Editor (Skizze)

Die Skizze zeigt die drei Bereiche des Strukturbaum-Editors.

Über die Schaltflächen können neue INTERLIS-Elemente, Diagramme oder Prosatexte hinzugefügt werden. Die Such-/ Filterfunktion reduziert mittels Texteingabe den Strukturbaum aufgrund der eingegebenen Zeichenfolge. Anzeige und Bearbeitung der hierarchischen Struktur eines IMC-Projekts.

Anzeigen des Strukturbaums

F7-1 Der Strukturbaum zeigt alle relevanten INTERLIS-Elemente hierarchisch an:

- MODEL
- DOMAIN
- UNIT
- TOPIC
- CLASS / STRUCTURE
- ATTRIBUTE
- ASSOCIATION
- CONSTRAINT

- VIEW
- GRAPHIC / SIGNATURE
- PARAMETER / METAOBJECT

F7-2 Die Hierarchie folgt der INTERLIS-Spezifikation (z. B. Model → Topic → Class → Attribute).

F7-3 Jedes Element zeigt Name und Typ (z. B. CLASS Gebaeude)

F7-4 Der Baum ist dynamisch — Änderungen im Modell (z. B. neue Klasse) werden sofort reflektiert.

F7-5 Benutzer:innen können im Strukturbaum neue Elemente anlegen (z. B. „Neue Klasse hinzufügen“).

Bearbeiten des Strukturbaums

F7-6 INTERLIS-Elemente können neu angelegt werden und werden in den Strukturbaum an der markierten/ ausgewählten Stelle unter Einhaltung der INTERLIS-Sprachregeln in Bezug auf die zulässige hierarchische Modell-Struktur eingefügt (z. B. Einfügen einer CLASS unter einem TOPIC. Unter einer CLASS kann kein TOPIC eingefügt werden).

F7-7 Per IMPORTS referenzierte Modelle werden im Strukturbaum als eigener Knoten dargestellt und sind visuell als importiert gekennzeichnet.

F7-8 Diagramm-Elemente (vgl. F8 in Kapitel 6.10) und Prosatext-Elemente (vgl. F12 in Kapitel 6.14) können neu angelegt und in den Strukturbaum an der markierten/ ausgewählten Stelle eingefügt werden.

F7-9 Diagramm Elemente können unter folgenden Elementen eingefügt werden:

- MODEL
- DOMAIN
- UNIT
- TOPIC

F7-10 Zusätzliche Prosatext-Elemente, für weitergehende Beschreibungen (z.B. Erfassungshinweise) können an beliebiger Stelle im Strukturbaum eingefügt werden. Es sind ergänzende Prosabeschreibungen, die sich nicht direkt auf ein Modell-Element beziehen (vgl. F10-20 in Kapitel 6.12.3). Die Bearbeitung der Prosatexte erfolgt direkt im Eigenschaften-Editor, inkl. allfälliger Übersetzungen (vgl. F12 in Kapitel 6.14).

F7-11 Für CLASS-Elemente kann eine Liste von Instanzen hinzugefügt werden. Die Liste der Instanzen beinhaltet die Definition von Instanzen gemäss CLASS-Definition die über den Instanz-Editor gepflegt werden können (vgl. Anforderung F11 in Kapitel 6.13).

F7-12 Elemente können umbenannt, gelöscht oder verschoben werden. Dabei sind die INTERLIS-Regeln einzuhalten.
Beispiel: Eine Beziehung darf im Strukturbaum zwingend erst nach der Definition ihrer Quell- und Zielklasse eingeordnet sein.

F7-13 Wird ein Element ausgewählt, dann wird dieses Element in den anderen geöffneten Editoren wie folgt ebenfalls ausgewählt:

- Eigenschaften-Editor: Das ausgewählte Element wird geladen.
- Text-Editor: Der Cursor springt direkt zum Beginn der Element-Definition im INTERLIS-Code des ausgewählten Elements.
- Diagramm-Editor: Ist das ausgewählte Element im Diagramm enthalten, dann soll es im Diagramm markiert werden.
- Objektkatalog-Editor: Das ausgewählte Element wird geladen.
- Instanz-Editor: Sofern eine Instanz-Liste ausgewählt wird, wird diese geladen.

F7-14 Drag-and-Drop ist möglich, z. B.:

- Eines oder mehrere Attribute können in eine andere Klasse verschoben werden.
- Eine Klasse kann zwischen Topics verschoben werden, wenn das zulässig ist.

F7-15 Ungültige Aktionen (z. B. Verschieben einer CLASS in eine andere CLASS) werden verhindert oder mit Fehlermeldung angezeigt.

F7-16 Nach Änderungen prüft der IMC automatisch die semantische Korrektheit des INTERLIS-Modells.

F7-17 Fehlerhafte oder unvollständige Elemente werden im Baum visuell markiert (z. B. rotes Symbol, Tooltip mit Fehlertext).

Bedienbarkeit und Darstellung

F7-18 Der Strukturbaum unterstützt Ein- und Ausklappen von Knoten.

F7-19 Kontextmenüs (Rechtsklick) bieten elementabhängige Aktionen an (z. B. „Neues Attribut“, „Löschen“).

F7-20 Icons oder Farben unterscheiden die Elementtypen. Nach Möglichkeit sind «gängige» Icons zu verwenden, die auch in anderen Editoren benutzt werden. Z. B. Icon für Klassen.

F7-21 Eine dynamische Such-/Filterfunktion ermöglicht das schnelle Auffinden von Elementen nach Namen oder Typ. Der Strukturbaum wird direkt auf die Elemente reduziert, die der Zeichenfolge der Filtereingabe entsprechen. Die Filterung hat keine Auswirkung auf die in den anderen Editoren geladenen Modellelemente.

F7-22 Die Anzeige im Strukturbaum kann auf die reinen INTERLIS-Modell-Artefakte eingeschränkt/ gefiltert werden.

Datenhaltung und Persistenz

F7-23 Beim Laden einer IMC-Projektdatei wird der Strukturbaum korrekt wiederhergestellt.

F7-24 Die Struktur bleibt auch nach IMC-Neustart erhalten.

6.10 Diagramm-Editor

Identifikation: F8

6.10.1 User Story

Als Modellierer:in möchte ich INTERLIS-Modelle visuell bearbeiten können,

um Klassen, Relationen und Attribute intuitiv zu erstellen und Modellstrukturen schneller zu verstehen.

6.10.2 Ziel und Beschreibung

Der visuelle Modell-Editor stellt INTERLIS-Modelle als grafisches Diagramm dar (analog UML). Er zeigt Klassen, Strukturen, Beziehungen, Multiplizitäten und Vererbungen als Knoten und Linien.

Benutzer:innen können Elemente per Drag-and-drop platzieren, bearbeiten oder löschen. Der Editor ist bidirektional mit dem textbasierten Editor (F1) und dem Strukturbaum (F11) verknüpft.

6.10.3 Akzeptanzkriterien

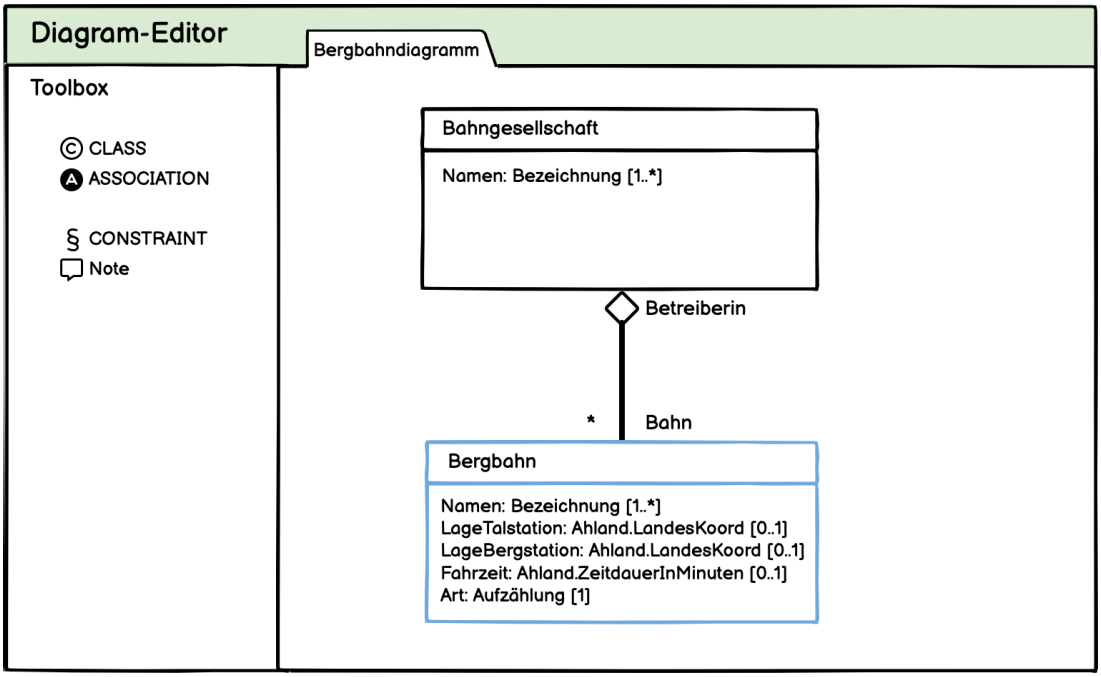

Abbildung 10. Benutzeroberfläche Diagramm-Editor (Skizze)

Darstellung

F8-1 Klassen werden als Boxen mit Namen und Attributen dargestellt.

F8-2 Bei abstrakten Klassen wird der Klassenname kursiv geschrieben

F8-3 Beziehungen (ASSOCIATIONs) erscheinen als Linien mit Rollen und Multiplizitäten. Aggregationen und Kompositionen werden als solche dargestellt.

F8-4 Beziehungen können durch Einfügen von Zwischenpunkten grafisch angepasst werden.

F8-5 Vererbungen (EXTENDS) werden mit Dreiecks-Pfeilen visualisiert.

F8-6 Farben und Formen unterscheiden Typen (z. B. Topic, Class, Structure).

F8-7 Anzeigensteuerung für das gesamte Diagramm für die Darstellung von Klassen (ein-/ ausblenden):

- Klassenname wird immer angezeigt
- Attributnamen
    - Multiplizitäten
    - Wertebereiche

F8-8 Der Benutzer kann Rollenbezeichnungen und Multiplizitäten für das gesamte Diagramm ein- oder ausblenden.

F8-9 Rahmen und Hintergrund von Diagramm-Elemente können für das Diagramm frei eingefärbt werden. Bei Mehrfachselektion werden alle Elemente gleichzeitig eingefärbt.

Interaktion

F8-10 Ein oder mehrere Elemente können per Drag-and-drop verschoben werden.

