# WorkKonzeption eines Objektkonfigurators zur Erstellung von Auszügen einer Objektbibliothek.

Maximilian Müller, B.Sc.
Technische Hochschule Brandenburg
muellmax@th-brandenburg.de

Matthias Dobkowicz, Dipl.-Inf. (FH)
Technische Hochschule Brandenburg
matthias.dobkowicz@th-brandenburg.de

Prof. Dr. Andreas Johannsen
Technische Hochschule Brandenburg
johannse@th-brandenburg.de

Allan Fodi, B.Sc.
Technische Hochschule Brandenburg
fodi@th-brandenburg.de

# 1. Zusammenfassung

Das vorliegende Paper beschreibt ein Konzept für einen Objektkonfigurator zur Erstellung von Auszügen einer Objektbibliothek, welche als Webanwendung realisiert werden soll. Dabei wurden die Funktionsweisen und Besonderheiten gängiger Web-API Beschreibungssprachen untersucht und entsprechend auf eine Eignung für die Definition der Schnittstellen, welche in den Konfigurator importiert werden, ausgewertet. Das Konzept des Objektkonfigurators wird in seinen wichtigen Bestandteilen ausführlich erläutert. Abschließend wird ein Ausblick auf weiterführende Entwicklungen und Nutzungsmöglichkeiten der Anwendung gegeben.

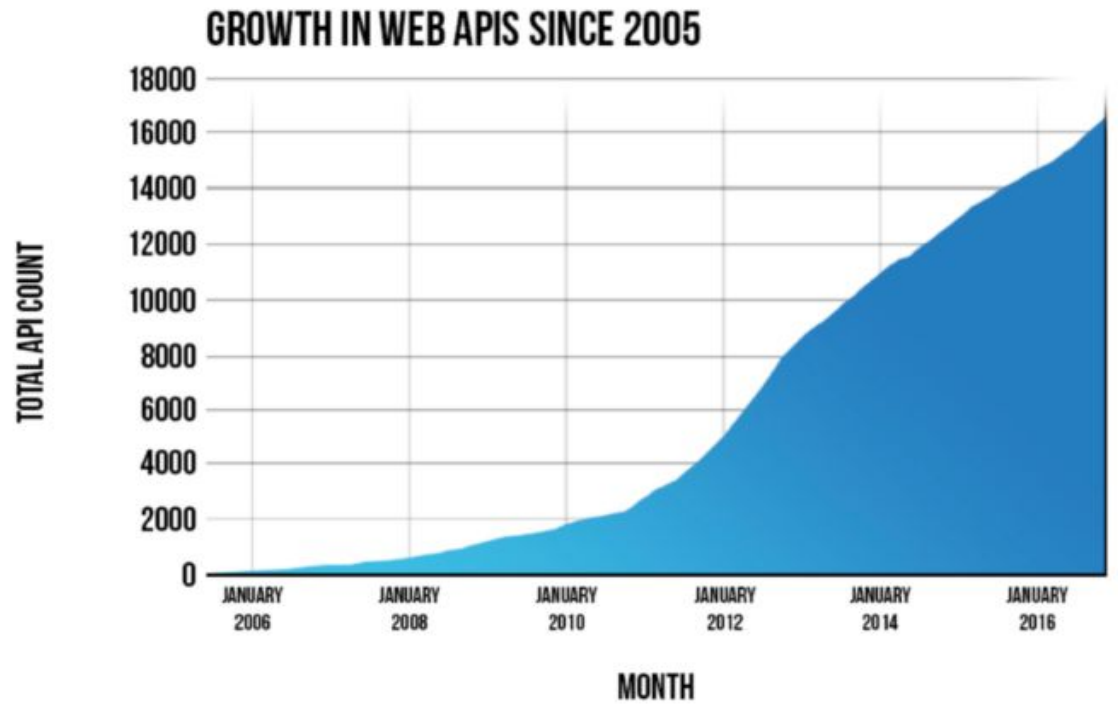
## 1.1 Einleitung

Moderne Softwareunternehmen haben die Wichtigkeit von Softwareschnittstellen in ihren Produkten bereits erkannt und dementsprechend reagiert (vgl. Abb. 1). Abhängig vom Use Case Szenario kommen in den Firmen verschiedene APIs und damit auch unterschiedliche Datenaustauschformate [Sind14] zum Einsatz. Zur besseren Wiederverwendbarkeit sind diese verwendeten Formate meist standardisiert. Einige Vertreter für elektronische Rechnungen sind z.B.:

**XRechnung** - ist ein Standard für elektronische Rechnungen auf Basis von XML. Entwickelt wurde der Standard speziell für die öffentliche Verwaltung nach den Vorgaben der CEN (Europäisches Komitee für Normung) (vgl. [IT Planungsrat 2017]).

**openTRANS** [opTR20] - ist ein offener XML-Standard für den Austausch von elektronischen Geschäftsdokumenten. Entwickelt wurde der Standard von deutschen und internationalen Unternehmen unter der Leitung von Fraunhofer IAO.

