Konzeptionierung und Entwicklung einer administrativen Applikation für die Editierung und Erweiterung von Softwareversprachlichung.

Professor Dr. Frank Herrmann
Innovationszentrum für Produktionslogistik und Fabrikplanung
Ostbayerische Technische Hochschule Regensburg
Galgenbergstraße 32
93053 Regensburg
Frank.Herrmann@OTH-Regensburg.de

Marcel Roth
Projekt29 GmbH & Co. KG
Ostengasse 14
93047 Regensburg
marcel.roth@projekt29.de

ABSTRACT
Dieser Artikel behandelt die Bearbeitung und Umsetzung eines Projekts mit dem obigen Titel. Dabei liegt das Augenmerk weniger auf der technischen Implementierung als viel mehr auf dem Umfang und dem Management des Projekts selbst.

Keywords
Softwareversprachlichung, Projektmanagement, Anforderungsmanagement, Softwaredesign, Dokumentation

1. KERNZIELE DES PROJEKTS
Die Datenschutzmanagementsoftware, kurz DSM-Software, der *Projekt29 GmbH & Co. KG*, muss aufgrund eines wachsenden Kundenstammes immer mehr Sprachen anbieten. Um das bestehende System um weitere Sprachen zu erweitern, gibt es bereits einen Weg, welcher allerdings für jede Änderung manuell von einem Administrator ausgeführt werden muss und nicht komplett Nutzergebunden ist. Außerdem ist es schwer einen Überblick über die bereits vorhandenen Sprachen und deren Umsetzung zu bekommen. Deshalb soll eine Anwendung (genauer ein Modul für den Adminbereich der DSM-Software) entwickelt werden, die die folgenden Kernziele umsetzt.

- einspielen von Sprachen durch CSV- oder JSON-Dateien
- erweitern von Sprachen durch CSV- oder JSON-Dateien
- darstellen einer detaillierten Übersicht über alle im System verfügbaren Sprachen
- editieren einzelner Sprachwerte der Software (z.B. kundenspezifische Bezeichnungen oder Berichtigung / Erneuerung einzelner Werte)
- exportieren von Sprachen oder Sprachwerten als CSV- oder JSON-Dateien

2. UMFANG DES PROJEKTS
Dieses Projekt dient der Entwicklung des im Vorangegangenen beschriebenen Moduls des Administrationsbereichs der DSM-Software. Dabei geht es um ein komplettes IT-Projekt. Dazu gehört die Wahl eines geeigneten Vorgehensmodells, die Erfassung genauer Anforderungen, die Planung der Schnittstellen, die Erstellung eines Designkonzepts, die Implementierung, sowie eine Dokumentation und Analyse des Endergebnisses. Zudem ist ein neuer Administrationsbereich geplant, der im Zuge dieses Projekts initial erstellt werden soll. Dieser wird dann mit seinem ersten Modul (für Versprachlichung) ausgestattet. Im Folgenden wird das Vorgehen für jeden dieser Tätigkeitsbereiche kurz beschrieben. Außerdem wird das Resultat jedes Abschnitts aufgezeigt und abschließend zu einem Endergebnis zusammengeführt.

3. AUSWAHL DES VORGEHENSMODELLS
Ein geeignetes Vorgehensmodell muss zur Projektstruktur passen, darf nicht unnötig kompliziert sein und soll einen klaren Projektablauf definieren. Um ein passendes Vorgehensmodell zu finden, müssen zunächst verschiedene Vorgehensmodelle angesehen und im Bezug auf das Projekt evaluiert werden. Dazu wurden in diesem Projekt die nachfolgend aufgelisteten Vorgehensmodelle genauer begutachtet.