F8-11 Bestehende Elemente können per Drag-and-drop aus dem Strukturbaum-Editor auf das Diagramm gezogen werden und können danach auf dem Diagramm weiterbearbeitet werden.

F8-12 Elemente aus per IMPORTS referenzierten Modellen können im Diagramm verwendet und angezeigt werden und sind visuell von Elementen des eigenen Modells unterscheidbar (z. B. durch Farbe oder Symbol).

F8-13 Über eine Toolbox können per Drag-and-drop neue Elemente (Klassen, Beziehungen) auf das Diagramm gezogen werden. Die Elemente werden im Strukturbaum an der Stelle, an der sich das Diagramm befindet, angelegt und können danach auf dem Diagramm oder in anderen Editoren weiterbearbeitet werden.

F8-14 Über eine Toolbox können per Drag-and-drop neue Constraints auf ein Element auf dem Diagramm gezogen werden. Der Constraint wird direkt mit dem selektierten Element verbunden und diesem im Strukturbaum untergeordnet.

F8-15 Constraints werden ähnlich einer Notiz dargestellt und es wird optisch eine Beziehung zum Element angezeigt, auf dass sich der Constraint bezieht, sofern sich dieses Element ebenfalls auf dem Diagramm befindet.

F8-16 Note-Elemente können aus der Toolbox auf dem Diagramm angebracht werden für zusätzliche Informationen oder Erläuterungen.

F8-17 Ein Note-Element kann einem oder mehreren Elementen auf dem Diagramm angehängt werden.

F8-18 Wird ein Element ausgewählt, dann wird dieses Element in den anderen geöffneten Editoren wie folgt ebenfalls ausgewählt:

- Strukturbaum-Editor: Das ausgewählte Element wird markiert.
- Eigenschaften-Editor: Das ausgewählte Element wird geladen.
- Text-Editor: Der Cursor springt direkt zum Beginn der Element-Definition im INTERLIS-Code des ausgewählten Elements.
- Objektkatalog-Editor: Das ausgewählte Element wird geladen.
- Instanz-Editor: Keine Interaktion.

F8-19 Elemente können aus dem Diagramm gelöscht werden. Löschen bedeutet für Klassen und Beziehungen lediglich, dass sie im Diagramm nicht mehr angezeigt werden.

F8-20 Note-Elemente können nur ganz gelöscht werden.

F8-21 Elemente können aus dem Diagramm und dem Modell gelöscht werden. Diese Aktion ist von den oben genannten Löschaktionen in der Interaktion genügend abzusichern, damit nicht versehentlich Elemente komplett aus dem Modell gelöscht werden.

F8-22 Benutzer:innen können Zoomen, die Ansicht verschieben (Pan) und Objekte an einem Raster ausrichten.

F8-23 Änderungen werden sofort in das «Core Model» übernommen.

Synchronisation & Konsistenz

Die allgemeinen Anforderungen an Synchronisation und Konsistenz sind in Anforderung F2 im Kapitel 6.12 beschrieben.

F8-24 Visualisierung bleibt konsistent bei Änderungen im Modell. Konsistenz bedeutet:

- Gelöschte Klassen verschwinden aus dem Diagramm.
- Neue Klassen müssen manuell hinzugefügt werden.
- Beziehungsänderungen aktualisieren Pfeile.
- User-Layout bleibt wenn möglich erhalten, solange keine strukturellen Änderungen gemacht werden.

F8-25 Fehler- oder Warnsymbole sind sichtbar und anklickbar.

Usability

F8-26 Anordnung der Elemente kann automatisch optimiert werden („Auto Layout“).

F8-27 Export als SVG, PNG oder PDF ist möglich.

F8-28 Ansichtseinstellungen (Zoom, Filter) bleiben pro Modell erhalten.

F8-29 „Undo/Redo“ ist soweit wie möglich zurück unterstützt, jedoch mindestens zwischen dem aktuellen Zustand und der letzten Speicherung.

F8-30 Grosse Modelle (> 30 Klassen) bleiben performant darstellbar. Beispiel VSA-DSS: 69 konkrete Klassen, 9 abstrakte Klassen, 490 Attribute

Datenhaltung und Persistenz

F8-31 Das Diagramm-Layout wird als Teil des IMC-Projekts gespeichert und versioniert.

F8-32 Beim Laden eines Diagramms wird die Ansicht korrekt wiederhergestellt.

F8-33 Die Ansicht bleibt auch nach IMC-Neustart erhalten.

F8-34 Zu einem Diagramm kann Prosatext erfasst werden (vgl. Anforderung F12 in Kapitel 6.14). Die Erfassung erfolgt über den Eigenschaften-Editor (vgl. Anforderung F10 in Kapitel 6.12).

6.11 Objektkatalog-Editor

Identifikation: F9

6.11.1 User Story

Als Modellierer:in möchte ich Objektkataloge in tabellarischer Form bearbeiten,

um Klassen, Attribute, Wertebereiche und Beschreibungen strukturiert verwalten zu können.

6.11.2 Ziel und Beschreibung

Der Objektkatalog-Editor stellt Klassen und ihre Eigenschaften tabellarisch dar. Er dient der Pflege der Klasse- und Attributdefinition im INTERLIS-Modell.

Bearbeitet werden können:

- Klassen/ Assoziationsklassen und ihre Attribute
- Typen, Multiplizitäten und Wertebereiche von Attributen

Die Tabellenansicht ist synchron mit dem textbasierten INTERLIS-Modell und unterstützt Export und Filterung.

Hinweis:

Der Umgang mit «Externen Katalogen» ist im Kapitel 6.13 beschrieben.

6.11.3 Akzeptanzkriterien

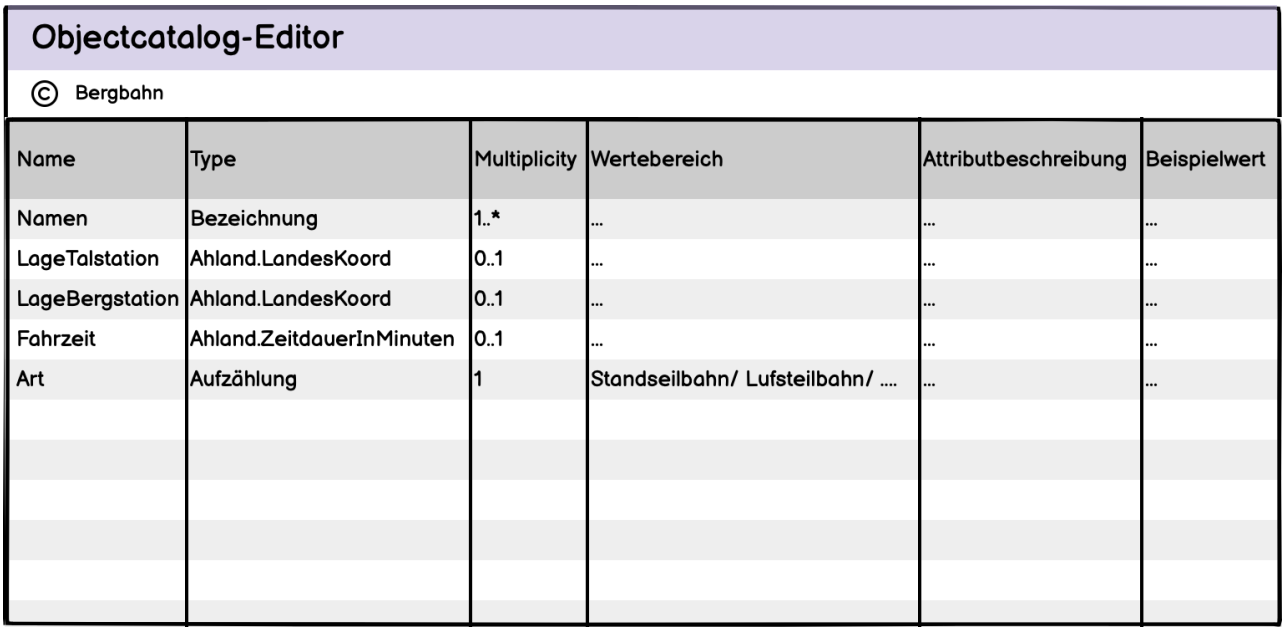

Abbildung 11. Benutzeroberfläche Objektkatalog-Editor (Skizze)

Tabellendarstellung

F9-1 Jede (Assoziations-)Klasse wird als eigene Tabelle angezeigt.

F9-2 Spalten: Attributname, Typ, Multiplizität, Wertebereich.

F9-3 Tabellen unterstützen Sortieren und Filtern pro Spalte. Die Sortierung wirkt sich nicht auf die Modell-Struktur aus, sondern ist lediglich eine Bearbeitungshilfe im Objektkatalog-Editor.

Bearbeitung

F9-4 Zellen sind direkt editierbar.

F9-5 Kontextmenü: „Neues Attribut“, „Löschen“, „Kopieren“.

F9-6 Eigenschaften (z. B. transient, abstract, etc.) werden im Eigenschaften-Editor gepflegt.

F9-7 Pro Attribut können eine ergänzende Beschreibung und ein Beispielwert gepflegt werden. Beschreibung und Beispielwert sind nicht Bestandteil des INTERLIS-Codes.

F9-8 Änderungen werden beim Verlassen der Zelle automatisch übernommen.

F9-9 Ungültige Eingaben (z. B. unbekannter Typ) werden sofort markiert.

F9-10 „Undo/Redo“ ist zwischen dem aktuellen Zustand und der letzten Speicherung unterstützt.

Validierung & Konsistenz

F9-11 Fehlerhafte Zeilen werden optisch auf geeignete Art und Weise erkennbar gemacht (z. B. Markierung mit Symbol und Tooltip).

F9-12 Änderungen werden automatisch in der Semantikprüfung berücksichtigt (vgl. Anforderung F6 in Kapitel 6.8).

F9-13 Synchronisation mit anderen Editoren erfolgt sofort. Die allgemeinen Anforderungen an Synchronisation und Konsistenz sind in Anforderung F2 im Kapitel 6.12 beschrieben.

Benutzerfreundlichkeit

F9-14 Spaltenbreiten und Reihenfolge sind anpassbar.

F9-15 Falls für ein Referenzattribut eine Liste von Instanzen definiert wurde, kann direkt in den Instanz-Editor abgesprungen werden (vgl. Anforderung F11 in Kapitel 6.13).

6.12 Eigenschaften-Editor

Identifikation: F10

6.12.1 User Story

Als Modellierer:in möchte ich die Eigenschaften aller Projekt-Elemente zentral bearbeiten können,
um deren INTERLIS-Auszeichnungen, Prosatexte und Übersetzungen konsistent zu verwalten.

