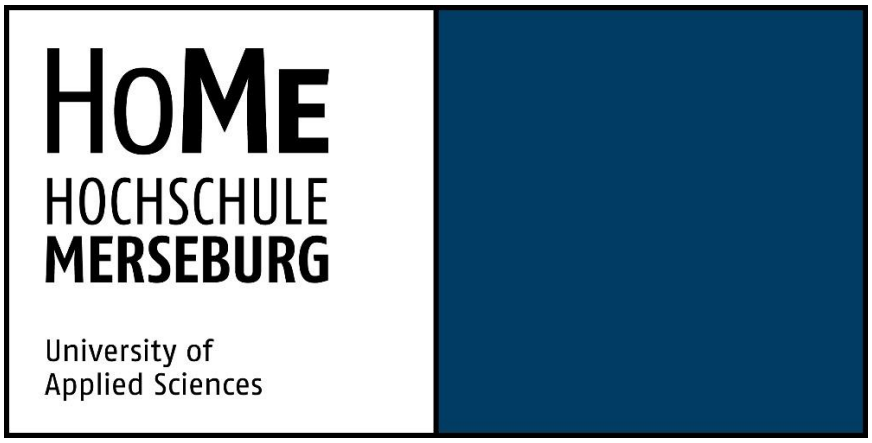
Hochschule Merseburg

University of Applied Sciences



Fachbereich Ingenieur- und Naturwissenschaften

Masterarbeit zur Erlangung des akademischen Grades

Master of Engineering – Informatik- und Kommunikationssysteme

Abrechnungsbeleg-Analyser für Regressionstests im SAP IS-U

Erstbetreuer: Prof. Dr. rer. pol. Uwe Schröter

Zweitbetreuer: Dipl.-Wirtsch.-Inf. Pascal Kovacs

eingereicht von Sascha Fehst (18696)

Merseburg, den 22.05.2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

ABAP	– Advanced Business Application Programming
ALV	– SAP List Viewer
DDIC	– Data Dictionary
FuBa (FM)	– Funktionsbaustein (Function Module)
IDE	– Integrated Development Environment
IS-U	– Industry Solutions for Utilities
MVC	– Model View Controller
PAI	– Process After Input
PBO	– Process Before Output
RLM	– Registrierende Leistungsmessung
RTTS	– Runtime Type Services
WYSIWYG	– What you see is what you get

Glossar

Commit	– Ein Befehl um Datenbankoperationen zu schreiben.
Customizing	– Die SAP Anwenderschnittstelle um Steuerdaten zu ändern.
Dynpro	– Ein Dynamisches Programm (Anzeigeklasse).
Feldsymbol	– Eine Variable die ähnlich wie ein Pointer funktioniert.
Funktionsbaustein	– Die Kapselung von Code innerhalb eines globalen Moduls.
Mandant	– Ein technisch abgetrennter Benutzer oder Gruppe.
Mockup	– Eine Skizze für Programmoberflächen.
Netweaver	– Der SAP Applikationsserver.
Pflegetabelle	– Ein persistentes Datenbankobjekt mit Nutzerschnittstelle.
Screen Painter	– Der WYSIWYG Editor für Dynpro.
Steuerdaten	– Ein Parameter zur Veränderung der technischen Prozesse.
Usability	– Benutzerfreundlichkeit

1 Einleitung

1.1 Motivation

Elektrischer Strom in den Haushalten ist kaum ersetzbar. Der Alltag wird bestimmt durch größere und kleinere Geräte, die diesen benötigen. Entsprechend groß ist daher der Bedarf an elektrischen Strom, der von einer Vielzahl von Lieferanten an den Endkunden geliefert wird. Für einen guten Überblick über die abgerechneten Tarife und deren Konditionen stellt SAP IS-U[1] Abrechnungsprozesse zur Verfügung. Diese enthalten mehrere Verwaltungskomponenten aus kaufmännischer und technischer Sicht. Um alles korrekt abbilden zu können, sind IS-U Abrechnungsprozesse komplex und umfangreich, weil Lieferanten immer mehr Konstellationen aus Tarifen, Konditionen, Abgaben und Steuern berücksichtigen müssen.

Die GISA GmbH betreut als IT-Dienstleister für große Energielieferanten besonders umfangreiche SAP IS-U Systeme und entwickelt diese weiter. Der manuelle Testaufwand stellt dabei eine ernste Herausforderung dar. Im IS-U Prozess kommt es regelmäßig vor, dass Parameter verändert werden müssen, um den Anforderungen des Vertriebs gerecht zu werden. Dadurch ergibt sich ein hoher Testaufwand für die komplexen Abrechnungsprozesse. Abrechnungsbelege werden dazu gebraucht, um eine Rechnung zu erstellen und die Posten für einzelne Leistungen festzuhalten. Ein Abrechnungsbeleg ist ein elektronisches Dokument, welches Belegzeilen für den jeweiligen Kunden und Vertrag enthält. In diesen Belegzeilen sind der Tarif, die Menge und der resultierende Preis festgehalten.

Das Werkzeug soll eine Aufwandsminimierung sicherstellen und gleichzeitig eine kundenorientierte Basis bereitstellen, die den weiteren Anforderungen entspricht. Damit spätere Anforderungen sich ohne Probleme implementieren lassen, steht die Erweiterbarkeit und der modulare Aufbau im Fokus des Programms. Da die komplexe Struktur des IS-U Abrechnungsprozesses aus Lieferanten-, Verteilnetzbetreiber- und Messstellenbetreiberseite besteht, müssen viele Ausnahmefälle berücksichtigt werden. So können beispielsweise ungewollte Seiteneffekte auftreten. Zusammengefasst lässt sich über den IS-U Abrechnungsprozess sagen, dass die Basis die IS-U Stammdaten bilden.

[1] IS-U steht hierbei für Industry Solutions for Utilities und stellt ein Modul im SAP dar.

Wie beispielsweise Vertrag, Anlage und Ableseergebnisse eine Berechnung verschiedener Bestandteile ermöglicht. Dabei werden diese gesteuert durch die Abrechnungsstammdaten wie Tarif, Tariftyp und Fakten. Als Zwischenergebnis werden die Daten in einem Abrechnungsbeleg zur Weiterverarbeitung (Rechnungslegung, Formulardruck) abgelegt.

1.2 Problemstellung

Um ungewollte Seiteneffekte des IS-U Abrechnungsprozesses zu verhindern, können manuelle Sichttests auf Abrechnungsbelegen vor und nach der Änderung durchgeführt werden. Die häufigsten Änderungen können veränderte Tarife und Mengen sein. Beim Tarif sind es die Konditionen, wie der Preis oder Rabatte auf diesen. Diese manuellen Tests sind einzeln sehr aufwändig durchzuführen. Hierdurch entsteht ein erhöhter Testaufwand. Außerdem erhöhen die kundeneigenen Entwicklungen im SAP den Testaufwand zusätzlich, da beispielsweise die verwendeten Spalten einer Datenbanktabelle erweitert oder gesamte Abläufe verändert wurden. Hierzu werden ein Referenzbeleg und ein Testbeleg verglichen. Der Referenzbeleg ist der erste und der Testbeleg der zweite zu vergleichende Abrechnungsbeleg. Das Problem betrifft vor allem die Anwender der Abrechnung.

Da dies Zeit und Kosten verursacht, ist die Frage nach einem Werkzeug aufgekommen, welches Abrechnungsbelege vor und nach Anpassungen miteinander vergleicht und die Unterschiede zwischen diesen aufzeigt. Es gibt neben technischen Kenntnissen auch fachliche Prozesse, deren Verständnis erforderlich ist, um eine Lösung des Problems zu gewährleisten und ein funktionierendes Werkzeug zu erstellen. Ein Werkzeug zur personellen Aufwandsminimierung ist sinnvoll und kostensenkend.

1.3 Zielstellung

Im Rahmen dieser Arbeit ist ein prototypisches Analysewerkzeug, zum Abgleich beliebiger Sammlungen von Abrechnungsbelegen vor und nach Änderungen, zu entwickeln. Es wird dabei untersucht, wie der Aufwand zum Testen der Auswirkung von Anpassungen der IS-U Abrechnungsprozesse minimiert werden kann. Dazu wird zusammen mit den Beratern der GISA GmbH eine Anforderungsermittlung durchgeführt und anschließend ein Prototyp implementiert, welcher die drei folgenden ermittelten Kernfunktionalitäten bereitstellen soll:

- Vergleich zweier Abrechnungsbelege,
- Verwalten von Abrechnungsbelegen,
- Erzeugen eines simulierten Abrechnungsbelegs.

Der Schwerpunkt der Arbeit liegt in der Lösung des Problems der Minimierung des manuellen Aufwands durch das Testen der Seiteneffekte und der Erarbeitung eines funktionalen Designs. Die Thematik der Regressionstests und des Testens allgemein soll nicht den Fokus dieser Arbeit bilden.

Damit nicht nur Entwickler und Berater dieses Werkzeug verwenden können, sollen ebenfalls fachliche Tester des Kunden als Zielgruppe hinzugezogen werden. Im Rahmen eines GISA GmbH internen Meetings ist die Idee eines Werkzeugs zur Vereinfachung des Testaufwands erarbeitet und von mehreren Fachkollegen bestätigt worden. Die GISA GmbH ist Themensteller dieser Arbeit und interessiert an einer Lösung des Problems. Dieses Tool wird in der SAP eigenen Programmiersprache ABAP entwickelt und mithilfe des Workbench im Netweaver[2] als IDE erarbeitet. Konkrete Anwendungsfälle sind Entwickler- bzw. Beratertests von Anpassungen und Tester vom Kunden, welche das Werkzeug benutzen werden. Es soll der Aufwand minimiert werden, einzelne oder mehrere Abrechnungsbelege zu vergleichen und die Unterschiede zu lokalisieren.