**KIW-Schnittstellenkatalog** [KIWSk20] - beruht auf offenen Standards (zur fachlichen Modellierung insb. die UN/CEFACT CCL [Geo93]) und gängigen Formaten. Unterstützt lose gekoppelte und flexibel erweiterbare Integrationsszenarien, in denen mehrere Business Software Anwendungen jeweils unabhängig voneinander Daten austauschen können, über generisch definierte KIW-Objekte und KIW-Methoden.



Quelle: https://blog.restcase.com/restful-apis-technologies-overview/

Abbildung 1: Wachstum von Web-APIs

Bei allen diesen Vertretern handelt es sich um lizenzkostenfreie Standards (nicht proprietär). Die Dokumentationen und Beschreibungen zu diesen Standards können ohne Registrierung im Internet kostenlos heruntergeladen werden.

## 1.2 Problembeschreibung

Im Kompetenzzentrums IT-Wirtschaft (KIW) besteht die Kernaufgabe in der Vernetzung von mittelständischen IT-Unternehmen und deren IT-Lösungen. Das KIW unterstützt dabei mit Informationen zum Thema Schnittstellen, dem KIW Schnittstellenkatalog, einem Techradar, Informationen zu rechtlichen Fragen beim Thema Kooperationen und einigem mehr[1].

Bei dem KIW Schnittstellenkatalog handelt es sich um eine API mit großer verfügbaren Objektbibliothek. Sie soll mittelständischen IT-Softwareentwicklern dabei helfen ihre eigenen IT-Lösungen einfacher mit Lösungen anderer Anbietern zu vernetzen, indem sich an einem bereits definierten Standard orientiert werden kann. In diesem Kontext wird eine Kooperation zwischen mehreren IT-Lösungen

[1] Weitere Angebote auf https://itwirtschaft.de/angebote/

verschiedener Anbieter immer als Konsortium bezeichnet. Das KIW versucht das “matchen” von Konsortien, also die Bildung von Konsortien selbst sowohl aktiv zu unterstützen, als auch Möglichkeiten zu schaffen das sich IT-Softwareanbieter eigenständig finden können und so zunehmend mehr Konsortien gebildet werden. Die Anforderungen der einzelnen Konsortien an die benötigten Austauschobjekte können meist mit einem Bruchteil des KIW-Schnittstellenkatalogs abgebildet werden, weshalb in der Regel nur Teile der Gesamtobjektbibliothek implementiert werden. Da das tatsächliche implementierte Subset des KIW-Schnittstellenkatalogs nur einen Auszug der Gesamtbibliothek darstellt muss auch die hervorgehende Dokumentation des Konsortiums nur den tatsächlich verwendeten Umfang umfassen. Das Problem besteht in der arbeitsaufwendigen Zusammenstellung der immer neuen Dokumentationen verschiedener Auszüge des Gesamtkatalogs für die einzelnen Konsortien, da je nach verwendeten Objekten und deren Attribute mitunter sehr komplexe Abhängigkeiten zu anderen Objekten entstehen, deren Zusammenstellung sehr zeitaufwendig sein kann.

## 1.3 Lösungsansatz

Um den Aufwand bei der häufigen Erstellung von Dokumentationen einzelner Auszüge einer Gesamtdokumentation zu verringern, stellt die Erstellung einer Anwendung, die es ermöglicht ein Subset unseres Gesamtkatalogs auszuwählen und automatisiert einen Auszug mit allen Abhängigkeiten exportierbar zu machen einen möglichen Lösungsansatz dar.

Die Software soll es ermöglichen eine Objektbibliothek zu importieren, benötigte Objekte und Attribute auszuwählen, und mit allen resultierenden Abhängigkeiten exportierbar machen.

## 1.4 Ideen zur Umsetzung

Zur Umsetzung kommen zwei grundlegende Varianten in Frage. Zum einen die Umsetzung als Desktopapplikation und zum anderen die Umsetzung als Webapplikation. Während die simple Umsetzung als Desktopapplikation den Lösungsansatz hinreichend umsetzt, wird die Umsetzung als Webapplikation [Jazayeri 2007] favorisiert, da das Tool so auch Außenstehenden leicht zugänglich gemacht werden kann. Die Idee besteht in einer öffentlich zugänglichen Plattform, auf der ohne weitere Anleitung Softwarekonsortien Subsets des KIW-Schnittstellenkatalogs an die eigenen Anforderungen anpassen können. Die Umsetzung als Webapplikation ist daher naheliegend, da so keine Installation

notwendig ist und weniger Einstiegshürden bestehen und das Tool leichter verbreitet werden kann.

Da keine bereits verfügbaren Tools mit vergleichbaren Funktionsumfang gefunden werden können, wird das Tool visionell auch für andere Objektbibliotheken konzipiert. Das Tool soll grundsätzlich also auch mit anderen Objektbibliotheken, wie bspw. XRechnung oder openTRANS nutzbar sein.