6.12.2 Ziel und Beschreibung

Der Eigenschaften-Editor dient zur Bearbeitung der Metadaten und Auszeichnungen aller Elemente des Projekts. Dies umfasst die INTERLIS-Elemente (TOPIC, CLASS, etc.) aber auch zusätzliche Prosatexte und Diagramme.

Je nach ausgewähltem Element werden nur die dafür gültigen, INTERLIS-konformen Eigenschaften angezeigt. Der Editor erlaubt:

- die Auszeichnung von allgemeinen und spezifischen Eigenschaften gemäss INTERLIS (z. B. ABSTRACT, BAG, LIST, TRANSIENT),
- die Pflege strukturierter Prosatexte,
- die Pflege von Übersetzungen (TRANSLATION OF) mit sprachspezifischen Namen und Beschreibungen.
- die Auswahl der Anzeigesprache für das Modell innerhalb des INTERLIS-Elements MODEL

Der Editor ist vollständig mit Strukturbaum-, Diagramm-, Objektkatalog- und Text-Editor synchronisiert.

6.12.3 Akzeptanzkriterien

Die nachfolgende Abbildung zeigt den Eigenschaften-Editor am Beispiel des Modellelements CLASS. Im linken Bereich sind die Modellelement-spezifischen Eigenschaften skizziert. Im Beispiel die Eigenschaft «Abstract», die für ein CLASS-Element definiert werden kann.

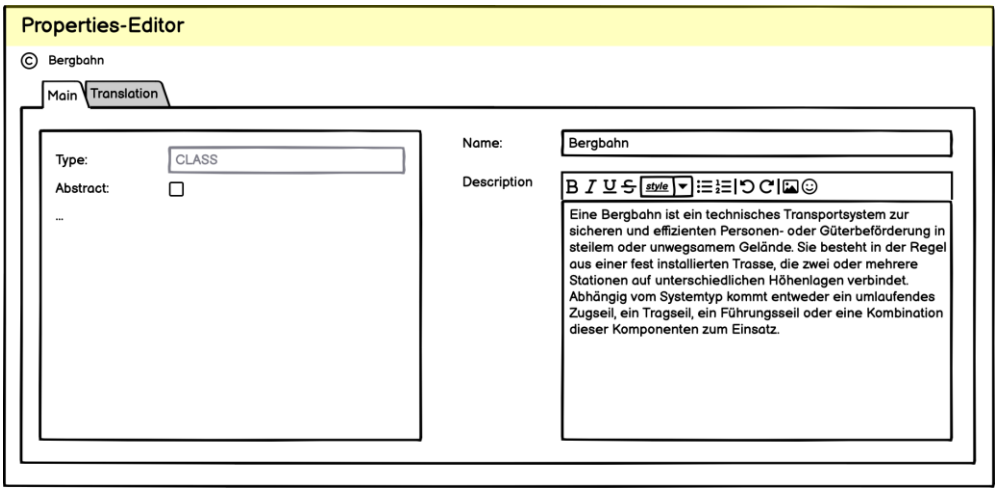

Abbildung 12. Benutzeroberfläche Eigenschaften-Editor – Tab Main (Skizze)

Die nachfolgende Abbildung skizziert die Darstellung von Übersetzungen. Wenn für das Modell Übersetzungen vorhanden sind kann auf der linken Seite die gewünschte Übersetzung ausgewählt und dafür die Übersetzung des Namens und der Beschreibung erfasst werden.

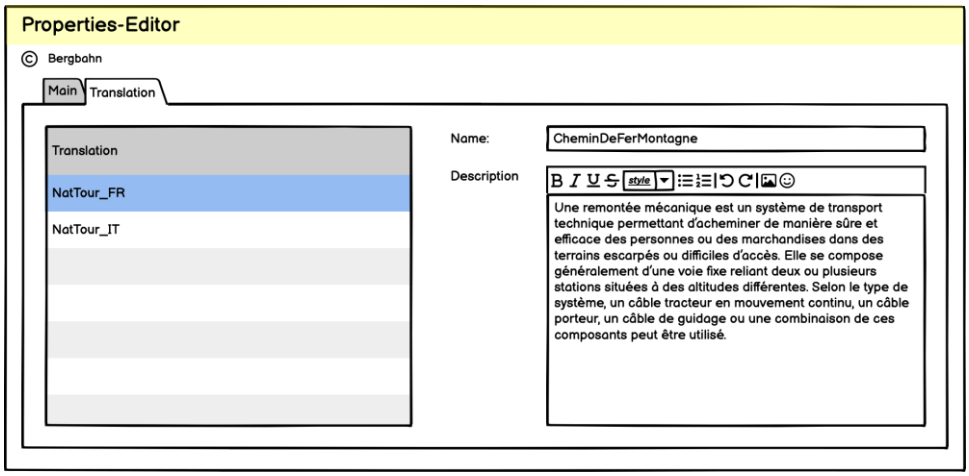

Abbildung 13. Benutzeroberfläche Eigenschaften-Editor – Tab Translation (Skizze)

Die nachfolgende Abbildung zeigt die spezifischen Eigenschaften des Modellelements MODEL. In diesem Bereich können Modellübersetzungen geladen und die Anzeigesprache ausgewählt werden.

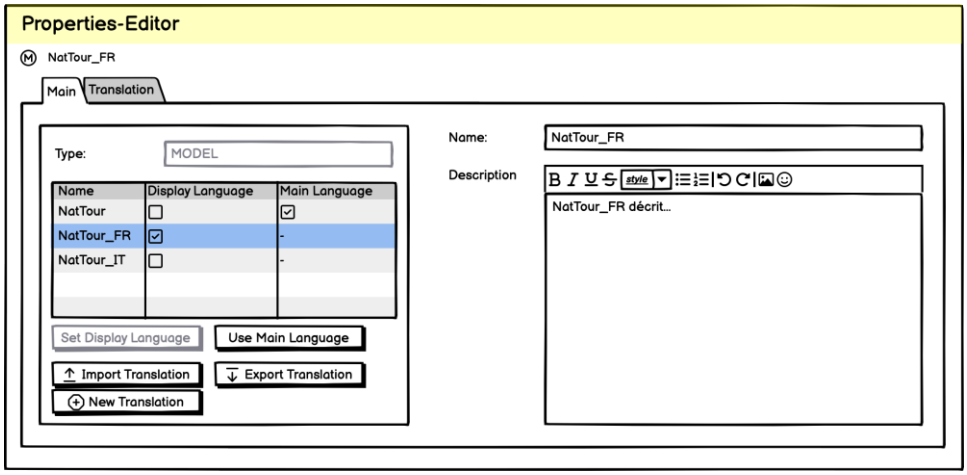

Abbildung 14. Benutzeroberfläche Eigenschaften -Editor – Eigenschaften MODEL (Skizze)

Allgemein

F10-1 Der Eigenschaften-Editor reagiert kontextsensitiv auf das aktuell ausgewählte Modellelement.

F10-2 Es gibt Bearbeitungsbereiche für die Haupteigenschaften (Tab «Main» in der Skizze) und Übersetzungen (Tab «Translation» in der Skizze).

Elementinformationen

F10-3 Anzeige des Elementtyps (z. B. CLASS, ATTRIBUTE, DOMAIN, MODEL).

F10-4 Anzeige und Bearbeitung von Name und einer fachlichen Prosabeschreibung (vgl. Anforderung F12 in Kapitel 6.14).

F10-5 Im Eigenschaften-Editor können die Eigenschaften von per IMPORTS referenzierter Modell-Elementen angezeigt, aber nicht bearbeitet werden.

INTERLIS-Auszeichnungen

F10-6 Es werden nur diejenigen Eigenschaften angezeigt, die für den jeweiligen Elementtyp gültig sind. Beispiele:

- Für Klassen: abstract, final, transient
- Für Attribute: Auswahl der Ausprägung (BAG, LIST)
- Für Assoziationen: Multiplizitäten, Aggregationstypen
- Für Modelle: Version, TRANSLATION OF

F10-7 Bei Attributen können, neben den von INTERLIS vorgegebenen Eigenschaften, zusätzlich eine Beschreibung und ein Beispielwert gepflegt werden (vgl. F9-7 in Kapitel 6.11.3).

F10-8 Die Auswahl erfolgt über geeignete UI-Elemente (Checkboxen, Dropdowns).

Modelleigenschaften - Übersetzungen

F10-9 Bei Elementtyp MODEL kann eine neue Übersetzung (TRANSLATION OF) definiert werden.

F10-10 Beim Anlegen einer neuen Übersetzung werden per Default die bestehenden Bezeichnungen (Name) auf die neue Übersetzung übernommen.

F10-11 Bei Elementtyp MODEL können bestehende Übersetzungen (TRANSLATION OF) als ili-Datei importiert werden.

F10-12 Bei Elementtyp MODEL können bestehende Übersetzungen (TRANSLATION OF) als ili-Datei exportiert werden.

F10-13 Die Hauptsprache, bzw. das Original-Modell ist erkennbar.

F10-14 Als Anzeigesprache kann das Original-Modell oder eine Übersetzung definiert werden.

F10-15 Ist eine Translation als Anzeigesprache ausgewählt, dann werden die Namen der Modellelemente in den Editoren Strukturbaum, Diagramm, Objektkatalog, Instanz in der ausgewählten Übersetzung angezeigt. Zudem werden in den Properties im Bereich «Main» der «Name» und die «Description» der Anzeigesprache ausgewählt. «Name» und «Description» für die andern Übersetzungen, bzw. die Originalsprache können im Reiter «Translation» bearbeitet werden.

Übersetzungsverwaltung/ -bearbeitung

F10-16 Anzeige aller im Modell definierten Sprachen (z. B. NatTour_DE, NatTour_FR).

F10-17 Auswahl einer Sprache zeigt die zugehörigen Felder:

- Übersetzter Name
- Übersetzte Beschreibung (mit gleichem Prosatext-Editor).

F10-18 Es wird geprüft, ob für jede Sprache ein Name und eine Beschreibung vorhanden ist.

F10-19 Fehlende Übersetzungen werden visuell gekennzeichnet.

WYSIWYG-Editor und Bearbeitung

F10-20 Prosabeschreibungen können über einen WYSIWYG-Editor erfasst werden, der die definierten Auszeichnungen der gewählten Auszeichnungssprache unterstützt. Prosatexte können als eigene Prosatext-Elemente erfasst werden (vgl. F7-10 in Kapitel 6.9.3) oder direkt zu einem beliebigen Modell-Element.

F10-21 Den Benutzer:innen wird direkt die Formatierung und Struktur des erfassten Textes angezeigt.

Benutzerfreundlichkeit

F10-22 Übersichtlich gruppierte Eigenschaften je Elementtyp.

F10-23 Tooltips erklären jede Eigenschaft und ihre INTERLIS-Bedeutung.

F10-24 Nicht verfügbare Eigenschaften werden ausgeblendet.

F10-25 Pflichtangaben sind visuell hervorgehoben.