1.4 Aufbau der Arbeit

Das erste Kapitel dient als Einleitung in die Thematik des Abrechnungsbeleg-Analysers. Im zweiten Kapitel werden die Grundlagen aus fachlicher und technischer Herkunft veranschaulicht. Die ABAP Grundlagen sind in den technischen Grundlagen enthalten und bieten einen Überblick über die verwendeten Technologien und Werkzeuge innerhalb der Programmiersprache von SAP. Das dritte Kapitel thematisiert die Anforderungen, welche innerhalb von firmeninternen Meetings besprochen wurden. Diese sind in textueller und grafischer Form dokumentiert. Anschließend erfolgt im vierten Kapitel die Vorstellung des Ergebnisses der Entwicklung. Neben der Präsentation der Funktionalität des Prototyps wird dabei auch ein Überblick über die Implementierung gegeben sowie auf Probleme und Lösungen während der Entwicklung eingegangen. Das Kapitel schließt mit einer Bewertung der Ergebnisse durch die beiden Abrechnungsexperten der GISA. Um die Arbeit abzurunden, enthält das fünfte und letzte Kapitel ein abschließendes Fazit. Zudem wird ein Ausblick über Verbesserungen und Erweiterungen gegeben.

[2] Der SAP Netweaver ist der Applikationsserver auf dem die Programme ausgeführt werden.

2 Grundlagen

2.1 Theoretische und fachliche Grundlagen

2.1.1 Definition IS-U

Die Definition des IS-U ist wie folgt zusammengefasst:

„Elektrizitäts-, Gas-, Wasserversorger und andere Unternehmen der Branche optimieren und überwachen mit SAP IS-U ihre Netze, können Kunden und Interessenten verwalten sowie abrechnen und erhalten einen Überblick über diese in Echtzeit. Neben der Verwaltung von Kundenstammdaten und der Abrechnung beinhaltet die Software Funktionalitäten für das Gerätemanagement, die Instandhaltung, Verkauf, Service und Buchhaltung.“[3]

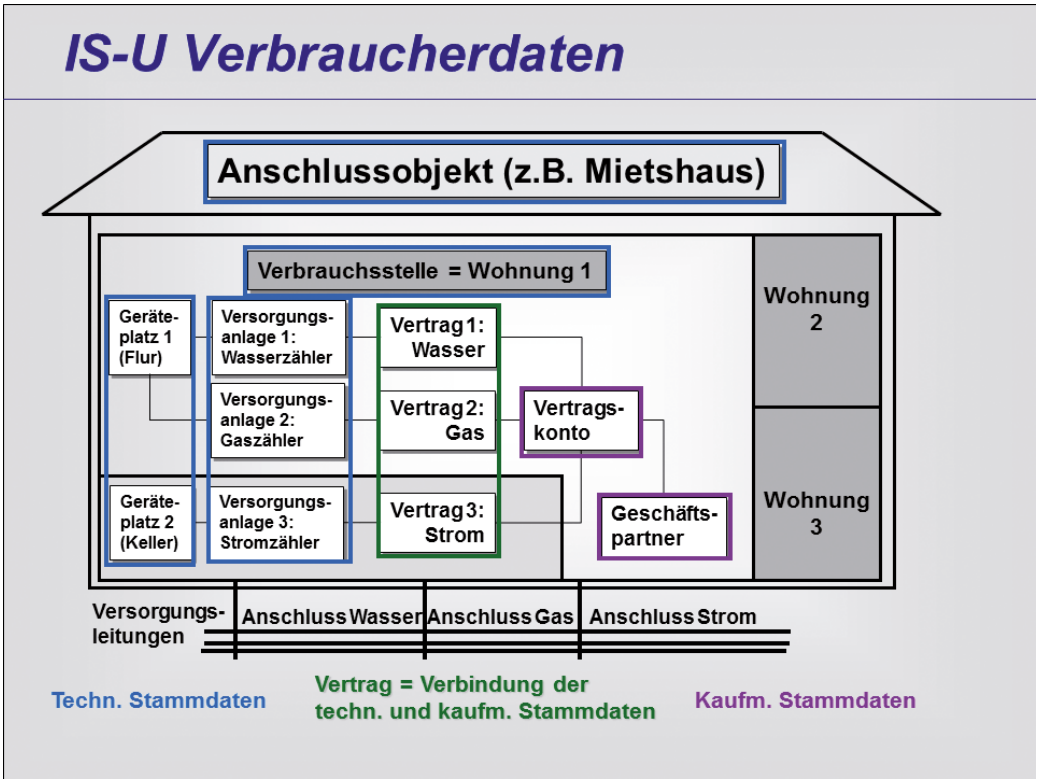


Abbildung 1 IS-U Verbraucherdaten[4]

Da die Nutzer des SAP IS-U große Datenmengen anlegen, um die Kunden abzubilden und trotzdem einen Überblick über diese behalten müssen, gibt es das Modell der IS-U Verbraucherdaten. Siehe hierzu Abbildung 1.

---

[3] https://mindsquare.de/knowhow/sap-is-u/ 19.12.19

[4] https://its-service.de/assets/img/projekte/stadtwerke-iserlohn-2011/screen-5_full.png 27.03.2020

Die IS-U Verbraucherdaten werden in verschieden verknüpften Datensätzen angelegt. Zuerst muss der Datensatz zu einem Geschäftspartner angelegt werden. Dieser stellt eine Person, Organisation oder Gruppe dar, welche/r ein oder mehreren Vertragskonten zugewiesen werden. In einer dieser Vertragskonten können ein oder mehrere Verträge, die abrechenbare Services (Dienstleistungen) darstellen, angelegt werden. Dem Vertrag ist immer eine Anlage zugewiesen. Er ist branchenspezifisch. Dazu ist dieser Anlage ein Gerät, beispielsweise ein Zähler zugeordnet. Dieser Zähler kann abgerechnet werden aufgrund seines Zählerstands zum jeweiligen Tarif.

Um nun aus diesen Datensätzen genaue Rechnungen zu erhalten, werden Abrechnungsbelege verwendet, um einzelne Daten zu erfassen. Der Rechnungsbeleg entsteht am Ende der Abrechnung. Laut der SAP Dokumentation werden Abrechnungsbelege wie folgt definiert:

„Enthält die Informationen, die die Fakturierung im Vertragskontokorrent für die Abrechnung erbrachter Leistungen gegenüber einem Kunden benötigt. Diese externen Belegdaten speichert das Vertragskontokorrent in Form von Abrechnungsbelegen.“[5]

Mit Fakturierung ist hier die Rechnungsstellung definiert.

Um die zahlreichen termingebundenen Daten zu erfassen, werden verschiedene Abrechnungsverfahren verwendet.

Es existieren folgende Abrechnungsverfahren:

- „Turnusabrechnung (jährliche oder monatlich regelmäßige Abrechnung)
- Gleitende Nachberechnung (Berechnung aller bisherigen Abrechnungen)
- Endabrechnung (Abrechnung zum Ende einer Turnusperiode)
- Zwischenabrechnung (manuelle Abrechnung)
- Schlussabrechnung (Abrechnung zum Vertragsende)“[6]

Im Folgenden wird vorrangig die Turnusabrechnung und der Bezug zum Prototypen betrachtet. (Siehe hierzu Abbildung 2). Als erster Schritt muss ein Ableseauftrag angelegt werden. Gleichzeitig dazu wird ein Abrechnungsauftrag erstellt. Beide erfassen die Ablesedaten und es gibt bei unplausiblen Ableseergebnissen eine Verfahrensweise.

---

[5] https://help.sap.com/viewer/efcbb384e71c48b99370ea411db1cf58/6.00.31/de-DE/20f5c5536a51204be10000000a174cb4.html 05.03.2020

[6] Jörg Frederick, Tobias Zierau SAP for Utilities S.228 2011

Diese heißt Ableseergebniskorrektur und muss erfolgen bis plausible Ableseergebnisse erfasst werden. Mit diesen Daten wird nun die Abrechnung angestoßen, welche als Ergebnis einen Abrechnungsbeleg erstellt. Dieser Abrechnungsbeleg in der Datenbank wird vom Abrechnungsbeleg-Analyser über die Abrechnungsbelegsnummer angefordert.

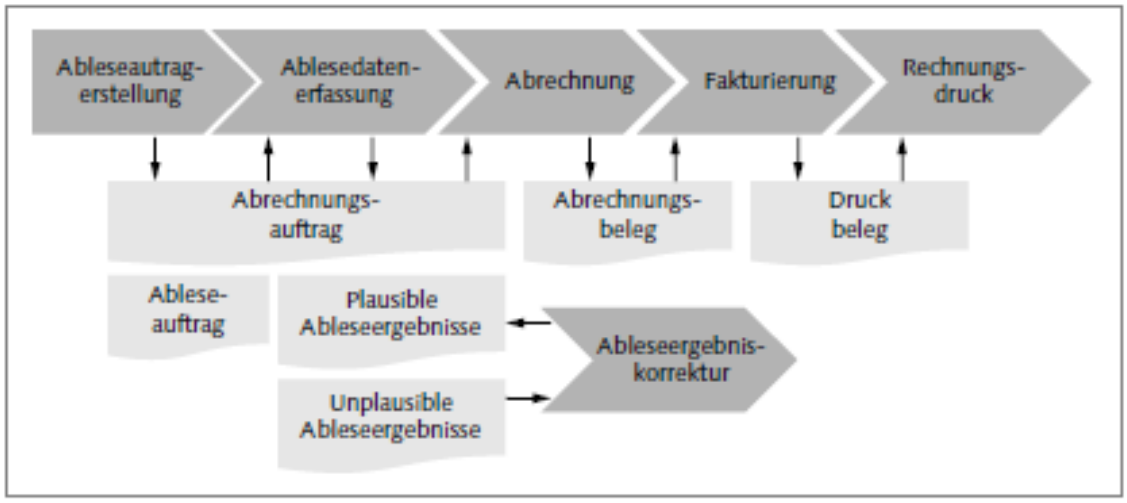


Abbildung 2 Ablauf der Abrechnung[7]

Die nachfolgenden Schritte, losgelöst von der Turnusabrechnung, also die Fakturierung und der Rechnungsdruck über einen Druckbeleg, werden nicht davon berührt.