Um eine Schnittstelle mit Objektbibliothek importierbar zu machen ist eine entsprechende Definition in maschienenlesbarer Form nötig. Dafür eignen sich sogenannte Beschreibungssprachen. Es gilt zu untersuchen welche Beschreibungssprachen existieren und relevant und schlussendlich von der umzusetzenden Applikation unterstützt werden sollen.

Auf der Webseite unserer Applikation kann die API- und Objektdefinition die in einer unterstützten Beschreibungssprache vorliegt hochgeladen/importiert werden. Alle Objektdefinitionen werden durch den import in generalisierter Form in der Datenbank der Webapplikation gespeichert. Im Anschluss stellt die Webapplikation eine Übersicht aller Objekte zur Verfügung, wo nun eine Auswahl der benötigten Objekte getroffen werden kann. Für den öffentlichen Ansatz ist die aktuelle Version des KIW-Schnittstellenkatalogs bereits importiert, wodurch in dieser Situation direkt mit der Auswahl begonnen werden kann. Sind alle benötigten Objekte ausgewählt, können diese über einen Button inklusive aller Abhängigkeiten in einer unterstützten Beschreibungssprache exportiert werden. Der Endnutzer kann das Tool dann wie in Abbildung 2 beschrieben verwenden.


Abbildung 2: Grundlegender Nutzungsprozess

# 2. Technische Betrachtung der Komponenten

In diesem Kapitel werden einige Aspekte und Komponenten der Umsetzung, basierend auf dem aktuellen Stand, näher betrachtet.

## 2.1 Web-API Beschreibungssprachen

API (Application Programming Interface) Beschreibungssprachen sind formale Sprachen, die eine strukturierte Beschreibung einer API ermöglichen. Häufig handelt es sich bei diesen Beschreibungssprachen um Markup-Sprachen, wodurch sie sowohl leicht von Menschen, aber auch gut von Maschinen auszuwerten sind. Ist eine Schnittstelle mithilfe einer API Beschreibungssprache beschrieben, kann diese Definition verwendet werden, um mit Hilfe der maschienenlesbaren Form unter anderem Dokumentationen, Objektbibliotheken, oder Dummyendpunkte automatisiert zu generieren. Je nach gewählter Beschreibungssprachen stehen verschiedene Tools bereits zur Verfügung um die ausgearbeitete Definition für verschiedene Zwecke zu nutzen. Eine Beschreibungssprache als Importformat für die umzusetzende Webapplikation zu verwenden bietet sich daher an, da sie zum einen alle inhaltlichen Anforderungen der Objekte die wir benötigen abbilden [Danielsen 2013]. Hinzu kommt, dass viele Schnittstellen häufig bereits mit einer Beschreibungssprache abgebildet sind und so auf bereits vorliegenden Definitionen aufgesetzt werden kann. Auch im Kompetenzzentrum IT-Wirtschaft wird mit OpenAPI, einer Beschreibungssprache, die eigene Schnittstelle bereits abgebildet, um automatisiert Dokumentation basierend auf einer Datenbasis generieren zu können. Um die Relevanz einzelner Beschreibungssprachen, für die Unterstützung in der Webapplikation, besser einschätzen zu können und eine Vorauswahl treffen zu können, werden im Folgenden einige Beschreibungssprachen näher betrachtet.

### Swagger

Das Swagger Framework [SwFr20] bietet als gegenwärtig führendes Framework (basierend auf der Anzahl an Repositories auf GitHub) (vgl. [GitH20]) für API-Beschreibungssprachen. Swagger und Open API Formate sind austauschbar, da sie die gleiche Spezifikation unterstützen. Die Swagger-Definitionsdatei kann in JSON oder YAML geschrieben werden. Es existieren viele Tools zur Unterstützung von Dokumentation und Implementierung von API‘s auf Basis der Open API Spezifikation.

Das Swagger-Modell definiert einige allgemeine Informationen auf der Root-Ebene: Die Swagger-Version, den Host, der den Hostnamen oder die IP-Adresse enthält, auf der die API bereitgestellt wird, und deren Basispfad, der relativ zum Host ist.