F10-26 Synchronisation mit anderen Editoren erfolgt sofort. Die allgemeinen Anforderungen an Synchronisation und Konsistenz sind in Anforderung F2 im Kapitel 6.4 beschrieben.

6.13 Instanz-Editor

Identifikation: F11

6.13.1 User Story

Als Modellierer:in möchte ich Instanzen von definierten Klassen tabellarisch erfassen und bearbeiten können,

um «externe Kataloge» auf Basis des Modells einfach aufbauen oder Testdaten erfassen zu können.

6.13.2 Ziel und Beschreibung

Der Instanz-Editor dient der Erfassung und Pflege von Instanzen für bereits definierte Klassen. Er ist auf die Pflege konkreter Kataloginhalte (Objekte mit Attributwerten) ausgerichtet.

Prämissen der ersten Ausbaustufe:

- Instanzen können nur für Klassen ohne Unterstrukturen (keine verschachtelten Klassen) angelegt werden. Keine mehrstufigen/ hierarchischen Kataloge.
- Keine Validierungen durch den IMC bei der Bearbeitung – alle Attributwerte werden als Text behandelt.
- Beim Import und Export über xtf/ xml-Dateien werden die Inhalte mittels ilivalidator validiert.

6.13.3 Akzeptanzkriterien

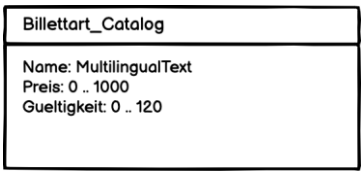

Abbildung 15. Klassendefinition als Basis für die Pflege von Instanzen

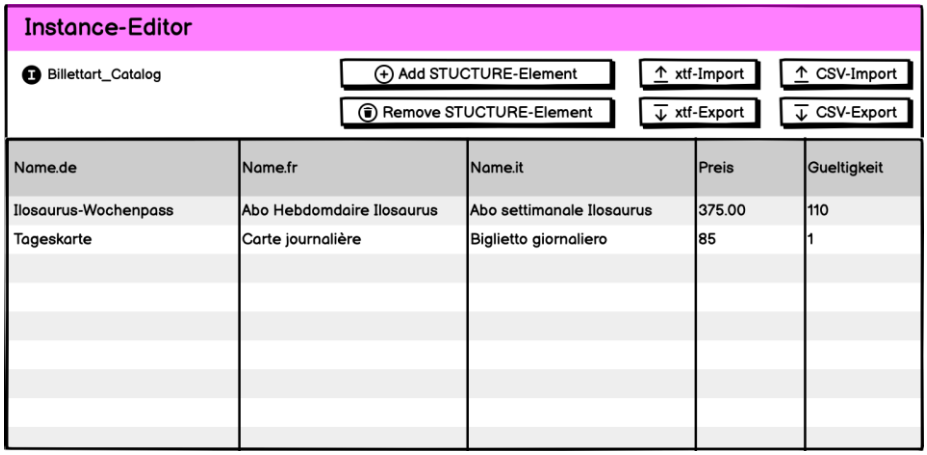

Abbildung 16. Benutzeroberfläche Instanz-Editor (Skizze)

Tabellendarstellung

F11-1 Instanzen einer Klasse werden in Tabellenform dargestellt.

F11-2 Spalten entsprechen den Attributen der Klasse.

F11-3 Jede Zeile repräsentiert eine Instanz.

F11-4 Datentypen werden angezeigt, aber initial nicht streng durchgesetzt.

Bearbeitung

F11-5 Instanzen können erstellt, geändert und gelöscht werden (CRUD).

F11-6 Zellen sind direkt editierbar, Eingabe zunächst als Freitext.

F11-7 Kontextmenü: „Neue Instanz“, „Löschen“, „Kopieren“.

F11-8 Import von Instanzen über CSV möglich.

F11-9 „Undo/Redo“ ist zwischen dem aktuellen Zustand und der letzten Speicherung unterstützt.

Validierung & Konsistenz

F11-10 Beim Import und Export erfolgt eine Validierung mittels ilivalidator aufgrund der Klassendefinition.

F11-11 Bei der Bearbeitung der Daten im Editor erfolgt keine Validierung mittels ilivalidator.

Export & Import

F11-12 Import und Export mittels CSV-Datei ist möglich.

F11-13 Anstelle vom CSV kann auch ein Import/ Export mittels Excel angeboten werden, z. B. unter Nutzung der Komponenten xls2xtf.

F11-14 Beim Import mittels CSV-Datei muss angegeben werden, für welches CLASS-Element der Import erfolgen soll.

F11-15 Eine CSV-Datei kann auch exportiert werden, wenn (noch) keine Instanzen definiert sind. Damit kann aus der Applikation heraus eine «Erfassungstabelle» bereitgestellt werden.

F11-16 Import und Export mittels xtf/ xml-Datei ist möglich.

F11-17 Beim Import wird die Struktur geprüft und die Inhalte werden auf Duplikate geprüft.

Sprachunterstützung

F11-18 Für Attribute vom Typ STRUCTURE können weitere Spalten, bzw. STRUCTURE-Elemente hinzugefügt werden. Es muss ein Elementname definiert werden. Die Spaltenbezeichnung wird wie folgt zusammengesetzt: Attributname + . + Elementname. Beispiele: «Name.de», «Name.it»

F11-19 Für Attribute vom Typ STRUCTURE können durch Benutzer Spalten entfernt werden. Nach Bestätigung einer Warnmeldung wird die Spalte mit allen erfassten Werten entfernt.

Benutzerfreundlichkeit

F11-20 Gleiche Bedienlogik wie Objektkatalog-Editor.

F11-21 Unterstützung für Sortieren und Filtern.

F11-22 Tooltips zeigen die zugehörigen Attributdefinitionen aus dem Objektkatalog.

F11-23 Visuelle Kennzeichnung von Pflichtattributen.

6.14 Fachliche Prosabeschreibungen

Identifikation: F12

6.14.1 User Story

Als Modellierer:in möchte ich fachliche Prosabeschreibungen für das gesamte INTERLIS-Modell, als auch für einzelne Elemente wie Klassen oder Attribute erfassen,

damit ich das Modell für alle Nutzer:innen verständlich erklären kann.

6.14.2 Ziel und Beschreibung

Mit der fachlichen Prosabeschreibung können Modelle und Modellelemente mit zusätzlichen Erklärungen ergänzt werden. Die Prosabeschreibungen sind direkt im «Core Model» gespeichert und können als eigenständige Textelemente exportiert werden.

Diese Textelemente können sowohl losgelöst von INTERLIS-Modellelementen sein als auch einen Bezug zu spezifischen INTERLIS-Elementen (z. B. Klassen oder Attribute) haben.

Die Prosabeschreibungen werden in einer Auszeichnungssprache wie Asciidoc oder Markup erfasst. Damit wird eine strukturierte und anpassbare Formatierung ermöglicht.

Prosatexte umfassen insbesondere erläuternde Beschreibungen von Klassen, Attributen und Beziehungen, fachliche Begründungen und Hintergrundinformationen, Grafiken wie UML-Diagramme, Übersichtskarten oder Illustrationen sowie Hinweise zur Anwendung und Interpretation des Modells. Sie können Bezüge zu gesetzlichen Grundlagen (z. B. Verordnungen, Weisungen oder technische Richtlinien) herstellen, fachliche Definitionen und Interpretationsregeln für Modellelemente festhalten sowie Vorgaben zur Datenerhebung, -qualität und -pflege dokumentieren, die sich nicht vollständig in INTERLIS ausdrücken lassen.

Speicherung und Persistenz: Prosatexte werden unabhängig vom INTERLIS-Modell gespeichert und gepflegt. Sie sind nicht als Kommentare oder Annotationen innerhalb der Modelldatei (.ili) abgelegt, sondern werden eigenständig persistiert. Die inhaltliche Zuordnung zu den jeweiligen Modellelementen erfolgt über eindeutige Referenzen (z. B. vollqualifizierte Elementnamen). Änderungen am INTERLIS-Modell erfordern eine entsprechende Überprüfung und Anpassung der zugehörigen Prosatexte, um die Konsistenz zwischen formalem Modell und beschreibender Dokumentation sicherzustellen.

6.14.3 Akzeptanzkriterien

Speicherung und Struktur

F12-1 Die Prosabeschreibungen werden in der gewählten Auszeichnungssprache geschrieben und im «Core Model» separiert vom INTERLIS-Code gespeichert.
Hinweis: Eine Integration in das INTERLIS-Modell mittels Kommentare (ilidoc) wird nicht in Betracht gezogen, damit der INTERLIS-Code möglichst übersichtlich bleibt.

F12-2 Die Textelemente können losgelöst vom INTERLIS-Modell erfasst und gespeichert werden oder mit spezifischen INTERLIS-Modellelementen verknüpft werden.

F12-3 Als primäres Erfassungsformat für Prosatexte wird ein textbasiertes Format festgelegt (AsciiDoc oder Markdown; präferiert: AsciiDoc, aufgrund des grösseren Sprachumfang für komplexere Dokumente).

F12-4 Textelemente, die mit den INTERLIS-Modellelementen verknüpft sind, müssen eindeutig identifiziert werden können, damit sie bei Änderungen im Modell automatisch aktualisiert werden können.

F12-5 Werden Modellelemente gelöscht, dann werden auch die zugehörigen Textelemente gelöscht.

Bearbeitung

F12-6 Die Bearbeitung von Prosatexten erfolgt über den Eigenschaften-Editor (vgl. Anforderung F10 in Kapitel 6.12).

F12-7 Prosatexte müssen für alle Übersetzungen (vgl. Anforderung F10 in Kapitel 6.12) erfasst werden können.

F12-8 In allen Editoren kann zu einem Element in die Erfassung für Prosatexte abgesprungen werden (z. B. über Kontextmenu).

Einbindung und Wiederverwendung

F12-9 Textelemente können verschachtelt werden und nutzen dafür die in der gewählten Auszeichnungssprache definierten Anweisungen (z. B. «include»).

F12-10 Die Dateien mit Prosabeschreibungen werden ebenfalls in die Versionsverwaltung mit eingebunden und sind damit Teil des gesamten «Core Models».

6.15 Versionsverwaltung und Kollaboration

Identifikation: F13

6.15.1 User Story

Als Modellierer:in möchte ich verschiedene Modell- und Dateiversionen vergleichen, zusammenführen und verwalten sowie Änderungen kommentieren und im Team reviewen,

um Änderungen nachvollziehen, ältere Stände wiederherstellen und kollaborativ an Modellen arbeiten zu können.

6.15.2 Ziel und Beschreibung

Der IMC nutzt Git als Versionskontrollsystem und GitHub/GitLab als Kollaborationsplattform.