[7] Jörg Frederick, Tobias Zierau SAP for Utilities S.229 2011

2.1.2 Einordnung des IS-U Moduls in SAP

In der Abbildung 3 sind SAP-Module aufgelistet, welche an einem Beispiel-Netweaver eingebunden sind.

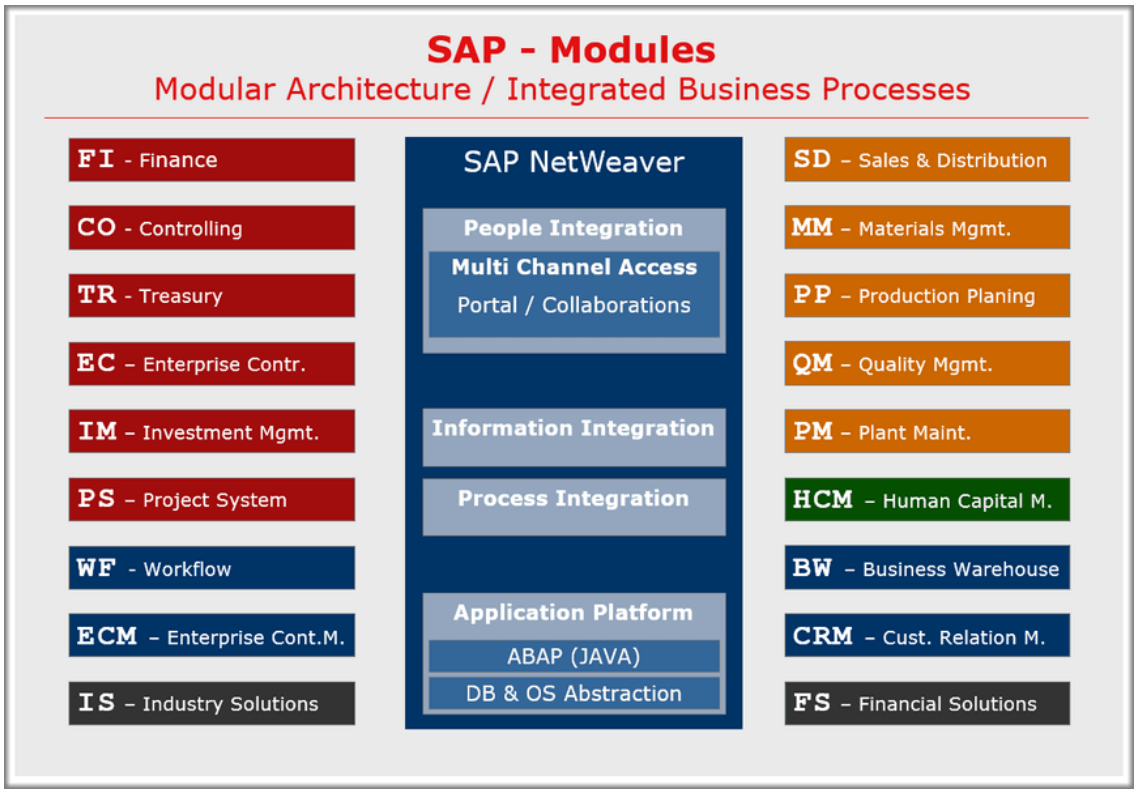


Abbildung 3 SAP Module[8]

Im Modul IS-U gibt es neben Versorgern (IS-Utilities) auch andere wichtige Bereiche:

- „IS-A Automobil (Automotive)
- IS-R Einzelhandel (Retail)
- IS-RE Immobilienwirtschaft (Real Estate)“[9]

Für Unternehmen, die eine große Anzahl an Kunden mit dazugehöriger großer Verwaltung besitzen, wird aus der Vielzahl an Modulen eine allumfassende Lösung angeboten. Diese Unterteilung ist sinnvoll, damit Kunden nicht den gesamten Modulkatalog buchen müssen, sondern ein Angebot, zugeschnitten auf ihre Bedürfnisse, erhalten können.

[8] https://www.stichpunkt.de/sap/module.html 23.12.2019

[9] https://www.stichpunkt.de/sap/module.html 23.12.2019

2.2 Technische Grundlagen

2.2.1 SAP und ABAP

„ABAP ist die proprietäre Programmiersprache von SAP“[10]

Diese kann sowohl prozedural als auch objektorientiert genutzt werden. Die Workbench Elemente werden im DDIC (Data Dictionary) festgehalten. Das DDIC stellt alle globalen Objekte, die sich in einer zusammengefassten Einheit befinden, dem sogenannten Paket, zur Verfügung. Der ABAP Workbench ist die IDE im Netweaver von SAP. Die fachlichen Prozesse, welche in einem SAP System hinterlegt sind, brauchen aus technischer Sicht einen Speicherort für große Datenmengen. Diese sind in nicht-normalisierten Tabellen abgelegt in einer im SAP System integrierten Datenbank. Darauf kann schließlich dann mittels Funktionen aus dem SAP Standard Coding oder direkt zugegriffen werden. Hierbei werden SQL-Statements verwendet, um Daten abzufragen. Mit SAP Standard Coding sind die in ABAP geschriebenen Funktionen gemeint, um beispielsweise per Funktionsbaustein einen wiederkehrenden Programmteil für Anfragen an die Datenbank zu gewährleisten. Ein Funktionsbaustein ist eine Möglichkeit in ABAP lokal verwendetes Coding auszulagern, um wiederholende Zugriffe innerhalb eines oder mehrerer Programme global möglich zu machen.

Um nutzerspezifisch die Berechtigungen zu trennen, gibt es in SAP die Mandantenfähigkeit. Das bedeutet, dass die Organisation von beispielsweise Tabellen und Berechtigungen getrennt, abhängig vom angemeldeten Benutzer ist. Jeder Mandant kann über seine Daten exklusiv verfügen. Ein Anwender hat beispielsweise weniger Rechte als der Administrator.

Im Workbench des Netweaver können SAP Programme in ABAP erstellt und getestet werden. Das Testen von Änderungen im gesamten Programm wird Regressionstest genannt. Die Definition davon ist:

“Unter einem Regressionstest (von lateinisch regredior, regressus sum ‚zurückschreiten‘) versteht man in der Softwaretechnik die Wiederholung von Testfällen, um sicherzustellen, dass Modifikationen in bereits getesteten Teilen der Software keine neuen Fehler („Regressionen“) verursachen.”[11]

---

[10] https://mindsquare.de/knowhow/abap/ 15.04.2020

[11] https://www.testbirds.de/leistungen/quality-assurance/regressionstest/ 05.03.2020

2.2.2 Strukturen und interne Tabellen

Strukturen können als Typ die Zeile einer persistenten oder internen Tabelle haben. Eine interne Tabelle ist eine relationale, nicht persistente Tabelle. Es kann über eine interne Tabelle eine Schleife laufen, die dann in eine Struktur schreibt und diese direkt ausgelesen werden kann. Beide können lokal oder global definiert werden. Die lokale Definition ist direkt im Quellcode und eine globale Definition findet im DDIC statt. Es gibt bei Strukturen einen Unterschied zwischen flachen und tiefen Strukturen. Eine flache Struktur ist in Abbildung 4 zu sehen.

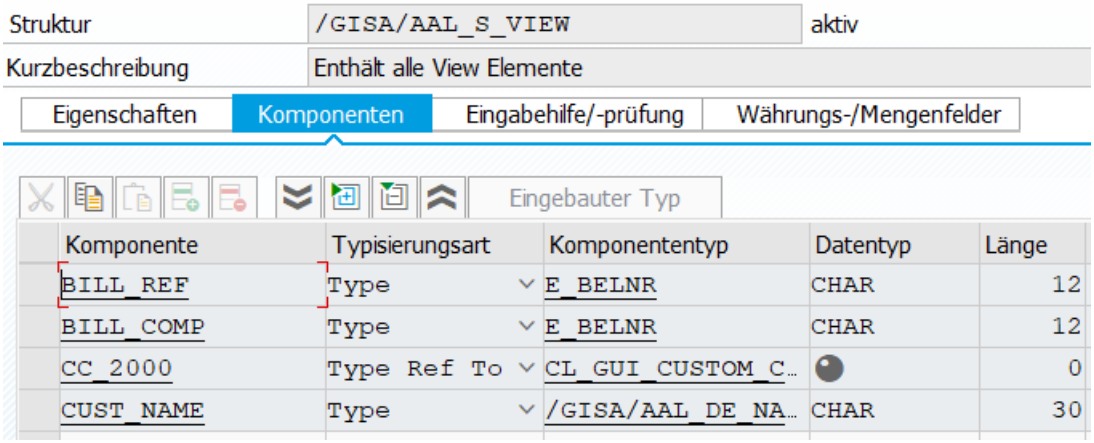


Abbildung 4 Flache Struktur Workbench

Die Komponenten sind die Namen der Spalten der Struktur und es können auch Klassen hinterlegt werden als Typen, die dann als Objektreferenzen abgelegt werden können.