Darüber hinaus enthält ein obligatorisches Info-Objekt andere allgemeine Informationen wie Titel, Beschreibung, API-Version, Kontakt usw. Darüber hinaus definiert die globale Schema-Eigenschaft das Transportprotokoll (z.B. HTTPS). Die einzelnen Ressourcen, welche als *path* Objekte (Im Root-Objekt *paths*) repräsentiert werden, müssen mindestens eine Operation enthalten.

```yaml
openapi: 3.0.0
servers:
  - url: 'http://petstore.swagger.io/v2'


info:
  description: >-
    This is a sample server Petstore server. You can find out more about Swagger
    at [http://swagger.io](http://swagger.io) or on [irc.freenode.net,
    #swagger](http://swagger.io/irc/). For this sample, you can use the api key
    `special-key` to test the authorization filters.
  version: "1.0-oas3"
  title: Swagger Petstore
  termsOfService: 'http://swagger.io/terms/'
  contact:
    email: apiteam@swagger.io
  license:
    name: Apache 2.0
    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'
externalDocs:
  description: Find out more about Swagger
  url: 'http://swagger.io'
paths:
  /pet:
    post:
      tags:
        - pet
      summary: Add a new pet to the store
      operationId: addPet
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Pet'
```

Abbildung 3: Auszug einer beispielhaften Swagger API Dokumentation[2]

## RAML

RESTful API Modeling Language [RAML20] (RAML) ist eine YAML-basierte Sprache zur Beschreibung von RESTful APIs.Dahinter stehen Arbeitsgruppen von Firmen wie MuleSoft (Leiter), VMware, Cisco, etc., deren Ziel es ist ebenfalls wie bei Swagger einen offenen Standard im Bereich der API Beschreibungssprachen zu etablieren.

[2] Die gesamte Beispieldokumentation ist hier zu finden:
https://app.swaggerhub.com/apis/helen/petstore/1.0-oas3

Wie bei Swagger ist auch RAML durch viele unterstützende Tools vielseitig anwendbar. Im Gegensatz zu Swagger ist RAML ein reiner Design-First-Ansatz. Der Fokus von RAML liegt auf dem Entwerfen von sauberen und lesbaren Beschreibungen, weshalb sie YAML als Design-Markup Sprache gewählt haben. Die größte Stärke von RAML besteht darin, dass die Beschreibungen von nicht technisch versierten Stakeholdern verstanden werden kann. Tatsächlich unterscheiden sich die Grundlagen des RAML-Metamodells nicht wesentlich von denen des Swagger-Metamodells.

**Unterschiede zu SWAGGER**

Grundlegende Informationen (root) Version, Titel, Medientypen usw. können hier definiert werden. Im Gegensatz zu Swagger können externe Dateien wie Schemadateien, Ressourcentypen oder Merkmale referenziert werden. Dies bedeutet, dass normalerweise mehrere Dateien eine API beschreiben.

Methoden Es gibt keinen Unterschied in der Definition von RAML-Methoden und Swagger-Operationen. mit der Ausnahme, dass die Sicherheitsschemata in RAML mit dem Tag "secured by" und der dazugehörigen Anwendung referenziert werden.

## API Blueprint (Markdown)

API Blueprint ist eine leistungsstarke API-Designsprache auf hoher Ebene für Web-APIs. API Blueprint ist einfach und für alle am API-Lebenszyklus Beteiligten zugänglich. Die Syntax ist eine Kombination aus Markdown- und MSON-Syntax. Das Ziel des API Blueprint-Formats besteht darin, die Design-First-Philosophie für REST-APIs zu ermöglichen. Das Format funktioniert jedoch genauso gut für die Dokumentation vorhandener APIs.

**API Blueprint und Swagger Unterschiede**

Swagger wird in YAML definiert, ist einfach zu parsen und bietet eine breite Wahl an Erweiterungen an, welche passend in den Software-Lebenszyklus eingebunden werden können. API Blueprint hingegen benutzt eine Markdown Syntax (Dokumentation), MSON (Schemas) und bietet vollen Support für Open-Source-Tooling. API Blueprint ist vollständig Open Source unter der MIT-Lizenz, Swagger hingegen ebenfalls Open Source jedoch unter der Apache Lizenz verfügbar. Erweiternde Dienste wie Swagger Hub sind kostenpflichtig und sind im Vergleich zu API Blueprint daher eine kostenintensivere Variante. Ziel ist es, dass API Blueprint einfach und frei zugänglich bleibt. Der Fokus liegt bei der Semantik der Anwendungsdomäne, Datenmodellierung und Beziehungen von API‘s. Es wird vermieden, dass Benutzer zu einem bestimmten Entwurfsprinzip

(Architekturstil) gezwungen werden. Es soll die Entwicklung von REST- und HTTP-basierten API's stark vereinfachen. Swagger wird mit Tools geliefert, mit denen eine Beschreibung aus Code generiert werden kann. Die API Blueprint-Syntax erleichtert die Beschreibung von Hypermedia / REST-APIs. Die Vorteile von Swagger machen sich daher besonders bei den unterstützenden Tools auf Konstrukt- und Codeebene erkennbar, welche bei API Blueprint fehlen.

## OpenAPI

Die OpenAPI-Spezifikation ist eine Community-gesteuerte offene Spezifikation innerhalb der OpenAPI-Initiative, einem Projekt der Linux Foundation, welches als Konsortium eine Standardisierung von API- Beschreibungssprachen vorantreibt.