Git verwaltet unterschiedliche Zustände des Projekts und aller zugehörigen Dateien (xtf/ xml-Dateien, Textdateien, Diagramme). Es ermöglicht den Vergleich verschiedener Versionen (Diff) sowie deren Zusammenführung (Merge) und macht Änderungen durch Commit-Historie nachvollziehbar.

Die Kollaboration erfolgt über GitHub oder GitLab: Review-Workflows, Issue-Tracking, Kommentierung und Team-Kommunikation machen den Entstehungsprozess transparent und ermöglichen effiziente Zusammenarbeit.

Die Lösung basiert auf bestehenden, getesteten Plugins.

6.15.3 Akzeptanzkriterien

Kapselung GIT-Funktionen für vereinfachte Bedienung

F13-1 Die IMC-Applikation stellt eine Funktion „Aktuellen Stand im Repository speichern“. Diese Funktion führt folgende Schritte automatisch aus:

- Speichern der aktuellen .imcp-Datei
- Speichern aller zugehörigen Projektdateien (z. B. xtf/ xml, ili, Textdokumente, Diagramme)
- Erstellen eines lokalen GIT-Commits
- Pushen ins GIT-Remote-Repository

F13-2 Die Funktion ist z. B. über die Menü-Toolbar erreichbar. Vor dem Commit muss eine einfache Commit-Nachricht eingegeben werden. Es werden verständliche, fachlich formulierte Bezeichnungen verwendet (z. B. „Projektstand sichern“, „Änderungen speichern“).

F13-3 Automatische Änderungserfassung: Alle seit dem letzten Commit geänderten Dateien werden automatisch erkannt und in das Commit aufgenommen. Evtl. eine kompakte Übersicht der betroffenen Dateien wird angezeigt.

F13-4 Die Funktion kann nur bei bestehender Internetverbindung ausgeführt werden.

F13-5 Fehlerbehandlung & Rückmeldung: Fehlersituationen werden in verständlicher Sprache angezeigt (z. B. kein Repository vorhanden, keine Netzwerkverbindung, fehlende Authentifizierung). Erfolgs- und Warnmeldungen sind klar formuliert (z. B. Projektstand lokal gesichert, Speichern im Repository fehlgeschlagen).

F13-6 Abgrenzung: Diese gekapselte Funktion ergänzt, ersetzt aber nicht die erweiterten GIT-Funktionen der IMC-Plugins. Fortgeschrittene GIT-Funktionen bleiben vollständig verfügbar.

6.16 Dokumentationsgenerierung

Identifikation: F14

6.16.1 User Story

Als Modellierer:in möchte ich aus dem INTERLIS-Modell und den zugehörigen fachlichen Prosabeschreibungen automatisch eine vollständige Dokumentation generieren,

um Inhalte in HTML-, PDF-, Markdown- oder Word- (.docx) Form zur Veröffentlichung oder internen Abstimmung zu nutzen.

6.16.2 Ziel und Beschreibung

Die Dokumentationsgenerierung erstellt aus dem INTERLIS-Modell und den zugehörigen Prosabeschreibungen automatisch eine strukturierte Dokumentation mit Inhaltsverzeichnis, Klassendiagrammen, Tabellen und Beschreibungen.

Die Prosabeschreibungen können sowohl losgelöst vom Modell existieren als auch mit spezifischen Modellelementen verknüpft sein. Verschachtelte Textelemente (z. B. via „include“ in Asciidoc oder Markdown) werden korrekt aufgelöst.

Die Dokumentation kann in gängigen Ausgabeformaten (z. B. PDF, .docx (Word), HTML, Markdown) erzeugt werden und berücksichtigt die Versionierung der Modell- und Textelemente.

Die Dokumentation enthält Metadaten (Autor, Version, Datum), alle Modellinhalte (Klassen, Attribute, Beziehungen) und Links zu externen Ressourcen.

Abgrenzung:

Die Verwaltung und Bearbeitung eines Dokumentlayouts ist nicht Teil dieser User Story. Die Erstellung des Dokumentlayouts oder allfällige Anpassungen in der Kapitelstruktur können ausserhalb des IMC vorgenommen werden, z.B. in einem aus dem IMC generierten Word-Dokument.

6.16.3 Akzeptanzkriterien

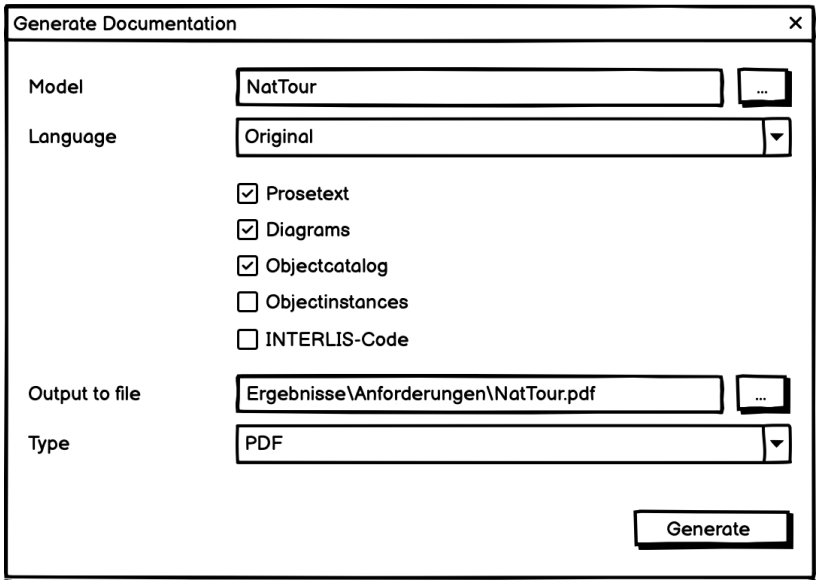

Abbildung 17. Benutzeroberfläche Dialog Dokumentgenerierung (Skizze)

Dokumentinhalt

F14-1 Jede Dokumentation enthält Kopfzeile mit Modellname, Version, Autor, Datum.

F14-2 Klassen, Attribute, Beziehungen, Constraints und Listen von Instanzen werden tabellarisch dargestellt.

F14-3 Das generierte Dokument enthält per Default automatisch alle Modellelemente (Strukturen, Klassen, Attribute), Grafiken sowie die zugehörigen fachlichen Prosabeschreibungen (vgl. Kapitel 6.14).

F14-4 Die Benutzer:in kann auswählen, welche Modellelemente in der Dokumentation ausgegeben werden.

F14-5 Beim Generieren der Dokumentation wird ein Disclaimer erzeugt und im Dokument ausgegeben, in dem die von der Dokumentation ausgeschlossenen Modellelemente namentlich aufgeführt werden.

F14-6 Mit Modellelementen verknüpfte Textelemente werden in der Dokumentation als Teil der Dokumentation des Modellelements ausgegeben, sofern ausgewählt.

F14-7 Die Benutzer:in kann im Generierungsdialog auswählen, welche Inhalte in das Dokument aufgenommen werden:

- Prosatexte
- Diagramme
- Objektkataloge, inkl. Attributbeschreibungen und Beispielwerte (vgl. F9-7 in Kapitel 6.11.3)
- Instanzen
- INTERLIS-Code

F14-8 Nur die ausgewählten Inhalte werden in die Dokumentstruktur und den Umfang des Dokuments übernommen.

F14-9 Die Einstellungen für den Dokumentinhalt werden in den Projekteigenschaften gespeichert.

F14-10 Verschachtelte Textelemente werden korrekt aufgelöst.

F14-11 Inhaltsverzeichnis und interne Hyperlinks sind automatisch generiert.

Dokumentstruktur

F14-12 Die Kapitelstruktur des Dokuments entspricht der definierten Struktur für die Dokumentgenerierung.

F14-13 Reihenfolge der Modellelemente können für die Dokumentgenerierung in einer beliebigen Struktur festgelegt werden.
Hinweis: Es ist in der Verantwortung der Benutzer:in eine sinnvolle Struktur auszuwählen und darauf zu achten, dass die Modellstruktur nicht verletzt wird.

F14-14 Die Einstellungen der Dokumentstruktur werden in den Projekteigenschaften gespeichert.

Formate und Export

F14-15 Exportformate: html, pdf, docx

F14-16 Alle aus Prosatexten abgeleiteten Ausgabeformate werden direkt oder indirekt aus dem Erfassungsformat (z.B AsciiDoc) generiert.

F14-17 Die Benutzer:in kann im Generierungsdialog das Ausgabeformat festlegen.

F14-18 Dokumentation kann pro Modell erstellt werden. Der Generierungsdialog zeigt das aktuell ausgewählte Modell an und erlaubt den Wechsel auf ein anderes Modell, dass im IMC-Projekt enthalten ist.

F14-19 Grafiken mit ihren Prosabeschreibungen (vgl. F8 und F12 in Kapiteln 6.10 und 6.14) werden eingebettet, wenn Diagramm und Prosatexte als Bestandteil der Dokumentation markiert sind.

F14-20 Die Benutzer:in kann im Dialog den Ausgabepfad und Dateinamen frei wählen und den Dateinamen anpassen.

F14-21 Die Sprache der Dokumentation ist im Dialog auswählbar (z. B. „Original“, alternative Sprachversionen oder «Alle»)

Performance und Inhaltliche Qualität

F14-22 HTML-Version enthält Suchfunktion.

F14-23 Generierung dauert < 2 Minuten für ein mittleres Modell (50 Klassen, 500 Attribute, 10 Grafiken)

Bedienung und Benutzerführung

F14-24 Die Dokumentation wird über einen zentralen Dialog „Dokumentation generieren“ gestartet.

F14-25 Nach Abschluss der Generierung wird das Dokument im gewählten Ausgabeordner gespeichert.

F14-26 Fehlende Pflichtangaben (z. B. Ausgabepfad) werden vor der Generierung validiert und dem/der Benutzer:in angezeigt.

6.17 Suche, Filterung und Navigation

Identifikation: F15

6.17.1 User Story

Als Modellierer:in möchte ich schnell bestimmte Elemente im INTERLIS-Modell finden und direkt zu ihnen navigieren können,

um grosse Modelle effizient zu bearbeiten und zu verstehen.

6.17.2 Ziel und Beschreibung

Die Such-, Filter- und Navigationsfunktionen ermöglichen ein schnelles Auffinden und gezieltes Springen zu Klassen, Topics, Domains, Attributen, Beziehungen oder Prosatexten.

Die Suchleiste unterstützt Volltextsuche und Filter nach Typ.

Gefundene Elemente werden in Editor, Strukturbaum und Diagramm synchron hervorgehoben.

6.17.3 Akzeptanzkriterien

Die Such-, Filter- und Navigationsfunktionen müssen nicht individuell entwickelt werden, sondern müssen als Bestandteil des eingesetzten Frameworks (z. B. VS Code, Theia) genutzt und integriert werden. Diese Funktionen müssen die nachfolgenden Anforderungen erfüllen.