Als nächstes Beispiel wird in Abbildung 5 eine tiefe Struktur gezeigt:

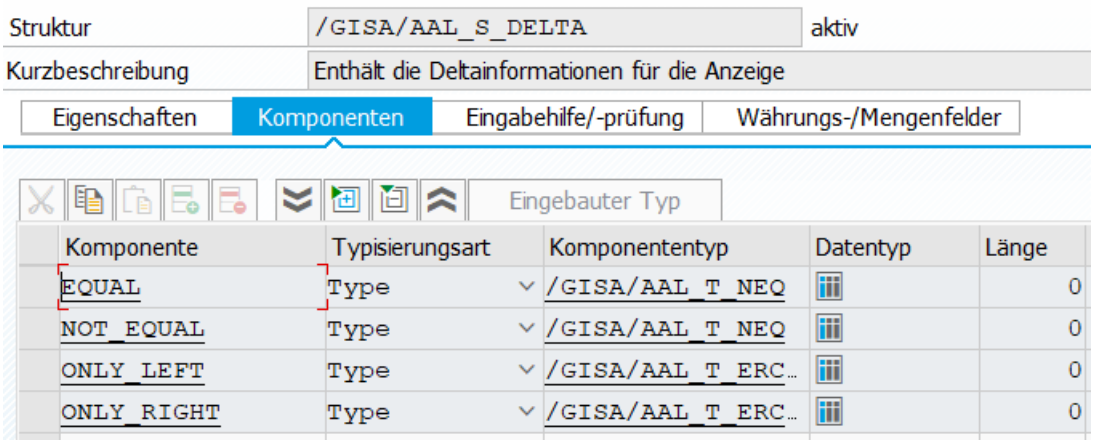


Abbildung 5 Tiefe Struktur Workbench

Der Datentyp der Spalten in der Struktur sind Tabellentypen. Der Unterschied ist jener, dass bei flachen Strukturen nur elementare oder Klassen-Datentypen genommen werden und bei tiefen Strukturen zusätzlich Tabellentypen zum Einsatz kommen.

Um den Unterschied zwischen einer tiefen Struktur und einer flachen Struktur aufzuzeigen dient die Abbildung 6. Diese zeigt eine abstrakte Darstellung einer flachen Struktur mit einfachen Feldern. Es ist zu sehen, dass es nur elementare Datentypen sind ohne Referenzen.

Abbildung 6 Flache Struktur Abstrakt[12]

Bei einer tiefen Struktur sind die Felder referenziert auf Tabellentypen und die Felder selbst können auf andere Typen zeigen. In der Abbildung 7 ist zu sehen, dass die Felder einer Tabelle auf eine Struktur verweisen, die dann einen elementaren Datentyp haben kann.

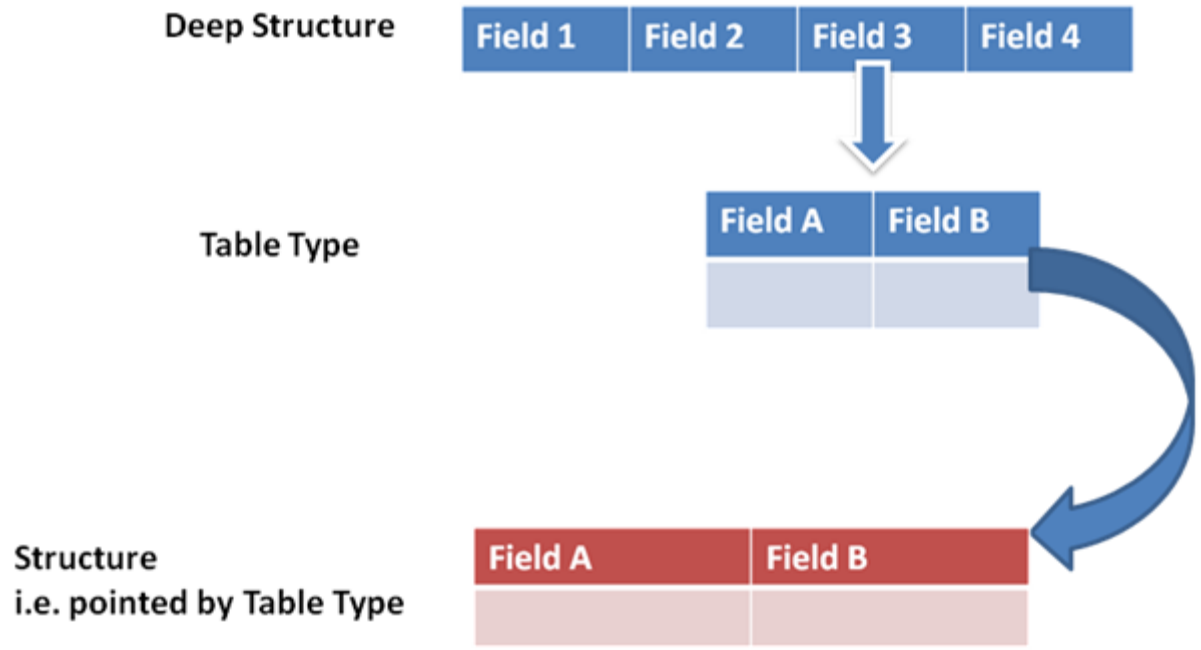


Abbildung 7 Tiefe Struktur Abstrakt[13]

Für die Implementierung ist dies ein wichtiger Bestandteil, da es ohne interne Tabellen, tiefen und flachen Strukturen nicht möglich ist, mit großen Datenmengen zu arbeiten und diese in Schleifen auszulesen und zu verändern.

[12] https://gocoding.org/structure-in-sap-abap 15.04.2020

[13] https://gocoding.org/structure-in-sap-abap 15.04.2020

2.2.3 SAP List Viewer

Den SAP List Viewer (ALV) kann man mit Funktionsbausteinen oder durch statische Klassenmethoden aufrufen. In dieser Arbeit wird nur der objektorientierte Ansatz verfolgt, die Möglichkeit des Aufrufs durch Funktionsbausteine wird nicht behandelt. Nachfolgend ein Code-Beispiel in Listing 1, um den Aufruf des ALV zu veranschaulichen.

Mit diesem statischen Aufruf einer Klassenmethode in Zeile 2 wird eine ALV_Tabelle erzeugt. Ausgehend von einer Eingabetabelle (Zeile 8) und Eingabeoberfläche (Zeile 6). In Zeile 12 erfolgt der Befehl für die Anzeige.

```
1       TRY.
2           CALL METHOD cl_salv_table=>factory
3             EXPORTING
4               list_display = sy-batch
5             IMPORTING
6               r_salv_table = lr_alv
7             CHANGING
8               t_table      = it_table.
9         CATCH cx_salv_msg.
10          MESSAGE 'Anzeige konnte nicht erstellt werden.' TYPE 'E'.
11      ENDTRY.
12    lr_alv->display( ).
```

Listing 1 ALV Tabelle

Abrechnungsbeleg

Mandant	AbrechBelegnr.	BelZeile	BruttoZeiB	System	Int.Schrtt	BelZArt	AbslZeile	Differenz	BuchRel.RZ	MengStaRel	Betr.StRel	StGrMenge	StGrBetrag	Druckrel.	AbrKlasse
100	610000012127	1			1	YQUHTG								X	V101
100	610000012127	2			4	Z99999								X	V101
100	610000012127	3			4	XQPHT			X	X	X	ZMAPHT	ZBAPHT	X	V101
100	610000012127	4			5	Z99999								X	V101
100	610000012127	5			5	XQPNT			X	X	X	ZMAPNT	ZBAPNT	X	V101
100	610000012127	6			6	XLPGP			X	X	X	000000	ZBGP	X	V101
100	610000012127	7			13	YQUHTG	X							X	V101
100	610000012127	8			15	Z99999	X							X	V101
100	610000012127	9			15	XQPHT	X		X			ZMAPHT	ZBAPHT	X	V101
100	610000012127	10			16	Z99999	X							X	V101
100	610000012127	11			16	XQPNT	X		X			ZMAPNT	ZBAPNT	X	V101
100	610000012127	12			17	XLPGP	X		X			000000	ZBGP	X	V101

Abbildung 8 SAP List Viewer Ergebnis

Als Ergebnis wird in Abbildung 8 eine Tabelle angezeigt, die mit it_table übergeben wurde. ALV Tabellen können mithilfe von Einstellungsmöglichkeiten in den Steuerdaten veränderte Ergebnisse anzeigen. Eine Einstellungsmöglichkeit ist beispielsweise das Customizing im nächsten Kapitel.

2.2.4 Customizing

Eine Änderung in Steuerdaten erreicht man mit Parametern, die der Nutzer ändern kann. Diese werden in einer persistenten Tabelle in der Datenbank festgehalten, damit darauf nutzerseitig und über das Programm zugegriffen werden kann. Durch den sogenannten Pflegedialog kann der Nutzer vorhandene Eingabefelder benutzen, um eine Änderung in der Abarbeitung des Programms durchzuführen. In der Abbildung 9 sieht man ein Beispiel des Customizings. Angezeigt wird der Pflegedialog. Um eine Tabelle aufzubauen gibt es ein Programm, in welchem die Spalten festlegt werden mit Namen, Datenelement, Datentyp und anderen Einstellungen. Siehe Abbildung 10.