- Basismodell
- Wasserfallmodell
- V-Modell
- Spiralmodell
- Extreme Programming
- Scrum

Vor allem die agilen Methoden (Extreme Programming und Scrum), sowie das Spiralmodell sind zu umfangreich für dies Projekt. Außerdem besteht keine große Notwendigkeit für regelmäßige Meetings, da es sich um ein Ein-Mann-Projekt handelt. Somit bleiben nur noch die drei erstgenannten Modelle. Das V-Modell ist ein gutes Vorgehensmodell, welches das Testen in den Vordergrund stellt. Allerdings würden automatisierte Tests den zeitlichen Rahmen für das Projekt sprengen, weswegen die Implementierung im Fokus steht. Letztendlich wurde das Basismodell verwendet und nach den Wünschen des Projektbetreuers etwas abgeändert.

Abbildung 1 zeigt das angewendete Vorgehensmodell, welches vier Phasen hat, die nacheinander durchgearbeitet werden müssen. Dabei ist die Reihenfolge entscheidend, da die Ergebnisse einer Phase als Grundlage für die nächste Phase dienen. Der Ablauf und die Ergebnisse der einzelnen Phasen werden in den nächsten Abschnitten beschrieben.

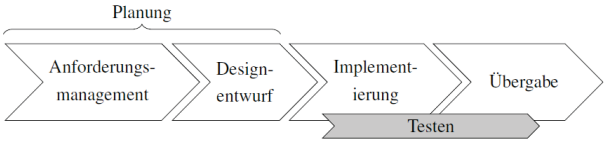

Abbildung 1: Vorgehensmodell dieses Projekts

4. ERFASSUNG DER ANFORDERUNGEN

Die erste Phase beschäftigt sich mit den Anforderungen an die Software (also das Endergebnis des Projekts). Hier geht es darum, das Ziel für das Projekt möglichst genau zu definieren. Dafür gab es einen iterativen Prozess (siehe Abbildung 2), bei dem ich als Anforderungsmanager fungierte. Dabei ging es darum möglichst detailliert die Funktionalitäten und Schnittstellen der Software festzulegen und regelmäßig mit dem Projektbetreuer abzustimmen und zu validieren. Um eine allgemein gültige Fassung zu schaffen und

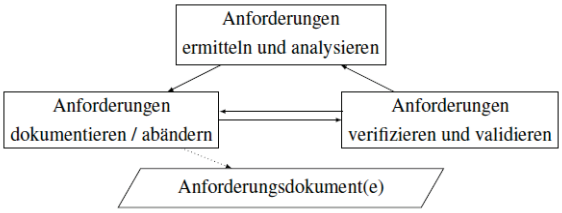

Abbildung 2: Zyklus des Anforderungsmanagements

spätere Streitigkeiten zu vermeiden ist es wichtig die Anforderungen korrekt zu dokumentieren. Auch dazu wurde Recherche betrieben. Aber anstatt eines speziellen Anforderungsmanagementtools wurde sich auf eine einfache Tabellendarstellung geeinigt, welche z.B. in Excel gepflegt werden kann. Der Aufbau dieses Anforderungsdokuments wird in Tabelle 1 beschrieben.

Attribut	**Beschreibung**
ID	Eindeutiger Wert zur Identifikation der Anforderung
Bezeichnung	Sprechender Name für die Anforderung
Beschreibung	Kurze aber exakte Beschreibung der Anforderung
Quelle	Ursprung der Anforderung (Personen oder Interessensgruppen)
Typ	Typ der Anforderung
Status	Umsetzungsfortschritt der Anforderung
Version	Versionierung der Anforderung
Letzte Änderung	Datum der letzten Änderung

Table 1: Vorlage für den Informationsumfang einer einzelnen Anforderung

5. ERSTELLUNG DES DESIGNKONZEPTS

Die zweite Phase dient dazu, die vorher festgelegten Anforderungen in ein Designkonzept für die Software umzuwandeln. Dabei geht es nicht darum das Frontend vollständig zu designen. Im Mittelpunkt steht ein Konzept für die Benutzeroberfläche der Anwendung. Dazu zählen das Farbschema, die Formatierung von bestimmten Elementen, sowie die Navigation. Beispielsweise geht es darum welche Elemente in den Kopfbereich kommen und ob Daten besser als Checkboxen oder Dropdown-Menüs dargestellt werden. Für die Umsetzung des Designkonzepts wurde der Webservice *balsamiq* verwendet. Dieser ermöglicht das Erstellen von UI-Scribbles. Außerdem können diese Zeichnungen gleich online gespeichert und anderen für einen Review bereit gestellt werden. Dieses Tool ist also perfekt geeignet für einen iterativen Ansatz, bei dem regelmäßig Rücksprache gehalten wird. Für dieses Projekt wurde wie in Abbildung 3 gezeigt vorgegangen. Die wichtigsten Ergebnisse aus dem Designkonzept