Suchfunktionen

F15-1 Eine globale Suchleiste erlaubt die Volltextsuche über alle Modelle.

F15-2 Suchergebnisse werden mit Typ, Name, Pfad und Zeilennummer angezeigt.

F15-3 Treffer können direkt per Klick geöffnet werden.

F15-4 Suche unterstützt Platzhalter.

F15-5 Suchbegriffe werden automatisch gespeichert (Suchhistorie).

Filterfunktionen

F15-6 Ergebnisse können nach Elementtyp (Class, Attribute, Domain, Association) gefiltert werden.

F15-7 Filterung kann auf aktuelle Ansicht (z. B. Topic) beschränkt werden.

F15-8 Kombination von Suchtext und Filter ist möglich.

F15-9 Filtereinstellungen werden im Projekt gespeichert.

Navigation

F15-10 Klick auf ein Suchergebnis lädt das ausgewählte Element in den geöffneten Editoren, in denen das Element angezeigt werden kann.

F15-11 Strukturbaum scrollt automatisch zum gewählten Element.

F15-12 Navigation zwischen letzter und aktueller Position ist möglich (Vor/Zurück).

F15-13 Tastenkürzel (z. B. Ctrl+F, F3) sind implementiert.

Leistung & Benutzerfreundlichkeit

F15-14 Suche liefert Ergebnisse in unter 1 Sekunde bei 10 000 Elementen.

F15-15 Anzeige ist auch während laufender Suche reaktionsfähig.

F15-16 Trefferanzahl wird angezeigt.

F15-17 Such- und Filterfenster kann angedockt oder frei platziert werden.

7 IDE- und Architekturanforderungen

Die folgenden Architektur- und IDE-Anforderungen konkretisieren die technischen Prinzipien, Strukturen und Schnittstellen zur Umsetzung der funktionalen Anforderungen (F1–F15) und Rahmenbedingungen (R1–R5).

Sie definieren **wie** der IMC technisch realisiert wird und stellen verbindliche Architekturentscheidungen und Qualitätsziele dar (Erweiterbarkeit, Modularität, Performance, Integrationsfähigkeit), die als Grundlage für Design, Implementierung und Wartung dienen.

Die Anforderungen sind nach **Architekturthemen** gruppiert und priorisiert für **Phase 1**.

7.1 Übersicht der Anforderungen

Nr.	Bezug	Bezeichnung	Beschreibung
Plattform und Deployment			
A1	R2, R5	IDE-Plattform und Deployment-Architektur	Webbasierte IDE (Priorität 1) und Desktop (Priorität 2) auf Basis einer geeigneten Plattform (z.B. Eclipse Theia). Plattformunabhängige Ausführung.
A2	R5	Modulare, erweiterbare Plugin-Architektur	Plugin-System zur Integration bestehender und zukünftiger Erweiterungen (Git, Formatter, etc.) ohne Kernänderungen.
Language Server und Compiler			
A3	F5, F6	Language Server Protocol (LSP)-Architektur	Trennung von UI (Client) und Sprachlogik (Server) über LSP. Syntax-Highlighting, Autovervollständigung, Fehlerdiagnostik.
A4	F5, F6, R1	ili2c-Integration und Compiler-Schnittstelle	Einbettung des bestehenden INTERLIS-Compilers ili2c für Syntax- und Semantikprüfung, INTERLIS-Modell-Repository-Zugriff.
Core Model und Synchronisation			
A5	F2	Core Model und Event-System	Single Source of Truth für das INTERLIS-Modell. Event-basierte Synchronisation zwischen allen Editoren und Ansichten.
A6	F1, F2, F3, F8, F12	Persistenz- und Serialisierung	Alle Artefakte in einer .imcp-Projektdatei. Import/Export einzelner Formate (.ili, .xtf/ xml) über F3.
UI-Komponenten und Editoren			
A7	F5	Texteditor-Komponente mit INTERLIS-Unterstützung	LSP-Client basierend auf Monaco Editor. Delegierung aller Sprachfunktionen an INTERLIS Language Server.
A8	F7	Strukturbaum-Komponente	Model-to-Tree-Adapter mit bidirektionaler Synchronisation. Bidirektionale, event-basierte Aktualisierung über das Core Model.
A9	F8	Grafische Visualisierungsschicht (Diagramm-Editor)	UML-ähnliche Darstellung mit Diagramm-Engine. Zoom, Pan, Auto-Layout. Bidirektionale, event-basierte Aktualisierung über das Core Model.
A10	F9, F11	Tabelleneditor und Grid-Framework	Grid-Komponente für Objektkataloge und externe Kataloge. CSV/XTF/ XML-Import/Export.
A11	F12	Dokumentations- und Prosa-Management	WYSIWYG-Editor für AsciiDoc/Markdown. Verlinkung zwischen Prosabeschreibungen und Modellelementen.
Versionierung			

Nr.	Bezug	Bezeichnung	Beschreibung
A12	F13	Git-Integration und Versionskontrolle	Nutzung bestehender Git-Plugins (z. B. Eclipse Theia). Diff, Merge, Commit-Historie, Branch-Management, GitHub/GitLab-Anbindung.
Export und Generierung			
A13	F14	Generierungs- und Export-Pipeline	Template-basierte Dokumentations-generierung (HTML, PDF, Markdown/AsciiDoc)..
Suche und Navigation			
A14	F15	Such- und Indexiersystem	Inkrementelle Indexierung. Volltextsuche über Modelle. Filter nach Typ, Kontext. Integration bestehender Suchfunktionen (z. B. Eclipse Theia).
Querschnittsanforderungen			
A15	R4	Internationalisierung und Lokalisierung (i18n/l10n)	Mehrsprachigkeit über Resource-Bundles. Englisch als Standardsprache. Erweiterbar für weitere Sprachen.
A16	R3, R5	Fehlerbehandlung, Logging und Debugging	Strukturiertes Logging. Error-Reporting. Debug-Unterstützung.
A17	R5	Performance und Skalierung	Lazy Loading, inkrementelle Updates, asynchrone Operationen. Performance-Budgets für grosse Modelle (>10.000 Zeilen, >50 Klassen).

7.2 IDE-Plattform und Deployment-Architektur

Identifikation: A1

Bezug: R2 (Browser + Desktop), R5 (Modularität)

7.2.1 Ziel und Beschreibung

Der IMC basiert auf einer modernen, webbasierten Plattform, die sowohl im Browser (Priorität 1) als auch als Desktop-Anwendung (Priorität 2) ausgeführt werden kann.

Als Plattform wird Eclipse Theia eingesetzt, da es Plugin-Ökosysteme bereitstellt, LSP unterstützt, Browser- und Desktop-Deployment ermöglicht und vollständig Open Source ist (R3).

7.2.2 Akzeptanzkriterien

Plattformauswahl

A1-1 Der IMC basiert auf einer geeigneten Plattform, z.B. Eclipse Theia (inkl. Monaco Editor).

A1-2 Plattformwahl ist dokumentiert und begründet (Browser-First inkl. Hosting, Plugin-Ökosystem, Community-Support).

Browser-Deployment (Priorität 1)

A1-3 IMC ist über moderne Browser (Chrome, Firefox, Edge, Safari) ohne Installation nutzbar.

A1-4 Hosting über Webserver (z. B. nginx, Apache) möglich.

Desktop-Deployment (Priorität 2)

A1-5 IMC kann als eigenständige Desktop-Anwendung (Windows, macOS, Linux) bereitgestellt werden.

A1-6 Wenn keine reine Web-Lösung realisiert wird, dann muss für Aufrufe gegen aussen (z.B. Zugriff auf Model-Repository) ein Proxy hinterlegt werden können.

Plattformunabhängigkeit

A1-7 Kernfunktionen sind plattformunabhängig.

A1-8 Plattformspezifische Anpassungen sind minimal und gekapselt.

Erweiterbarkeit

A1-9 Extension-API ist verfügbar für zukünftige Erweiterungen.

7.3 Modulare, erweiterbare Plugin-Architektur

Identifikation: A2

Bezug: R5 (Modularität)

7.3.1 Ziel und Beschreibung

Der IMC nutzt das Theia-Extension-System, um bestehende Funktionen (GIT, Formatierung, Debugging) zu integrieren und zukünftige Erweiterungen ohne Änderung am Kerncode zu ermöglichen.

Kern- und Erweiterungsfunktionen sind klar entkoppelt und kommunizieren über definierte APIs oder Events.

7.3.2 Akzeptanzkriterien

Modulsystem

A2-1 IMC nutzt das native Plugin-/Extension-System Eclipse Theia.

A2-2 Editoren und Komponenten sind möglichst lose gekoppelt, um eine Weiterentwicklung ohne Seiteneffekte zu ermöglichen.

API-Stabilität

A2-3 Kern-APIs sind versioniert und abwärtskompatibel.

A2-4 Breaking Changes erfordern Major-Version-Update und sind dokumentiert.

Plugin-Integration

A2-5 Bestehende Plugins (z. B. GIT, Markdown Preview) können ohne Anpassung genutzt werden.

A2-6 Eigene Extensions können über Extension-API entwickelt werden.

Dependency Management

A2-7 Plugin-Abhängigkeiten sind explizit deklariert.

A2-8 Konflikte werden zur Build-Zeit erkannt.

Gemeinsame Services

A2-9 Logging, Settings, i18n sind als zentrale Services verfügbar.

A2-10 Alle Plugins nutzen dieselben Services (keine Duplikate).

7.4 Language Server Protocol (LSP)-Architektur

Identifikation: A3

Bezug: F5 (Text-Editor), F6 (Struktur-/Semantikprüfung)

7.4.1 Ziel und Beschreibung

Der IMC implementiert eine Language-Server-Architektur (LSP) zur Syntax- und Strukturprüfung von INTERLIS. Sie trennt Darstellung (Client) und Sprachlogik (Server) über das LSP-Protokoll (≥ v3.17).

Der INTERLIS Language Server läuft als eigenständiger Prozess und bietet:

- Syntax-Highlighting
- Autovervollständigung
- Fehlerdiagnostik (Syntax + Semantik)
- Hover-Informationen
- Navigation (Go to Definition, Find References)

Kommunikation erfolgt über JSON-RPC 2.0 (asynchron, nicht-blockierend).

7.4.2 Akzeptanzkriterien

LSP-Konformität

A3-1 Language Server implementiert LSP ≥ v3.17.

A3-2 Kommunikation über JSON-RPC 2.0 (stdin/stdout oder WebSocket).