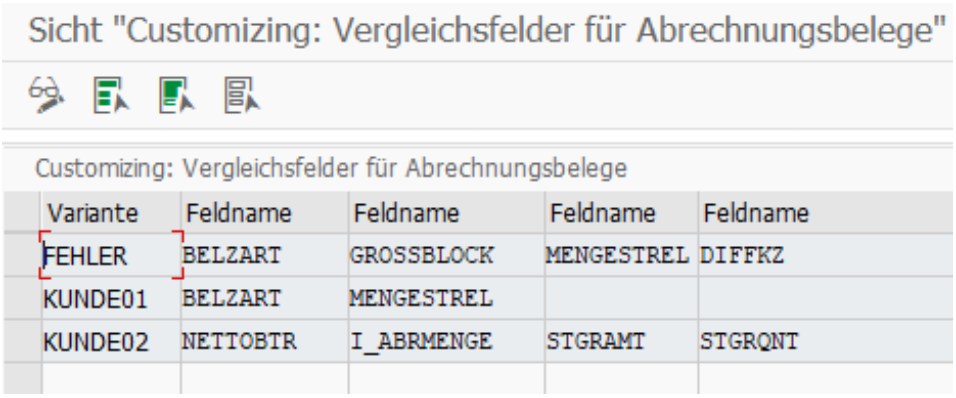


Abbildung 9 Customizing Pflegedialog

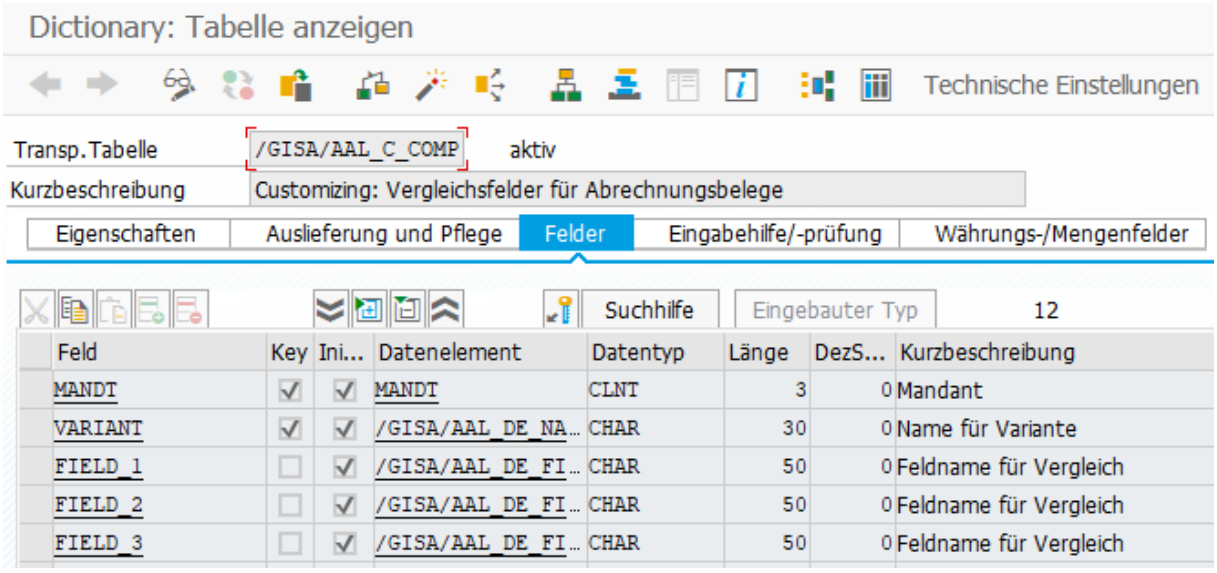


Abbildung 10 Aufbau Tabelle

Die erstellte Tabelle kann nun mit einem Pflegeview durch einen Tabellenpflegegenerator versehen werden. Damit kann die Tabelle durch den Benutzer gepflegt werden. (Siehe hierzu Abbildung 11).

Der Anwender muss diese Tabelle durch den Pflegedialog zuerst mit Einträgen füllen. Dadurch ist es möglich, sich die Einträge anzeigen zu lassen, diese zu verändern oder zu löschen.

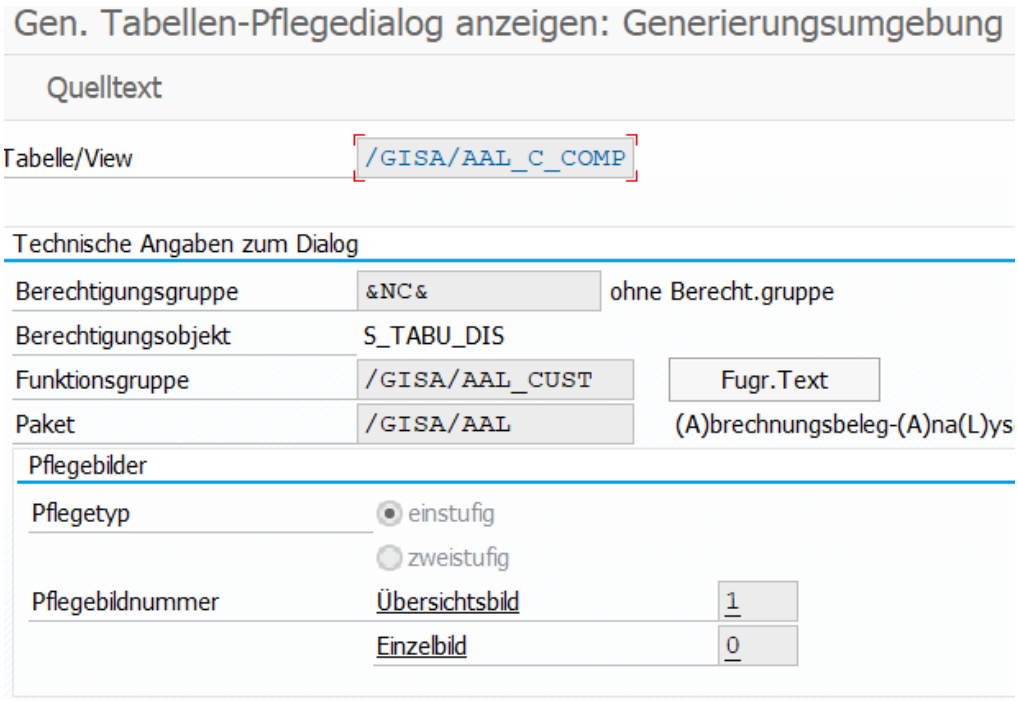


Abbildung 11 Tabellenpflegegenerator

Außerdem können die Daten in einer anderen Übersicht angezeigt werden. Dieser Vorgang verläuft unabhängig vom Pflegedialog und schafft einen besseren Überblick über die vorhandenen Daten. (Siehe hierzu Abbildung 12).

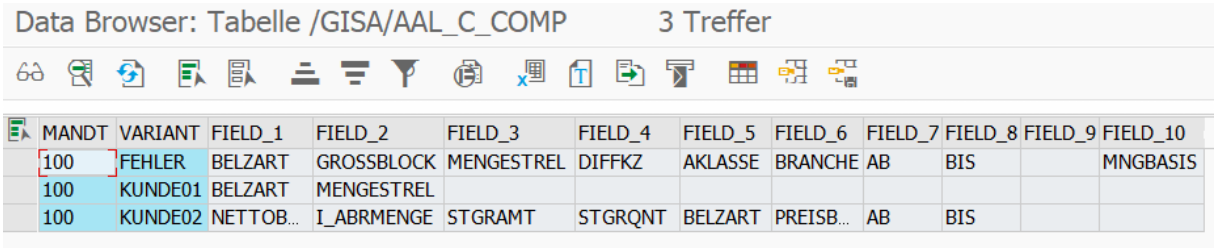


Abbildung 12 Tabelleninhaltsübersicht

Die erstellten Daten sind mandantenabhängig solange die Tabelle mit einer Mandantenspalte versehen ist.

2.2.5 Runtime Type Services

Die in SAP eingeführte Runtime Type Services sind für dynamische Programmierung wichtig. Denn mit diesen lassen sich zur Laufzeit Typen von Variablen bestimmen. Außerdem können zur Laufzeit auch neue Typen generiert werden. Wörtlich heißt es in der ABAP Dokumentation:

„Abkürzung RTTS. Zusammenfassung von RTTC und RTTI in einer Klassenhierarchie. Die Methoden der RTTS beschaffen Informationen zu Datentypen oder Klassen bzw. Interfaces von Objekten oder erzeugen neue Datentypen in der Form von Typbeschreibungsobjekten. Die RTTS sind als Hierarchie von Typbeschreibungsklassen implementiert, aus denen Typbeschreibungsobjekte erzeugt werden können.“[14]

RTTC steht dabei für Runtime Type Creation und RTTI für Runtime Type Identification. Durch die Methode `describe_by_data( <fs> )` wird von der Variable ein Objekt zusammengetragen, welches mehrere Daten darüber enthält wie beispielsweise die Komponentenliste. Ein Beispiel des Aufrufs der Komponentenliste wird in Listing 2 gezeigt:

```
1  lo_strucdescr ?= cl_abap_typedescr=>describe_by_data( <fs> ).
2  lt_components = lo_strucdescr->components.
```

Listing 2 RTTS

Damit ist es möglich, diese in einer Schleife auszugeben. Und mit diesem kurzem Code-Beispiel ist dynamisch die Liste der Komponenten einer Struktur ausgelesen worden. Hiermit können die technischen Typen der Spalten aus dem Customizing herausgelesen (RTTI) und als Typ eine neue Struktur erstellt werden (RTTC). Diese wird noch mit einer Color Spalte versehen, um die Färbung der ALV Tabelle zu gewährleisten.

[14] https://help.sap.com/doc/abapdocu_752_index_htm/7.52/de-DE/abenrun_time_type_services_glosry.htm 07.01.2020

2.2.6 Report

Der Report ist eine Klasse, um den Ablauf des erstellten Programms im Workbench zu starten und selbst ABAP Anweisungen auszuführen. Es werden Ereignisse wie Process Before Output (PBO) und Process After Input (PAI) ausgelöst und beispielsweise eine View, das sogenannte Dynpro (Dynamisches Programm), angezeigt. Der Process Before Output ist die Abarbeitung, bevor die Ausgabe am Bildschirm erfolgt. Nach einer Eingabe durch den Benutzer wird der Process After Input durchlaufen. Das Dynpro ist mit einem WYSIWYG Editor versehen (Screen Painter) und erstellt bestimmte Oberflächenelemente wie Buttons und Eingabefelder mit dem korrekten Listener automatisch. Durch Drag & Drop werden die Elemente in das mit Zeilen und Spalten begrenzte Dynpro gezogen. Es wird im Workbench das erste anzuzeigende Dynpro als Startausgabe festgelegt und in der Ablauflogik können weitere Dynpros angesteuert werden. Auf diese Weise können lokale und globale Elemente definiert werden. Lokale Elemente werden im Report an sich festgelegt oder in Klassen allgemein. Die Klassen selbst können global sein. Diese global definierten Elemente werden im DDIC hinterlegt und sind somit nutzbar im globalen Kontext.