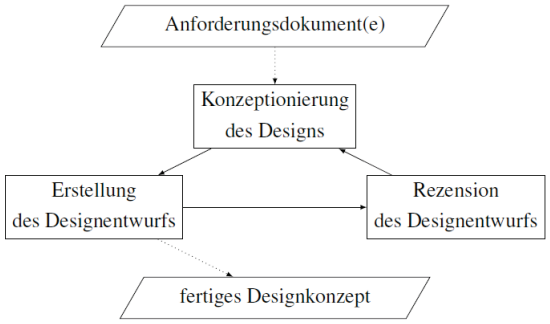

Abbildung 3: Zyklus des iterativen Designprozesses

waren die Aufteilung der Anwendung in Kernbereiche, sowie einzelne Komponenten (z.B. Warnungs-Popups und Navigationselemente). Die vier Kernbereiche, die nachfolgend aufgelistet sind, beinhalten zusammen alle nötigen Funktionalitäten und sollen visuell voneinander getrennt sein.

- Sprachen (Übersicht)
- Sprachwerte (Editor und Übersicht)
- alle Imports
- alle Exports

6. IMPLEMENTIERUNG DES SERVERS

Bevor der Server entwickelt werden kann muss zunächst die Datenbank designed und erstellt werden. Das entfällt bei diesem Projekt jedoch, da das Datenbankschema dafür bereits existiert und in die Datenbank der DSM-Software integriert ist. Abbildung 4 zeigt die Tabellen dieses sogenannten LValue-Schemas (steht für "*Language Values*"). Die Unternehmen, die bei *Projekt 29* Kunde sind, werden in der Datenbank als "*Tenants*" also Mandanten verwaltet. Es gibt somit zwei verschieden Sprachumgebungen. Zum einen die **GEN**-Umgebung, die die Standardsprachen der Software abbildet, die für alle Benutzer gleichermaßen zur Verfügung stehen. Zum anderen die **TID**-Umgebung, welche kundenspezifische Anpassungen oder Erweiterungen der Standardsprachen, sowie komplett andere Sprachen bereitstellen kann. Um die Software zu versprachlichen geht das LValue-Konzept wie folgt vor. Die **LValue**-Tabellen beinhalten die textuellen Werte für alle Sprachen und verbinden diese via IDs mit der zugehörigen Sprache (**Lang**-Tabellen) und dem repräsentativen Symbol (**Symbol**-Tabelle). In der DSM-Software an sich werden somit keine tatsächlichen Sprachwerte mehr ver-

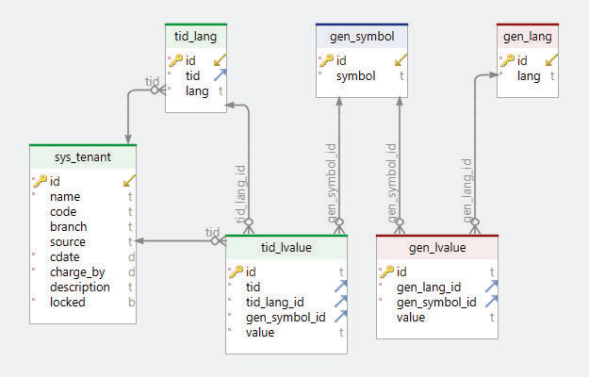

Abbildung 4: Datenbankschema des LValue-Konzepts

wendet sondern lediglich Symbole. Bevor dann ein Template oder eine Webseite an einen Nutzer ausgeliefert wird, werden die darin angegeben Symbole mit der passenden Sprache und Tenant-ID aufgelöst und durch den angeforderten Sprachwert substituiert.