OpenAPI basiert auf der Swagger 2.0 Spezifikation. Die OpenAPI-Spezifikation (OAS) definiert eine standardmäßige, programmiersprachenunabhängige Schnittstellenbeschreibung für HTTP-APIs, mit der sowohl Menschen als auch Computer die Funktionsweise eines Dienstes erkennen und verstehen können, ohne Zugriff auf Quellcode, zusätzliche Dokumentation oder Überprüfung des Netzwerkverkehrs zu benötigen. Bei erfolgreicher korrekter Definition mit OpenAPI kann ein Verbraucher den Remote-Service mit einem Minimum an Implementierungslogik verstehen und mit ihm interagieren. Ähnlich wie bei den Schnittstellenbeschreibungen für die untergeordnete Programmierung beseitigt die OpenAPI-Spezifikation das Rätselraten beim Aufrufen eines Dienstes.

Zu den Anwendungsfällen für maschinenlesbare API-Definitionsdokumente gehören unter anderem: interaktive Dokumentation; Codegenerierung für Dokumentation, Clients und Server; und Automatisierung von Testfällen. OpenAPI-Dokumente beschreiben APIs-Services und werden entweder im YAML- oder im JSON-Format dargestellt. Diese Dokumente können entweder statisch erstellt und bereitgestellt oder dynamisch aus einer Anwendung generiert werden.

**Gemeinsamkeiten zu Swagger**

Swagger bestand seit jeher aus der Spezifikation und den wichtigsten Open-Source-Tools, vor allem der Swagger-Benutzeroberfläche, dem Swagger-Editor und Swagger Codegen. Ein großer Grund, warum die Spezifikation so weit verbreitet wurde, war das Werkzeug, welches die Implementierung oder Dokumentation erleichtert. Da Swagger auf Basis der OpenAPI Spezifikation implementiert wurde, sind beide Beschreibungssprachen ähnlich und besitzen somit die gleiche praktikable Struktur.

## Entscheidung

Es sollen in Zukunft mehrere Web-API Beschreibungssprachen im Objektkonfigurator interpretiert werden können, um eine breitere Unterstützung zu ermöglichen. Dabei wird ein besonderes Augenmerk auf funktional passende Web-API Beschreibungssprachen gesetzt. Die OpenAPI Spezifikation wird hierbei anfänglich unterstützt, da diese gegenwärtig mehrheitlich in Projekten vertreten ist und durch die einfache leicht verständliche Logik eine gut übersichtliche Lösung darbietet und außerdem die Nutzung der KIW Schnittstelle ermöglicht.

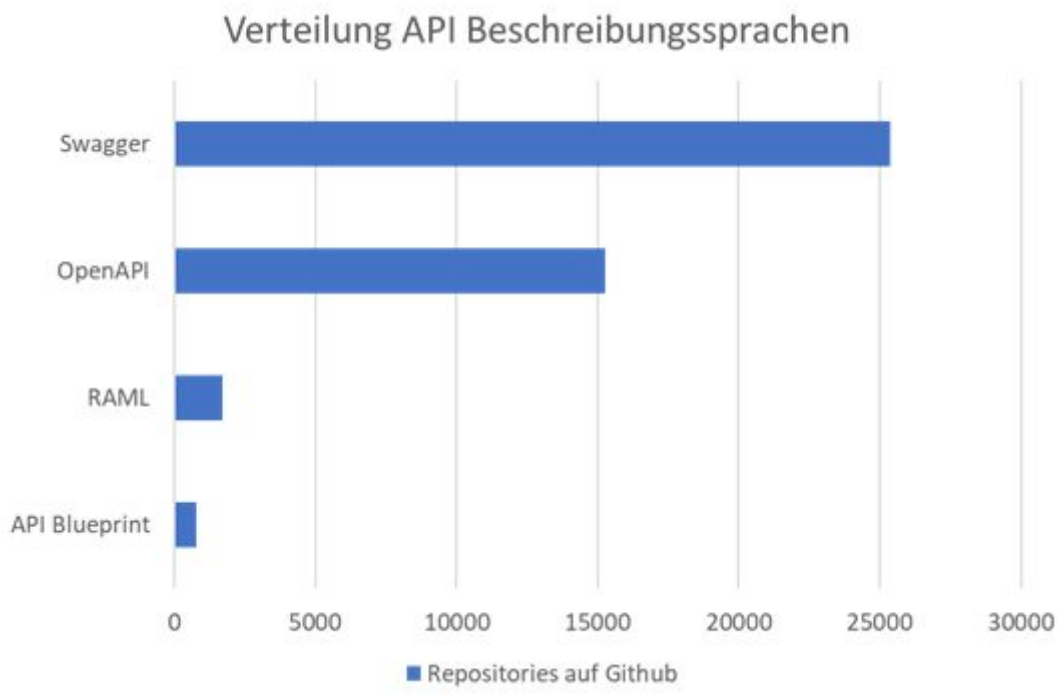



Abbildung 4: Verteilung der API Beschreibungssprachen auf Github (Stand: 17.09.2020)

## 2.2 Import von Objektbibliotheken in die Konfigurator-Datenbank

Im Folgenden wird der Ablauf beschrieben, wie Objektbibliotheken die als Beschreibungssprachen vorliegen in die Konfigurator Datenbank importiert werden. In der Entwicklungsphase ist vorgesehen, die Skriptsprache „Perl“ auf Konsolenebene einzusetzen [Wall 2001]. Im späteren Verlauf kann für den Import ein komfortables Frontend erstellt werden.