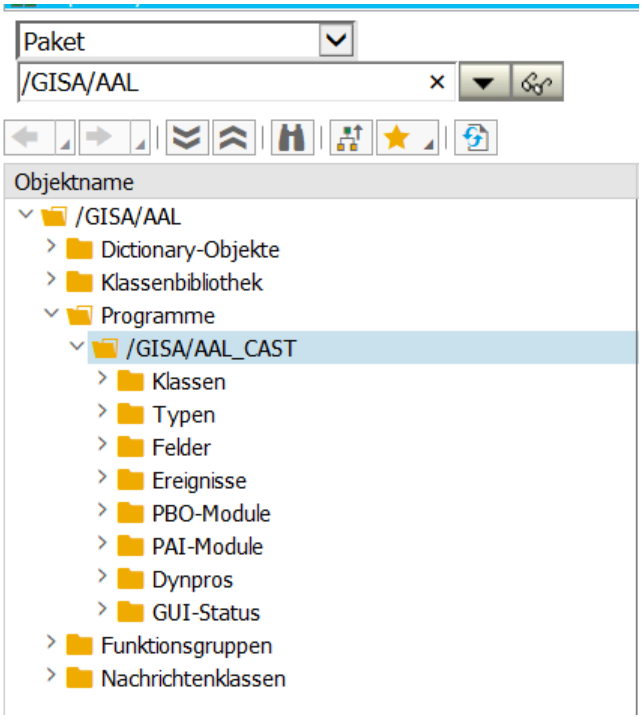


Abbildung 13 Report im DDIC

Das DDIC stellt das Repository dar. Als Beispiel Report im DDIC wird das Programm mit Namen angezeigt. (Siehe hierzu die Abbildung 13).

2.2.7 Suchhilfe

Suchhilfen sind ein technisches Werkzeug, um aus einer Datenbanktabelle die Einträge zu erhalten und in ein Eingabefeld zu schreiben. Außerdem können neben dem Primärschlüssel der Tabelle auch weitere Felder hinzugefügt werden, um die Suchhilfe zu präzisieren. (Siehe hierzu Abbildung 14). Unter dem Label Datenbeschaffung muss in dem Eingabefeld (Position 1) neben der Selektionsmethode eine Datenbanktabelle angegeben werden, um der Suchhilfe eine Tabelle mitzugeben, in die selektiert wird. Bei Suchhilfeparameter (Position 2) wird die Reihenfolge festgelegt, wie Spalten aus der Tabelle angezeigt werden sollen. Die relevanten Felder sind damit zu sehen. Diese müssen aber mit dem richtigen Datenelement versehen werden, um die gewünschten Inhalte zu finden.

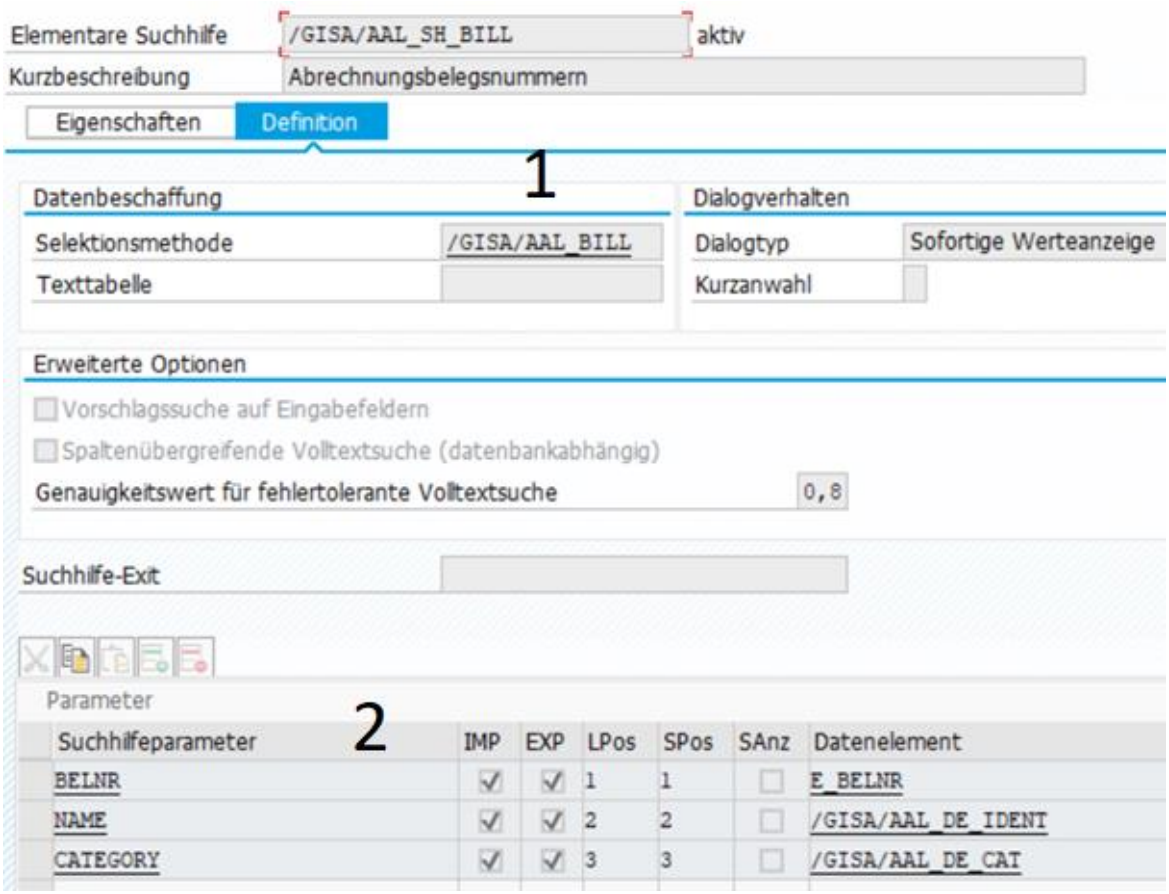


Abbildung 14 Suchhilfe

Prüftabellen dienen als weitere Hilfe, um die Anzeige von Daten zu gewährleisten. Durch diese Herangehensweise kann nun das Customizing (Siehe hierzu Kapitel 2.2.4) mit eingebunden werden.

2.2.8 Prüftabelle

Die Komponenten einer Struktur können mit einer Prüftabelle ausgestattet werden, um den Inhalt dieser Prüftabelle abzurufen. Dabei werden Fremdschlüsselfelder erwartet. Die Einrichtung erfolgt wie in Abbildung 15 abgebildet und das Ergebnis ist in Abbildung 16 zu sehen.

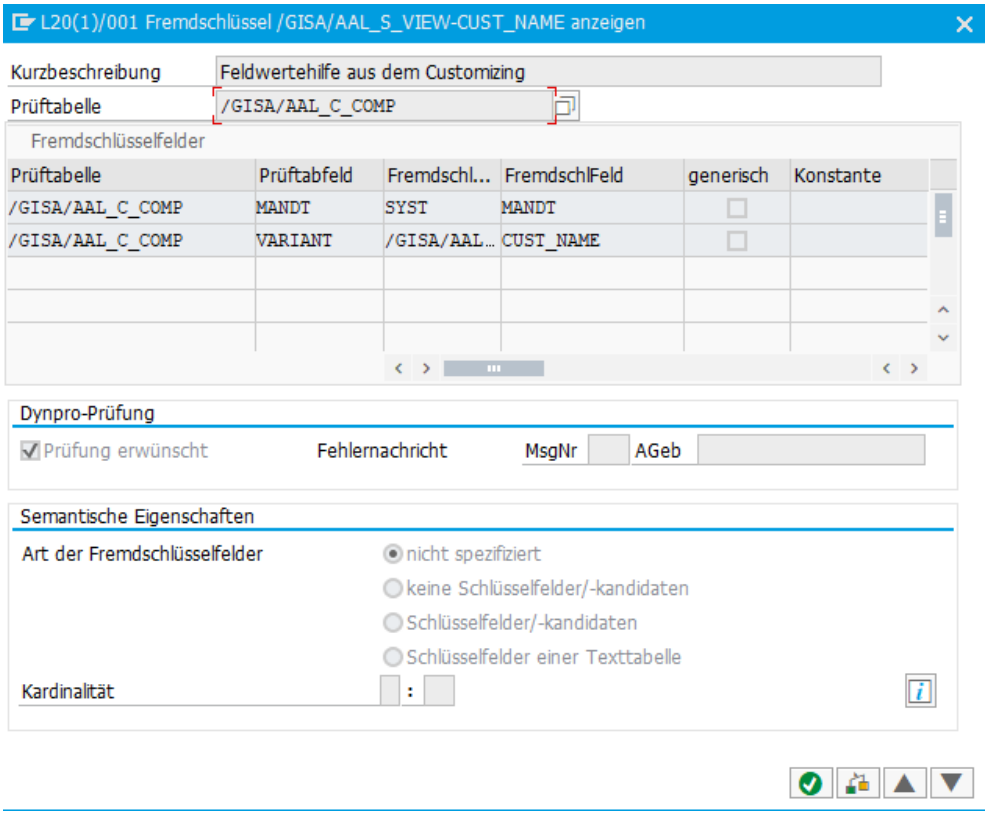


Abbildung 15 Prüftabelle Konfiguration

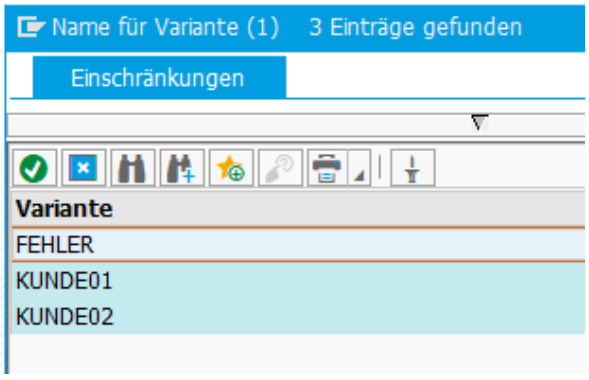


Abbildung 16 Prüftabelle Ergebnis

Mit diesem Kapitel sind die Grundlagen, die für die Erarbeitung des Prototyps nötig sind, definiert worden. Nachfolgend werden die Anforderungen des Prototyps aufgezeigt.