Severstruktur und Frameworks

Der letzte Schritt bevor die Implementierung starten kann, ist das Aufsetzen des Projekts. Dabei ging es darum zu entscheiden welcher Aufbau und welche Frameworks verwendet werden sollen. Auf Details zur Entscheidungsfindung und den einzelnen Frameworks wird in der Zusammenfassung verzichtet. Zum programmieren wird serverseitig Java in der Version 17.0.5 benutzt. Als Build-Management-Tool wird ein *Gradle* der Version 7.4.1 angewendet. Die größte Änderung ist, dass für den Administrationsbereich nicht mehr *Spring* (wird in der DSM-Software genutzt), sondern *Quarkus* als Fullstack-Java-Framework eingesetzt wird. Ein weiteres wichtiges Framework, welches aus der DSM-Software übernommen wurde, ist *jOOQ*. Dieses Framework ist für die Generierung von Java-Klassen in Bezug auf die Datenbank. Es ermöglicht einen einfachen und übersichtlichen Zugriff vom Server auf die Datenbank unter Verwendung einer eigenen *Domain Specific Language* in Java.

Serverschnittstellen

Tabelle 2 zeigt eine Übersicht über alle, in den Anforderungen dokumentierten, Schnittstellen und ihre technische Spezifizierung. Hier kann auch die Trennung der vier Kernbereiche (siehe Punkt 5) erkannt werden. Allerdings bedienen die Schnittstellen für Symbole und LValues zusammen den Bereich für Sprachwerte.

Zuerst zu den Sprachfunktionalitäten. Der Endpunkt **/lang/cards** liefert eine Liste von Datenobjekten zur Darstellung der Übersicht einer Sprache. Mit **/lang/ls/copy** werden alle Sprachen aufgelistet, die für eine Kopieroperation zwischen den Sprachumgebungen bereitstehen. Im Anschluss kann mit **/lang/do/copy** eine gewünschte Sprache inklusive aller Sprachwerte auf eine andere Sprachumgebung kopiert werden. Der Endpunkt **/lang/tenants** gibt eine Liste von Mandanten zurück, welche danach gefiltert werden kann, ob dieser Mandant über eigene Sprachen verfügt oder nicht. Mit **/lang/bindings** bekommt man eine List von Mandanten, die über eine bestimmte Sprache verfügen (für Suchoperationen). Eine einzelne neue Sprache ohne Sprachwerte kann über **/lang/create** erstellt werden. Eine Sprache zu löschen (inklusive aller Sprachwerte) ermöglicht die Schnittstelle **/lang/delete**.

Weiter mit den Schnittstellen für Sprachwerte. Der Endpunkte **/sym/pagination** liefert seitenweise alle Symbole. Nachdem sehr viele Symbole in der Datenbank liegen wird die Ladedauer dadurch verringert, dass immer nur eine bestimmte Anzahl geladen wird. So werden nur bei Bedarf weitere Symbole geliefert ("*Lazy Loading*"). Durch **/sym/analyze** kann ein einzelnes Symbol analysiert werden. Als Rückgabe kommt eine Auflistung aller Orte an denen das analysierte Symbol verwendet wird. Nur ein Symbol das nirgends Anwendung findet kann mit **/sym/delete** entfernt werden. Der Endpunkt **/lval/pagination** funktioniert genau wie die *pagination* von Symbolen nur für textuelle Sprachwerte. Die Schnittstelle **/lval/create** ermöglicht die Erstellung eines einzelnen Sprachwertes für ein bereits existierendes Symbol. Sprachwerte können mit **/lval/update** auch einzeln bearbeitet und mit **/lval/delete** einzeln gelöscht werden.