A3-3 Unterstützte Requests:

Request / Notification	Beschreibung
initialize	Vom Client gesendet, um den Server zu initialisieren. Der Server antwortet mit seinen Fähigkeiten (z. B. ob er Completion unterstützt).
initialized	Notification nach erfolgreicher Initialisierung.
shutdown	Der Client fordert den Server auf, sich ordentlich zu beenden.
exit	Schliesst den Prozess.
textDocument/didOpen	Wird gesendet, wenn eine Datei geöffnet wird.
textDocument/didChange	Wenn sich der Inhalt eines Dokuments ändert.
textDocument/didClose	Wenn das Dokument geschlossen wird.
textDocument/publishDiagnostics	Notification vom Server → Client, um Fehler/Warnungen zu melden.
textDocument/completion	Code-Vervollständigungen.
textDocument/hover	Tooltip mit Typinformationen oder Doku beim Überfahren.
textDocument/definition	Navigation zur Definition eines Symbols.
textDocument/references	Alle Referenzen eines Symbols finden.
textDocument/documentSymbol	Liste aller Symbole im aktuellen Dokument.
workspace/symbol	Symbolsuche im gesamten Workspace.
textDocument/semanticTokens	Syntax-Highlighting auf semantischer Ebene.

Prozessarchitektur

A3-4 Language Server läuft als separater Prozess (ein Server pro IMC-Instanz).

A3-5 Server-Start/Stop wird von IMC gesteuert.

A3-6 Bei Server-Absturz: Automatischer Neustart mit Fehler-Logging.

Performance

A3-7 Diagnostik-Antwort in <1 Sekunde nach Codeänderung (bei Dateien bis 10.000 Zeilen).

A3-8 Completion-Vorschläge in <300ms.

A3-9 Asynchrone Verarbeitung verhindert UI-Blockaden.

Erweiterbarkeit

A3-10 Language Server ist austauschbar ohne Client-Änderung.

A3-11 Neue INTERLIS-Versionen erfordern nur Server-Update.

7.5 ili2c-Integration und Compiler-Schnittstelle

Identifikation: A4

Bezug: F1 (Syntax), F2 (Semantik), R1 (Bestehende Werkzeuge einbinden)

7.5.1 Ziel und Beschreibung

Der bestehende INTERLIS-Compiler ili2c (Java-basiert) wird als Backend-Komponente genutzt für:

- Syntaxprüfung gemäss INTERLIS-Sprachversion
- Semantikprüfung (Beziehungen, Multiplizitäten, Domains, etc.)
- Zugriff auf öffentliche INTERLIS-Repositories für Modell-Imports

Herausforderung: Browser-Deployment vs. Java

ili2c ist eine Java-Bibliothek, die nicht nativ im Browser läuft (R2 Priorität 1: Browser-Deployment).

Es existieren grundsätzlich folgende Ansätze:

Einbettungsstrategie	Vorteile	Nachteile
1: Remote Language Server (Browser ↔ Backend-Server)	• Volle ili2c-Funktionalität • Wartbar • Keine Java-Konvertierung	• Erfordert Server • Offline nur Syntax-Highlighting • Session Handling
2: WebAssembly (WASM) (Java → WASM im Browser)	• Echte Browser-Lösung • Offline-fähig	• Experimentell (TeaVM/CheerpJ) • Dateigrösse >5 MB
3: Webfähige Neu-Implementierung (Langium/Rust)	• Echte Browser-Lösung • Offline-fähig • Moderne Technologie	• **Widerspricht R1 (Wiederverwendung)** • INTERLIS-Grammatik neu implementieren • Aufwändig für Feature-Parität und Kompatibilität
4: Desktop-First (Embedded JVM)	• Volle Funktionalität • Offline-fähig	• **Widerspricht R2 (Browser P1)** • Komplexe, aufwändige Distribution

7.5.2 Akzeptanzkriterien

Compiler-Integration

A4-1 ili2c ist im Language Server eingebettet.

A4-2 Konkrete Einbettungsstrategie ist mit Auftraggeberin abgestimmt, dokumentiert und begründet. Hinweis: Aufgrund der Verletzung der Rahmenbedingungen stehen die oben aufgeführten Einbettungsstrategien 3 und 4 nicht zur Auswahl.

Versionsverwaltung

A4-3 ili2c-Version ist konfigurierbar.

A4-4 IMC kann gegen verschiedene INTERLIS-Sprachversionen prüfen (2.3, 2.4).

Repository-Zugriff

A4-5 Language Server kann INTERLIS-Modelle aus öffentlichen Repositories laden (https://models.interlis.ch).

A4-6 Offline-Modus mit lokal gecachten Modellen ist unterstützt.

A4-7 Fehlende Modelle werden mit klarer Fehlermeldung angezeigt.

Fehlerbehandlung

A4-8 ili2c-Fehler werden in LSP-Diagnostics konvertiert (Zeile, Spalte, Severity, Nachricht).

A4-9 Schwere Fehler verhindern nicht das Öffnen/Bearbeiten der Datei.

Performance

A4-10 Prüfung eines mittleren Modells (50 Klassen, 500 Attribute, 10 Imports) in <3 Sekunden.

7.6 Core Modell und Event-System

Identifikation: A5

Bezug: F2 (Synchronisation)

7.6.1 Ziel und Beschreibung

Der IMC basiert auf einem zentralen Modell (Core Model) als Single Source of Truth. Alle Editoren (Text, Baum, Diagramm, Tabelle) arbeiten auf demselben Modell. Änderungen in einem Editor triggern Events, alle Editoren abonnieren Events und aktualisieren sich automatisch.

7.6.2 Akzeptanzkriterien

Core Modell

A5-1 Modell ist unabhängig von UI-Komponenten (MVC (Model–View–Controller) /MVVM (Model–View–ViewModel)).

A5-2 Modell repräsentiert vollständig das INTERLIS-Modell (AST (Abstract Syntax Tree) oder eigene Datenstruktur).

A5-3 Core Modell muss als INTERLIS Modell definiert sein.

Event-System

A5-4 Änderungen am Modell triggern typisierte Events (z. B. `ModelElementAdded`, `AttributeChanged`).

A5-5 Observer-Pattern oder Event-Bus.

A5-6 Events enthalten Origin-Information zur Vermeidung von Zyklen.

Transaktionale Änderungen

A5-7 Änderungen können gebündelt werden (Batch-Update).

A5-8 Rollback bei Fehler (z. B. ungültige Modell-Operation).

Synchronisation

A5-9 Alle Editoren aktualisieren sich in <1 Sekunde nach Modelländerung.

A5-10 Keine Race-Conditions bei parallelen Änderungen.

Undo/Redo

A5-11 Änderungshistorie auf Modellebene (nicht pro Editor).

A5-12 Undo/Redo ist konsistent über alle Ansichten hinweg.

7.7 Persistenz- und Serialisierung

Identifikation: A6

Bezug: F1 (Projektdatei), F2 (Synchronisation), F3 (Import/Export), F8 (Layout), F12 (Prosa)

7.7.1 Ziel und Beschreibung

Alle projektbezogenen Daten werden in einer einzigen .imcp-Projektdatei gespeichert. Die Projektdatei bündelt alle Artefakte: INTERLIS-Modelle, Diagramm-Layouts, Instanzen, Prosabeschreibungen und Projekteigenschaften.

Die Projektdatei ist portabel und kann ohne Zusatzdateien auf anderen Systemen geöffnet werden. Einzelne Bestandteile können über Import/Export-Funktionen (F3) als separate Dateien (.ili, .xtf/ xml) ausgetauscht werden.

7.7.2 Akzeptanzkriterien

Projektdatei (.imcp)

A6-1 Projektdatei bündelt alle Artefakte in einer portablen Datei.

A6-2 Konkretes Containerformat (z. B. ZIP-Archiv, strukturiertes XML) wird in der Konzeptphase festgelegt.

A6-3 Enthält: INTERLIS-Modelle, Diagramm-Layouts, Instanzen, Prosabeschreibungen, Projekteigenschaften (Editorstatus, Benutzereinstellungen).

A6-4 Interne Struktur der .imcp-Datei ist Git-kompatibel (textbasierte Formate, keine Binärdaten).

A6-5 Projektdatei ist ohne Zusatzdateien auf anderen Systemen öffenbar.

A6-6 In den Projektdaten werden ausschließlich die Quelltexte im Erfassungsformat gespeichert, keine abgeleiteten Formate.

A6-7 Die Prosatexte werden intern in modularen Einheiten („Schnipseln“) verwaltet, die in der .imcp-Datei als einzelne textuelle Dateien geführt werden.

A6-8 Diese textuellen Einheiten müssen so vorliegen, dass sie bei Verwendung von Git als separate Dateien sichtbar, versionierbar und ausserhalb des IMC editierbar sind.

Auto-Save

A6-9 Automatisches Speichern in konfigurierbarem Intervall (Standard: 30s).

A6-10 Automatisches Speichern kann durch die Benutzer:in bei Bedarf ein-/ausgeschaltet werden.

A6-11 Recovery bei IMC-Absturz (Wiederherstellung aus Auto-Save).

Laden und Parsen

A6-12 Fehlerhafte INTERLIS-Modelle können geöffnet werden (Best-Effort-Parsing).

A6-13 Zeichensatz-Erkennung (UTF-8, ISO-8859-1) mit Warnung bei Kodierungsproblemen.

Versionierung

A6-14 Interne Struktur der .imcp-Datei ist Git-kompatibel (text-basierte Formate, keine Binärdaten).

7.8 Texteditor-Komponente mit INTERLIS-Unterstützung

Identifikation: A7

Bezug: F5 (Text-Editor mit Syntaxprüfung)

7.8.1 Ziel und Beschreibung

Der Texteditor nutzt den Monaco Editor als Basis und erweitert diesen um INTERLIS-spezifische Funktionen über den Language Server (A3). Die Editor-Funktionalitäten sind in F5 beschrieben.

7.8.2 Akzeptanzkriterien

Editor-Framework

A7-1 Monaco wird genutzt.

A7-2 Keine Eigenentwicklung eines Code-Editors.

LSP-Integration

A7-3 Editor ist LSP-Client und kommuniziert mit INTERLIS Language Server (A3).

A7-4 Syntax-Highlighting, Completion, Diagnostics über LSP.

Performance

A7-5 Flüssiges Scrollen bei Dateien mit >10.000 Zeilen.

A7-6 Syntax-Highlighting in <100ms beim Öffnen.

7.9 Strukturbaum- und Navigationskomponente

Identifikation: A8

Bezug: F7 (Strukturbaum-Editor)

7.9.1 Ziel und Beschreibung

Der Strukturbaum zeigt die hierarchische INTERLIS-Modellstruktur (MODEL → TOPIC → CLASS → ATTRIBUTE) als interaktive Tree-View. Die Funktionalitäten sind in F7 beschrieben.

7.9.2 Akzeptanzkriterien

Tree-View-Framework

A8-1 Nutzung der nativen Tree-Komponente von Eclipse Theia.