3 Anforderungsermittlung

3.1 Formulierte Anforderungen

Bei den Terminen zur Beschreibung der Lösung aus der Zielsetzung wurden durch die GISA GmbH folgende Anforderungen definiert:

- Vergleich zweier Abrechnungsbelege,
- Verwalten von Referenzabrechnungsbelegen,
- Erzeugen eines simulierten Abrechnungsbelegs (Basis Referenzbeleg).

Diese entsprechen auch den Teilzielen der Arbeit und werden im Folgenden definiert.

Anforderung 1: Vergleich zweier Abrechnungsbelege

Als erste Anforderung wurde der elementare Vergleich zweier Abrechnungsbelege festgelegt. Die direkte Eingabe der Belegnummern in der grafischen Oberfläche, anschließendes Auslesen aus der Datenbank mittels eines Funktionsbausteins und zuletzt das Anzeigen in einer Gegenüberstellung mithilfe einer Tabelle sind nötig, um die Anforderung zu erfüllen. Zusätzlich muss es noch ein Customizing geben. In diesem wird festgelegt, welche Felder verglichen werden sollen. Zudem sollen die Abrechnungsbelege durch das Customizing nicht vollständig angezeigt werden. Vielmehr soll durch den Benutzer eine Visualisierung von relevanten Feldern der Abrechnung stattfinden. Die Abrechnungsbelege werden somit auf die anzuzeigenden Felder reduziert. Das Customizing ist eine Pflegetabelle, die durch den Nutzer gepflegt wird, um Änderungen in den Steuerdaten schnell und einfach zu gewährleisten.

Anforderung 2: Verwalten von Referenzabrechnungsbelegen

Die eingegebenen Abrechnungsbelege können als Anwendungsdaten und somit als Referenzbelege mit einer Kategorie und einem Bezeichner abgelegt werden. Das vereinfacht den Vergleich und die Auffindbarkeit der zuletzt benutzten Belege, welche bei Bedarf wieder gelöscht werden können.

Anforderung 3: Erzeugen eines simulierten Abrechnungsbelegs (Basis Referenzbeleg)

Beim Auslesen des Referenzabrechnungsbelegs wird dessen Vertrag und der entsprechende Zeitraum ermittelt. Mit diesen Angaben kann nun ein aktueller Abrechnungsbeleg simuliert und für den Vergleich genutzt werden.

3.2 Anwendungsfälle

Abbildung 17 zeigt ein Mockup, welches aus den formulierten Anforderungen erstellt worden ist. Dieses stellt nur eine erste Idee einer Benutzeroberfläche dar. Das Mockup ist als Skizze zu betrachten in der das Vergleichen, das Generieren, das Speichern und das Löschen von Abrechnungsbelegen ermöglicht wird. Es müssen beim Vergleichen die Nummer des Referenzbelegs, des Testbelegs und der Name der Variante eingegeben und auf Vergleichen geklickt werden. Damit werden dann in der Anzeige gleiche Zeilen in grün, ähnliche Zeilen in rot und Zeilen, die sich nur in einem Beleg befinden orange angezeigt. Um einen Abrechnungsbeleg zu generieren, muss ein Referenzbeleg eingegeben werden und auf Generieren geklickt werden. Nun wird in das Feld des Testbelegs eine Nummer hineingeschrieben vom generierten Abrechnungsbeleg. Das Speichern und das Löschen sollen ausgewählte Belege speichern oder löschen.

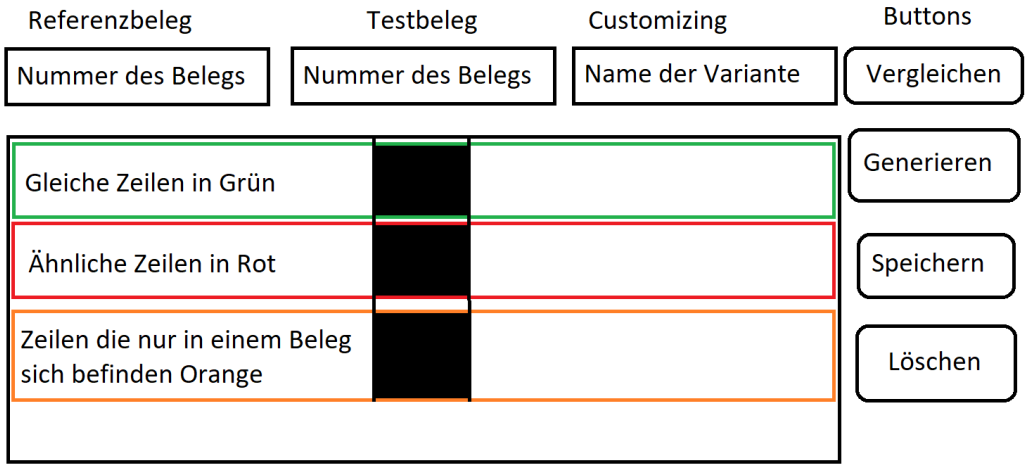


Abbildung 17 Mockup erste Idee

Abbildung 18 zeigt die formulierten Anforderungen in Form eines Use-Case-Diagramms. Zuerst sollte die Möglichkeit bestehen, einen Customizingeintrag anzulegen. Mit seiner Verfügbarkeit können die Einträge des Benutzers in die Datenbank für die spätere Verwendung abgelegt werden. Danach sollten Abrechnungsbelege verglichen und auch das Customizing geprüft werden können. Der Vergleich der Abrechnungsbelege ist auf den elementaren Vergleich durch Benutzereingaben im Customizing beschränkt. Das heißt, dass die Felder aus dem Customizing in den Prozess des Vergleichs und der Anzeige der Abrechnungsbelege hinzugezogen werden. Die Anzeige des Vergleichs der Abrechnungsbelege ist hierbei ebenfalls zu beachten. Dafür sollen mithilfe einer Tabelle die Gemeinsamkeiten und Unterschiede der beiden Abrechnungsbelege anschaulich dargestellt werden.

Zusätzlich können hierbei verschiedene farbliche Markierungen der Unterschiede und Gemeinsamkeiten der Abrechnungsbelege hinzugefügt werden.