Zuletzt geht es noch um den Import und Export. Über **/import/json** können JSON-Dateien ausgelesen und Sprachen sowie Symbole (inklusive von Sprachwerten) daraus importiert werden. Der Unterschied zum CSV-Import (**/import/csv**) ist, dass CSV-Dateien auf ein Symbol oder eine Sprache pro Datei limitiert sind. Exports werden auch für Sprachen und Symbole angeboten. Dafür gibt es die Endpunkte **/export/json** und **/export/csv**. Jedoch gilt für CSV-Dateien die gleiche Limitierung wie bei den Imports.

Zuordnung	HTTP	Pfad
Sprache	GET	/lang/cards
	GET	/lang/ls/copy
	GET	/lang/tenants
	GET	/lang/bindings
	POST	/lang/create
	POST	/lang/do/copy
	DELETE	/lang/delete
Symbole	GET	/sym/pagination
	GET	/sym/analyze
	DELETE	/sym/delete
LValues	GET	/lval/pagination
	POST	/lval/create
	PUT	/lval/update
	DELETE	/lval/delete
Imports	POST	/import/json
	POST	/import/csv
Exports	GET	/export/json
	GET	/export/csv

Table 2: Übersicht der Serverschnittstellen

7. IMPLEMENTIERUNG DES CLIENTS

Clientstruktur und Frameworks

Auch beim Client muss die Projektumgebung eingerichtet werden, bevor die Implementierung des Clients beginnen kann. Der Client wurde mit Typescript anstatt von Javascript geschrieben, um zumindest etwas besser an das vom Java-Server vorgegebene Typsystem anzuknüpfen. Zudem wurde für das Projekt die Node-Version 16.20.0 verwendet. Als Build-Management-Tool kommt *Vite* (Version 4.3.9) zum Einsatz. Als Client-Development-Framework wird *SvelteKit* eingesetzt. *SvelteKit* basiert auf Svelte und bietet eine einfache Komponentenbauweise, ein eigenes Routingkonzept, sowie weitere Extras (z.B. Icons und Animationen). Außerdem wird ein CSS-Framework names *Tailwind* verwendet, welches auch in allen anderen Frontends bei *Projekt 29* Anwendung findet. Es bietet einheitliche und standardisierte Klassen für die Formatierung von HTML-Elementen und entfert zuvor die Standardformatierung von allen Elementen.

Überblick des Clients

Wie bereits unter Punkt 5 beschrieben wurde der Client visuell in vier Teile mit unterschiedlichen Schwerpunkten aufgeteilt. Für die Erstellung des Frontends, welches nachfolgend gezeigt wird, wurden nur die in Tabelle 2 aufgelisteten Schnittstellen verwendet. In Abbildung 5 wird zunächst der Kopfbereich gezeigt, welcher in jedem Teil der Anwendung zu sehen ist. Er ist die Hauptnavigationskomponente und ermöglicht die Navigation durch die vier Kernbereiche der Software. Außerdem soll in der weiteren Entwicklung noch ein **Home**-Bereich eingeführt werden, auf dem Statistiken zu den Standardsprachen dargestellt werden. Nun folgt der

Abbildung 5: Kopfbereich des LValue-Moduls

Sprachteil des Moduls. Dieser besitzt eine eigene Navigationsleiste (siehe Abbildung 6 links) und folgende Funktionalitäten:

- Übersicht über die Standardsprachen
- Suche von Mandanten nach Tenant-ID
- Suche von Mandanten nach verfügbaren Sprachen