Im ersten Schritt wird die Objektbibliothek über die Skriptsprache eingelesen. Dabei wird eine Analyse gestartet, um die Beschreibungssprache zu ermitteln.

Nach erfolgreicher Erkennung der Sprache wird in Schritt zwei das benötigte Beschreibungssprachen Modul geladen.

Im dritten und letzten Schritt werden mit Hilfe des geladenen Moduls die Objekte, Attribute und deren Eigenschaften extrahiert und danach in die Konfigurator Datenbank geschrieben.

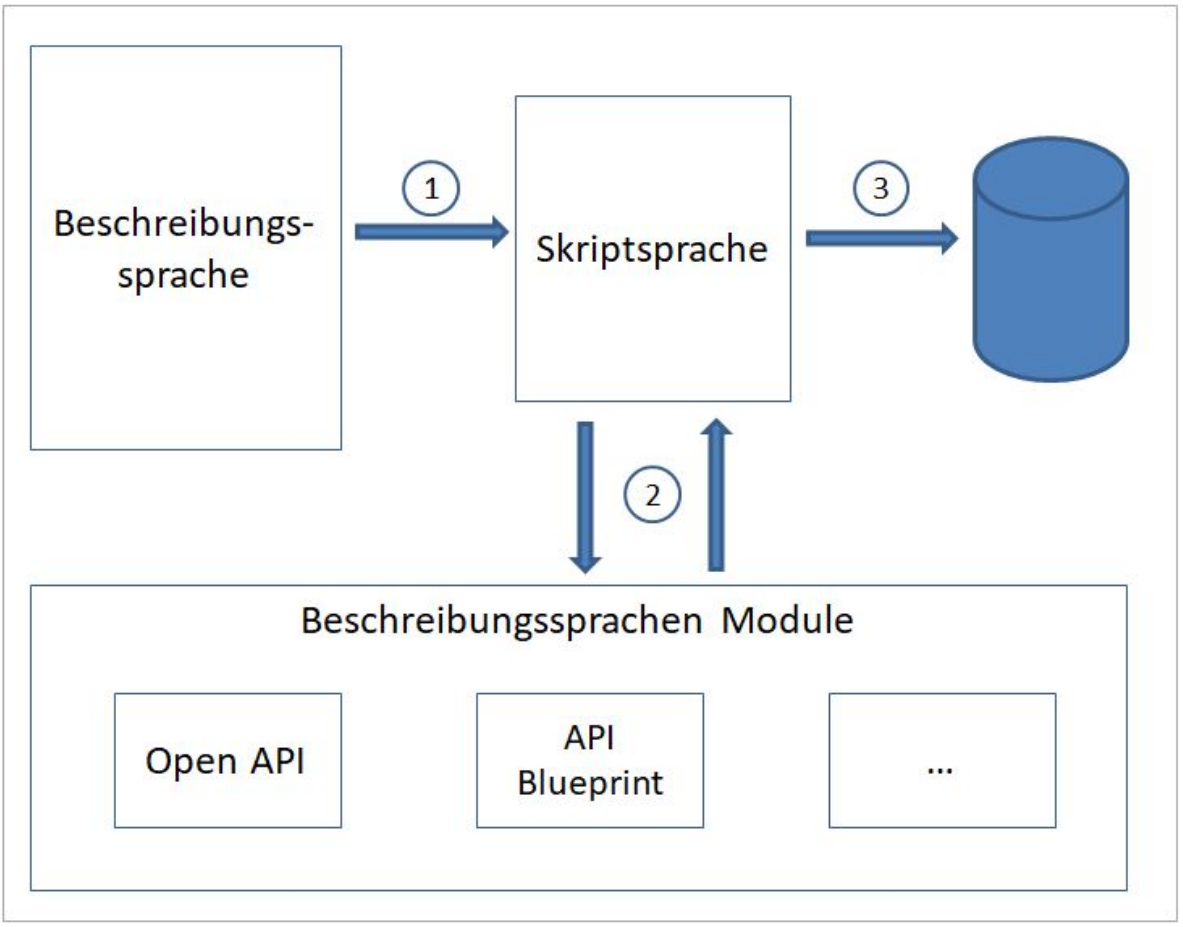



Abbildung 5: Skizze des Importprozesses

## 2.3 Die Webanwendung

Der Schnittstellen-Konfigurator soll über eine Online Plattform bereitgestellt werden, so dass dieser mit Hilfe eines Browsers schnell erreichbar und ohne weitere Installation verfügbar ist. Im Zuge dieses Konzeptes wurden bereits einige Tests durchgeführt. Hierbei kamen folgende Skriptsprachen und Plugins zum Einsatz.