A8-2 Keine Eigenentwicklung einer Baumkomponente.

Synchronisation

A8-3 Änderungen in anderen Editoren aktualisieren Baum in <1s.

7.10 Grafische Visualisierungsschicht (Diagramm-Editor)

Identifikation: A9

Bezug: F8 (Diagramm-Editor)

7.10.1 Ziel und Beschreibung

Der Diagramm-Editor stellt INTERLIS-Modelle als UML-ähnliche Klassendiagramme dar. Die Funktionalitäten sind in F8 beschrieben.

7.10.2 Akzeptanzkriterien

Rendering-Engine

A9-1 Eine etablierte Diagramm-Bibliothek wird genutzt.

A9-2 Wahl ist dokumentiert und begründet (Lizenz, Performance, Community).

Performance

A9-3 Flüssiges Rendering bei >50 Klassen.

A9-4 Auto-Layout in <2 Sekunden.

7.11 Tabelleneditor und Grid-Framework

Identifikation: A10

Bezug: F9 (Objektkatalog-Editor), F11 (Instanz-Editor)

7.11.1 Ziel und Beschreibung

Der Tabelleneditor bietet eine tabellarische Ansicht für Objektkataloge und externe Kataloge. Die Funktionalitäten sind in F9 und F11 beschrieben.

7.11.2 Akzeptanzkriterien

Grid-Komponente

A10-1 Etabliertes Grid-Framework wird genutzt.

A10-2 Wahl ist dokumentiert und begründet (Lizenz, Performance, Community).

Performance

A10-3 Flüssige Darstellung bei >1.000 Zeilen.

A10-4 Filterung/Sortierung in <500ms.

7.12 Dokumentations- und Prosa-Management

Identifikation: A11

Bezug: F12 (Fachliche Prosabeschreibungen)

7.12.1 Ziel und Beschreibung

Prosabeschreibungen werden über einen WYSIWYG-Editor bearbeitet und sind Bestandteil des Core Models. Die Funktionalitäten sind in F12 beschrieben.

7.12.2 Akzeptanzkriterien

Markup-Sprache

A11-1 AsciiDoc und/oder Markdown wird unterstützt.

A11-2 Entscheidung dokumentiert.

Persistenz

A11-3 Prosabeschreibungen sind Teil des Core Models und werden in der .imcp-Projektdatei gespeichert (vgl. A6).

A11-4 Export als separate .adoc/.md-Dateien ist über F3 möglich.

7.13 Git-Integration und Versionskontrolle

Identifikation: A12

Bezug: F13 (Versionsmanagement und Kollaboration)

7.13.1 Ziel und Beschreibung

Der IMC nutzt bestehende Git-Plugins ohne Eigenentwicklung. Git versioniert die .imcp-Projektdatei sowie ggf. exportierte Artefakte. Die Funktionalitäten sind in F13 beschrieben.

7.13.2 Akzeptanzkriterien

Plugin-Nutzung

A12-1 Bestehendes Git-Plugin der Plattform wird genutzt.

A12-2 Keine Eigenentwicklung für Git-Operationen.

Versionierung

A12-3 Die .imcp-Projektdatei wird versioniert.

A12-4 Interne Struktur der .imcp-Datei ist text-basiert für sinnvolle Diffs (vgl. A6).

7.14 Generierungs- und Export-Pipeline

Identifikation: A13

Bezug: F14 (Dokumentationsgenerierung)

7.14.1 Ziel und Beschreibung

Die Generierungs- und Export-Pipeline erstellt aus dem Core Model Dokumentationen in verschiedenen Formaten. Die Funktionalitäten sind in F14 beschrieben.

7.14.2 Akzeptanzkriterien

Formate

A13-1 Export nach HTML, PDF, Markdown/AsciiDoc.

Template-System

A13-2 Für die Dokumentgenerierung wird in Bezug auf die Darstellung und Kapitelstruktur/ Nummerierung ein Template genutzt.

A13-3 Das Template muss so bereitgestellt werden, dass es ohne Anpassungen am Programmcode des IMC modifiziert werden kann.

Performance

A13-4 Generierung eines mittleren Modells (50 Klassen, 10 Grafiken) in <1 Minute.

7.15 Such- und Indexiersystem

Identifikation: A14

Bezug: F15 (Suche, Filterung und Navigation)

7.15.1 Ziel und Beschreibung

Der IMC nutzt die native Suchfunktion von Eclipse Theia. Die Suche operiert auf dem Core Model. Die Funktionalitäten sind in F15 beschrieben.

7.15.2 Akzeptanzkriterien

Suchfunktion

A14-1 Native Suche von Eclipse Theia wird genutzt.

A14-2 Volltextsuche über alle INTERLIS-Modelle und Prosabeschreibungen im Projekt.

Performance

A14-3 Suche liefert Ergebnisse in <1 Sekunde bei 10.000 Elementen.

A14-4 Inkrementelle Indexierung bei Dateiänderungen.

7.16 Internationalisierung und Lokalisierung (i18n/l10n)

Identifikation: A15

Bezug: R4 (Mehrsprachigkeit)

7.16.1 Ziel und Beschreibung

Der IMC ist mehrsprachig über Resource-Bundles (JSON/Properties-Dateien). Standardsprache: Englisch. Erweiterbar für: Deutsch, Französisch, Italienisch.

7.16.2 Akzeptanzkriterien

Lokalisierung

A15-1 Alle UI-Texte sind in Resource-Bundles ausgelagert (keine Hardcoded-Strings).

A15-2 Standardsprache: Englisch.

A15-3 Mindestens eine weitere Sprache ist vorbereitet (z. B. Deutsch).

Framework

A15-4 i18next oder natives i18n-System der Plattform.

A15-5 Sprachumschaltung zur Laufzeit (ohne Neustart).

Erweiterbarkeit

A15-6 Neue Sprachen können durch Hinzufügen von JSON/Properties-Dateien ergänzt werden.

A15-7 Es ist ein Standard-Format zu verwenden, welches z. B. eine später Integration in eine Verwaltungslösung wie Transifex erlaubt.

7.17 Fehlerbehandlung, Logging und Debugging

Identifikation: A16

Bezug: R5 (Wartbarkeit)

7.17.1 Ziel und Beschreibung

Der IMC implementiert strukturiertes Logging für Benutzeraktionen, Fehler und Performance-Metriken.

7.17.2 Akzeptanzkriterien

Logging

A16-1 Strukturiertes Logging (JSON-Format).

A16-2 Log-Level: DEBUG, INFO, WARN, ERROR.

A16-3 Log-Ausgabe: Konsole (Development), Datei (Production).

Error-Handling

A16-4 Unbehandelte Exceptions werden geloggt und Benutzer erhält verständliche Fehlermeldung.

A16-5 Stack-Traces nur im Debug-Modus sichtbar.

Debugging

A16-6 Entwickler können LSP-Server debuggen (z. B. via Theia Debugger).

A16-7 Logging kann zur Laufzeit aktiviert/deaktiviert werden.

7.18 Performance und Skalierung

Identifikation: A17

Bezug: R5 (Erweiterbarkeit), F1–F15 (alle funktionalen Anforderungen)

7.18.1 Ziel und Beschreibung

Der IMC ist optimiert für grosse Modelle (>10.000 Zeilen, >50 Klassen) und nutzt Lazy Loading, inkrementelle Updates, asynchrone Operationen und Memory-Management.

Architektonische Prinzipien:

- **Lazy Loading**: Modell-Teile werden on-demand geladen (nicht alles beim Start)
- **Inkrementelle Updates**: Nur geänderte AST-Nodes werden neu geparst
- **Asynchrone Operationen**: Lange Tasks in Web Workers (UI bleibt responsiv)
- **Virtualisierung**: Tree/Grid rendern nur sichtbare Elemente (Viewport)

7.18.2 Akzeptanzkriterien

Performance-Budgets

A17-1 Syntaxprüfung: <1s bei Dateien bis 10.000 Zeilen

A17-2 Diagramm-Rendering: <2s bei 50 Klassen

A17-3 Tabelleneditor: <500ms bei 1.000 Zeilen

A17-4 Suche: <1s bei 10.000 Elementen

A17-5 Dokumentationsgenerierung: <2min bei mittlerem Modell

Lazy Loading

A17-6 Diagramme werden nur bei Bedarf gerendert (nicht bei IDE-Start).

Asynchrone Operationen

A17-7 Syntaxprüfung, Generierung, Suche laufen asynchron (Web Workers, Threads).

A17-8 UI bleibt bedienbar während laufender Operationen.

Memory

A17-9 IMC nutzt <500 MB RAM bei mittlerem Modell.

A17-10 Grosse Modelle (>100 Klassen) triggern Garbage Collection.

8 Anforderungen für weiteren Ausbau

Nachfolgend werden weitere Anforderungen für einen späteren möglichen Ausbau des IMC aufgelistet. Die Reihenfolge stellt keine Priorisierung dar.

Nr.	Beschreibung
W1	Converter um bestehende UML-Modelle, die im UML/INTERLIS-Editor erstellt wurden, in ein Format umzuwandeln, mit dem ein Import in den IMC möglich ist.
W2	Metadaten zu einem Modell sollen beschrieben werden können, wie auf geocat.ch. Im IMC sollten diese Daten erfasst und auch exportiert werden können.
W3	KI-gestützter Constraint-Editor (Wizard), der Constraints aus Prosabeschreibungen erstellt und validiert
W4	Constraint-Editor ohne KI mit if/then/else-Logik und Pop-Up zur Prüfung
W5	Vorschlag für automatisierte Modellentwürfe basierend auf Prosabeschreibung (KI)
W6	Versionierung, Änderungsverfolgung und Visualisierung von Modelländerungen
W7	Kommentarfunktion zur kollaborativen Kommunikation
W8	Automatisierte, KI-basierte Übersetzung als Vorschlag, manuelle Nachbearbeitungen möglich
W9	Durchsuchbarkeit und Prüfmechanismen (z.B. OID)
W10	Prüfung „on the fly“ oder „on demand“, mit Korrekturvorschlägen analog zu Rechtschreibprüfung
W11	Prüfung gegen Empfehlungen aus bisherigen Leitfäden und Standards (mit Auswahlmöglichkeit)
W12	Konsistente Vorschläge zwischen Funktionen, Prüfungen und Hilfefunktion erforderlich
W13	Sichtbarkeit des Deltas zu früheren Modellversionen
W14	Integration eines Darstellungs- bzw. Symbol-Editors in der Workbench
W15	Regeln zur Stilisierung, grafische Definition und Export in Formate wie SLD, QML, SLD-Codeschnipsel
W16	Erzeugung einer Symbollegende (Symbole, RGB-Werte, Objektbezug etc.)
W17	Gemeinsame Verwaltung von Datenmodell und Darstellungsmodell mittels Darstellungs- und Symbol-Editor