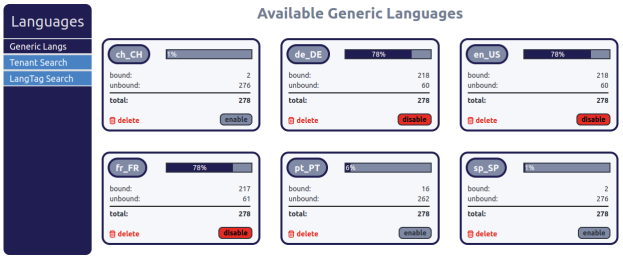

Abbildung 6: Übersicht über die Standardsprachen

Aus Gründen des Umfangs wird nicht jede Ansicht in dieser Zusammenfassung gezeigt. Das Herzstück des LValue-Moduls ist der sogenannte LValue-Editor, welcher in Abbildung 7 zu sehen ist. Er kann sowohl alle Symbole als auch alle LValues des Systems auflisten. Außerdem ist es möglich einzelne Symbole zu löschen. Das gleiche gilt auch für LValues, mit dem Unterschied, dass diese auch erstellt und geändert werden können. Da für jedes Symbol beliebig viele LValues existieren können, ist es auch möglich nur alle LValues für ein bestimmtes Symbol zu laden. Dabei kann oben links im Editor die gewünschte Zielgruppe (z.B. alle Symbole, alle LValues, nur Template-Symbole, usw.), ausgewählt werden. Daraufhin erscheinen je nach Zielgruppe in der linken Leiste die gewünschten Symbole oder in der Mitte alle LValues. Klickt man auf ein gelistetes Symbol erscheinen in mittleren Teil des Editors alle zugehörigen LValues. Die Leiste rechts ist für alle Operationen (erstellen, bearbeiten, löschen) auf ausgewählten Objekten zuständig. Das Info-Icon mittig oben kann benutzt werden, um ein ausgewähltes Symbol zu analysieren. Sollte dieses Symbol keine Anwendungsfälle haben, so kann es im erschienenen Popup optional gelöscht werden. Zum Schluss bleiben noch

Abbildung 7: LValue-Editor von *pso2admin*

das Import- und das Export-Segment übrig. Für beide gibt es einen Konfigurations-Wizard, welcher durchlaufen werden muss, bevor ein Import oder Export getätigt werden kann. Dabei geht es darum die folgenden Punkte festzulegen:

- Dateiformat: JSON / CSV
- Exporttyp: Sprache / Sprachwerte
- Umgebung: TID / GEN
- welche(r) Sprache(n) oder Sprachwert(e)

Erst wenn diese Konfigurationen eingestellt wurden ist es möglich eine Datei hochzuladen (Import) oder einen Export anzufordern. Allerdings hat der Import-Teil der Anwendung noch eine weitere Funktionalität. Dabei handelt es sich um das Kopieren von Sprachen. So können Sprachen von einer Umgebung auf eine andere kopiert werden. Zunächst muss dafür eine Zielumgebung (ein spezifischer Mandant oder die **GEN**-Umgebung) ausgewählt werden. Dann werden alle kopierbaren Sprachen aufgelistet. Aus diesen können dann Sprachen ausgewählt werden, die inklusive all ihrer Sprachwerte auf die Zielumgebung kopiert werden. Das ist beispielsweise nützlich wenn eine kundenspezifische Sprache neuerdings auch als Standardsprache bereitgestellt werden soll. Oder wenn ein Kunde bestimmte Sprachwerte ändern möchte, können diese aus der **GEN**-Umgebung auf diesen Mandanten kopiert und dort angepasst werden.

Endergebnis des Projekts

Das Kernergebniss in diesem Projekt ist die fertige erste Version des Sprachmoduls für den neuen Administrationsbereich (*pso2admin*) der DSM-Software. Also genau die Software, die im Vorangegangenen erklärt und beschrieben wurde. Insgesamt ist das Unternehmen mit der übergebenen Software sehr zufrieden. Sowohl mit dem Code als auch mit der Dokumentation dazu. Um einen groben Überblick über den Umfang des Projektes zu geben, zeigt Tabelle 3 die Menge an Code und Dokumentation. Zudem ist festzuhalten, dass alle unter Punkt 1 aufgezählten Ziele der Software erreicht wurden. Das LValue-Modul ist im Administrationsbereich integriert und ist bereits in Benutzung. Es ist in Planung das Modul um eine SQL-Skript-Generierung für die Datenbankskripte im DEV-Bereich zu erweitern und mit KI-Übersetzung auszustatten.

Scope	**Lines of Code**	**Lines of Comments**
Serverside	3.217	1.496
Clientside	2.919	98
Total	6.136	1.598

Table 3: Überblick über die Codebasis des LValue-Moduls