- **PHP** [PHP20] - ist eine weit verbreitet Open Source Skriptsprache, welche größtenteils im Bereich Webanwendungen zum Einsatz kommt
- **JQuery** [JQY20] - ist eine der meist verwendeten freien JavaScript - Bibliothek und steht unter der MIT Lizenz
- **jsTree** [JsTr20] - ist ein kostenloses JQuery Plugin (MIT Lizenz), welches die Möglichkeit bereitstellt, Datenstrukturen als interaktive Bäume darzustellen.

## Darstellung der Objekte und Attribute über die Online Plattform

Um die Objekte und deren Attribute für den Anwender sichtbar und auswählbar zu machen, ergaben sich drei Hauptprobleme bei der Analyse:

1. Verschiedene Versionen eines Standards darstellen/auswählen
2. Darstellung der Objekte und Attribute
3. Suchbarkeit innerhalb der Objekte und Attribute

Eine ausführliche Recherche mit anschließenden Tests ergab, dass das kostenlose Plugin „jsTree“ [JsTr20] die oben beschriebenen drei Probleme löst.

In der Abbildung 6 werden die unterschiedlichen Versionen als „Tabs“ im oberen Bereich dargestellt. Direkt darunter befindet sich ein Suchfeld, was die Suche von Objekten und deren Attribute erlaubt und den Baum auch direkt an der Trefferstelle öffnet. Als letzte werden die Objekte und Attribute als Baumstruktur dargestellt. Gleichzeitig erlaubt das Plugin einzelne Objekte und deren Attribute zu markieren.


Abbildung 6: Darstellung von Objekten und Attributen als Baumstruktur

## Datenbankkonzept

Zur Umsetzung des Datenbankkonzeptes des Objektkonfigurators wird eine relationale Datenbank verwendet. Um das Datenbankschema für die

Konfigurator-Datenbank zu erstellen, wurden drei Tabellen modelliert. Abbildung 7 zeigt die daraus entstandenen Datenbanktabellen. Im Folgenden wird erklärt, welche Aufgaben die jeweiligen Tabellen haben.

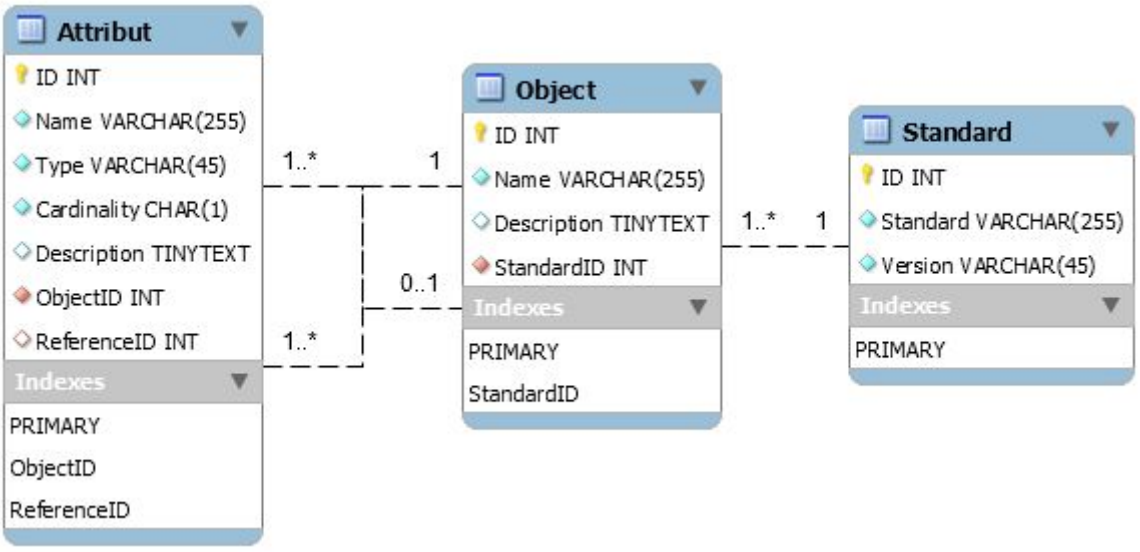



Abbildung 7: EER Diagramm der Tabellen der Konfigurator-Datenbank

## Tabelle “Standard”

Die Tabelle “Standard” stellt wichtige Informationen zur Objektbibliothek bereit.

| **Spaltenname** | **Funktion** |
|---|---|
| ID | Primarschlüssel der Tabelle “Standard |
| Standard | Name des jeweiligen Standards |
| Version | Version des Standards |

## Tabelle “Object”

Die Tabelle “Object” verwaltet alle Objekte und stellt über den Fremdschlüssel die Beziehung zur Tabelle “Standard” her.

| **Spaltenname** | **Funktion** |
|---|---|
| ID | Primärschlüssel der Tabelle “Object” |
| StandardID | Fremdschlüssel der Tabelle “Standard” |
| Name | Name des Objektes |
| Description | Beschreibung des Objektes |

## Tabelle “Attribut”

Die Tabelle “Attribut” stellt wichtige Informationen zum Attribut bereit und wird über den Fremdschlüssel “ObjectID” mit der Tabelle “Object” verknüpft.

| **Spaltenname** | **Funktion** |
|---|---|
| ID | Primärschlüssel der Tabelle “Attribut” |
| ObjectID | Fremdschlüssel der Tabelle “Object” |
| Name | Name des Attributes |
| Type | Datentyp des Attributes |
| Cardinality | Kardinalität mit folgenden Parametern: “O” für Optional, “M” für Mandatory, “C” für Conditional |
| Description | Beschreibung des Attributes |
| ReferenceID | Referenzierung eines anderen Objektes |

# 3. Ausblick

Nach einer ersten Betrachtung einer Softwarelösung zur Generierung von Objektbibliotheksauszügen, haben sich nach den ersten Recherchen keine auf den ersten Blick erkennbaren Hindernisse in Bezug auf die Umsetzbarkeit des Vorhabens aufgetan. Der erste Entwurf als Webapplikation scheint alle Anforderungen abdecken zu können. Der Import durch eine Beschreibungssprachen scheint praktikabel, und die Darstellung für den Benutzer als Objektbaum sollte die Ansprüche an die Übersichtlichkeit erfüllen.

Natürlich handelt es momentan noch um ein rudimentäres Konzept, dennoch scheint die grundsätzliche Umsetzbarkeit gegeben zu sein. Weiterführend sind weitere Betrachtungen von Teilkomponenten der Softwarelösung nötig. So müssen bspw. die Implementierung weiterer Beschreibungssprachen in Form von Modulen evaluiert werden, oder erste Recherchen zur Umsetzung der einzelnen Interpreter der verschiedenen Beschreibungssprachen für den Datenbankimport begonnen werden. Auch der Algorithmus zur Bestimmung und Zusammenstellung abhängiger Objekte bedarf einer genaueren Betrachtung.

Es gibt also noch einige, auch konzeptionell betrachtet, offenen Punkte, die ausblickend analysiert werden müssen, um mittelfristig einen ersten Prototypen zu realisieren und schlussendlich das fertige Tool zu veröffentlichen.

# Quellenverzeichnis

[Danielsen 2013] Danielsen, P.J. and Jeffrey, A., 2013, June. Validation and interactivity of web API documentation. In 2013 IEEE 20th International Conference on Web Services (pp. 523-530). IEEE.

[Geo93] Georg T., EDIFACT - Ein Implementierungskonzept für mittelständische Unternehmen (German Edition) (Wirtschaftsinformatik); Springer Fachmedien Wiesbaden GmbH; Wiesbaden 1993, ISBN 978-3-8244-2044-5

[GitH20] GitHub, Swagger Repository, URL: https://github.com/search?q=swagger, Abruf am 18.09.2020

[IT Planungsrat 2017] IT-Planungsrat; Standard XRechnung, Version 1.1, URL: https://www.xoev.de/sixcms/media.php/13/XRechnung%201.1%20-%2030.pdf; November 2017

[Jazayeri 2007] Jazayeri, M., 2007, May. Some trends in web application development. In Future of Software Engineering (FOSE'07) (pp. 199-213). IEEE.

[JQY20] JQuery: JavaScript-Bibliothek, URL: https://jquery.com/, Abruf am 18.09.2020

[JsTr20] jsTree: JQuery Plugin, Datenstrukturen als Bäume darstellen, URL: https://www.jstree.com/, Abruf am 18.09.2020

[KIWSk20] KIW Schnittstellenkatalog, Technische Dokumentation, URL: https://itwirtschaft.de/angebote/schnittstellen/schnittstellenkatalog/, Abruf am 18.09.2020

[opTR20] OpenTrans Dokumentation und Spezifikation, Version 2.1, URL: https://www.digital.iao.fraunhofer.de/de/publikationen/OpenTRANS21.html, Abruf am 18.09.2020

[PHP20] Serverseitige Skriptsprache: PHP, URL: https://www.php.net, Abruf am 18.09.2020

[RAML20] RESTful API Modeling Language, URL: https://raml.org/, Abruf am 18.09.2020

[Sind14] Sindermann Sebastian, Modellbasierte virtuelle Produktentwicklung, Springer-Verlag Berlin Heidelberg 2014, DOI 10.1007/978-3-662-43816-9_14

[SwFr20] Swagger Framework, API Dokumentation, URL: https://swagger.io/, Abruf am 18.09.2020

[Wall 2001] Wall L. Christiansen T., Orwant J; Programmieren mit Perl; O'Reilly Verlag; Köln 2001