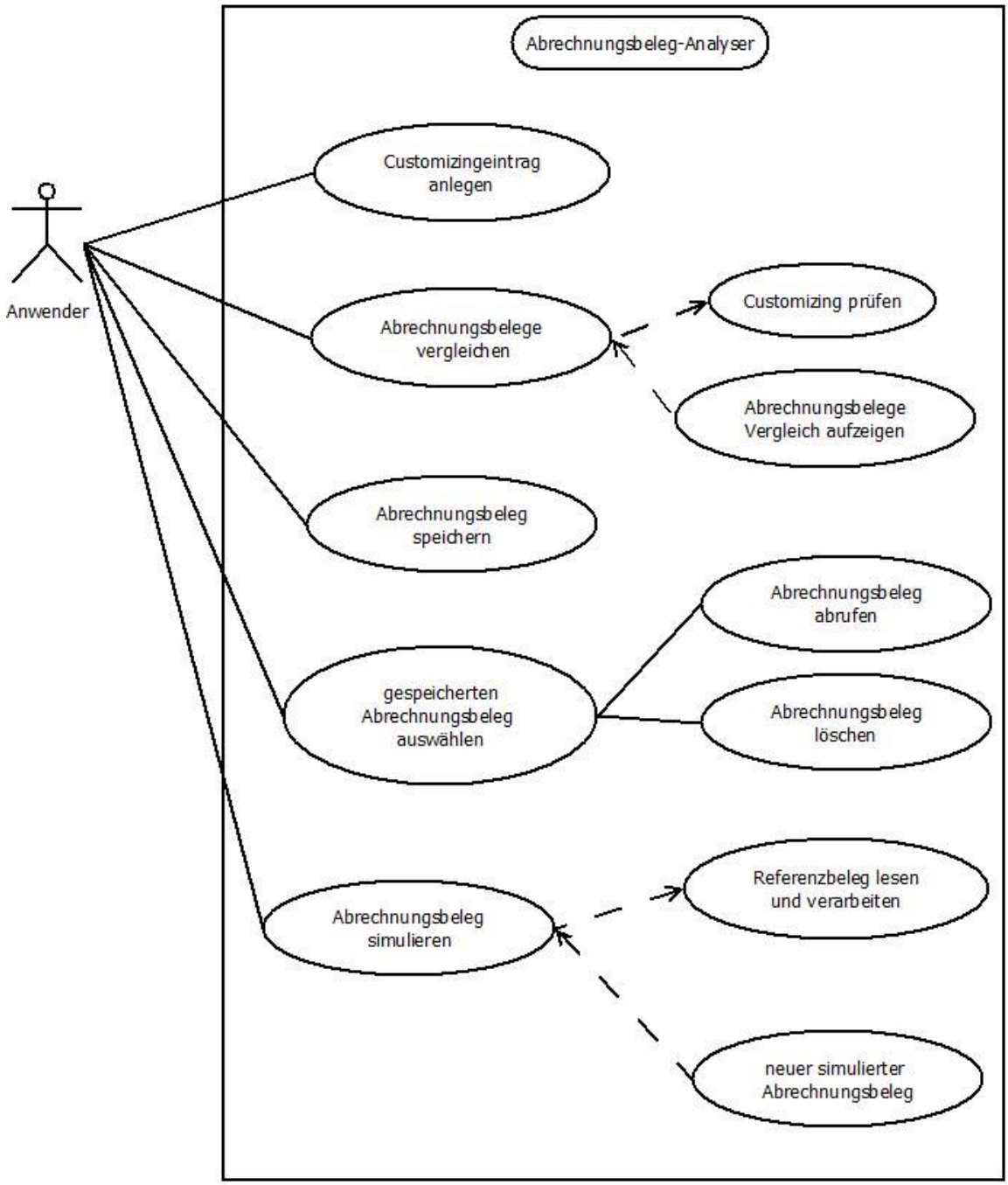


Abbildung 18 Use-Case-Diagramm

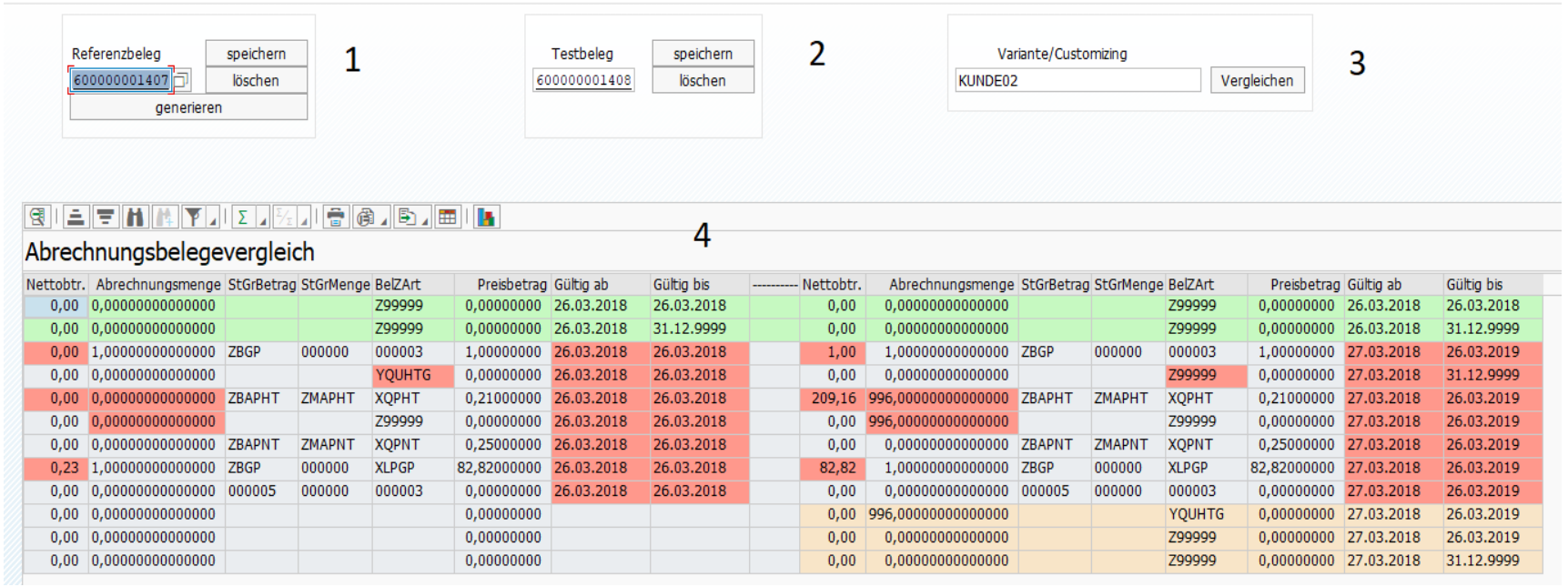
Eine entsprechende Gegenüberstellung ist hierbei wünschenswert. Dabei bildet das Speichern der Abrechnungsbelege eine weitere Anforderung. In diesem Schritt können Datensätze in einer Datenbank in eine persistente Tabelle mit Nummern, Kategorie und Bezeichner hinterlegt werden.

Anschließend werden mit einer Suchhilfe oder Wertetabelle gespeicherte Abrechnungsbelege ausgewählt und abgerufen. Der Abrechnungsbeleg soll aus der Liste der gespeicherten Abrechnungsbelege wieder gelöscht werden können. Als letzte Anforderung soll ein Abrechnungsbeleg simuliert werden. Das bedeutet, es muss der Referenzbeleg gelesen und verarbeitet werden. Mit den gewonnenen Daten aus dem Referenzbeleg kann ein neuer Beleg simuliert und angezeigt werden.

4 Prototyp

4.1 Funktionsbeschreibung

Abbildung 19, welche die Oberfläche des entwickelten Prototyps zeigt, verdeutlicht, dass an den Positionen 1 und 2 Eingabefelder für Abrechnungsbelegsnummern stehen, wobei diese Nummern mit einem Bezeichner und einer Kategorie gespeichert werden können. Zusätzlich ist es möglich, aus dem Beleg an Position 1 einen Beleg zu generieren, welcher dann bei Position 2 im Eingabefeld erscheint.
Bei Position 3 ist der Bezeichner der hinterlegten Felder im Customizing einzugeben. Es existieren bei allen drei Eingabefeldern Suchhilfen, welche mit der Taste F4 angesteuert werden können. Bei den Eingabefeldern für die Abrechnungsbelegsnummern kann mit Doppelklick auf die SAP Anzeige für Abrechnungsbelege navigiert werden. Mit einem Klick auf den Button „Vergleichen“ (Position 3) wird der Vergleich angestoßen und in Position 4 angezeigt. Mithilfe einer ALV-Tabelle wird dieser Vergleich visualisiert. Die Anzeige ist mit einer Trennerspalte versehen, um den Referenzbeleg (Position 1) und den Testbeleg (Position 2) klar abzutrennen.
An der Position 4 befinden sich grüne, rote und braune Zeilen beziehungsweise Zellen. Die grünen Zeilen signalisieren übereinstimmende Zeilen bei beiden Belegen. Die roten Zellen bedeuten, dass unterschiedliche Zeilen vorliegen und bei welchen Feldern die Unterschiede bestehen. Die braunen Zeilen sind nur bei einem der beiden Belege vorhanden und deswegen nur auf einer Seite farblich hervorgehoben. Bei der Anzeige werden nur die Spalten berücksichtigt, die im Customizing hinterlegt worden sind. Es können verschiedene Einträge des Customizings durch verschieden gespeicherte Customizingvarianten abgerufen werden. Die Varianten sind dabei Bezeichner für die Customizingzeile in der Datenbank.



Nettobtr.	Abrechnungsmenge	StGrBetrag	StGrMenge	BelZArt	Preisbetrag	Gültig ab	Gültig bis	---------	Nettobtr.	Abrechnungsmenge	StGrBetrag	StGrMenge	BelZArt	Preisbetrag	Gültig ab	Gültig bis
0,00	0,00000000000000			Z99999	0,00000000	26.03.2018	26.03.2018		0,00	0,00000000000000			Z99999	0,00000000	26.03.2018	26.03.2018
0,00	0,00000000000000			Z99999	0,00000000	26.03.2018	31.12.9999		0,00	0,00000000000000			Z99999	0,00000000	26.03.2018	31.12.9999
0,00	1,00000000000000	ZBGP	000000	000003	1,00000000	26.03.2018	26.03.2018		1,00	1,00000000000000	ZBGP	000000	000003	1,00000000	27.03.2018	26.03.2019
0,00	0,00000000000000			YQUHTG	0,00000000	26.03.2018	26.03.2018		0,00	0,00000000000000			Z99999	0,00000000	27.03.2018	31.12.9999
0,00	0,00000000000000	ZBAPHT	ZMAPHT	XQPHT	0,21000000	26.03.2018	26.03.2018		209,16	996,00000000000000	ZBAPHT	ZMAPHT	XQPHT	0,21000000	27.03.2018	26.03.2019
0,00	0,00000000000000			Z99999	0,00000000	26.03.2018	26.03.2018		0,00	996,00000000000000			Z99999	0,00000000	27.03.2018	26.03.2019
0,00	0,00000000000000	ZBAPNT	ZMAPNT	XQPNT	0,25000000	26.03.2018	26.03.2018		0,00	0,00000000000000	ZBAPNT	ZMAPNT	XQPNT	0,25000000	27.03.2018	26.03.2019
0,23	1,00000000000000	ZBGP	000000	XLPGP	82,82000000	26.03.2018	26.03.2018		82,82	1,00000000000000	ZBGP	000000	XLPGP	82,82000000	27.03.2018	26.03.2019
0,00	0,00000000000000	000005	000000	000003	0,00000000	26.03.2018	26.03.2018		0,00	0,00000000000000	000005	000000	000003	0,00000000	27.03.2018	26.03.2019
0,00	0,00000000000000				0,00000000				0,00	996,00000000000000			YQUHTG	0,00000000	27.03.2018	26.03.2019
0,00	0,00000000000000				0,00000000				0,00	0,00000000000000			Z99999	0,00000000	27.03.2018	26.03.2019
0,00	0,00000000000000				0,00000000				0,00	0,00000000000000			Z99999	0,00000000	27.03.2018	31.12.9999

Abbildung 19 Screenshot Abrechnungsbeleg-Analyser

4.2 Implementierungsbeschreibung

4.2.1 Klassendiagramm

Die Abbildung 20 beinhaltet das Klassendiagramm des Prototyps. Es wurde dabei nach dem MVC Modell vorgegangen, um eine Trennung zwischen Präsentation (View), Logik (Control) und Datenhaltung (Model) vorzunehmen.

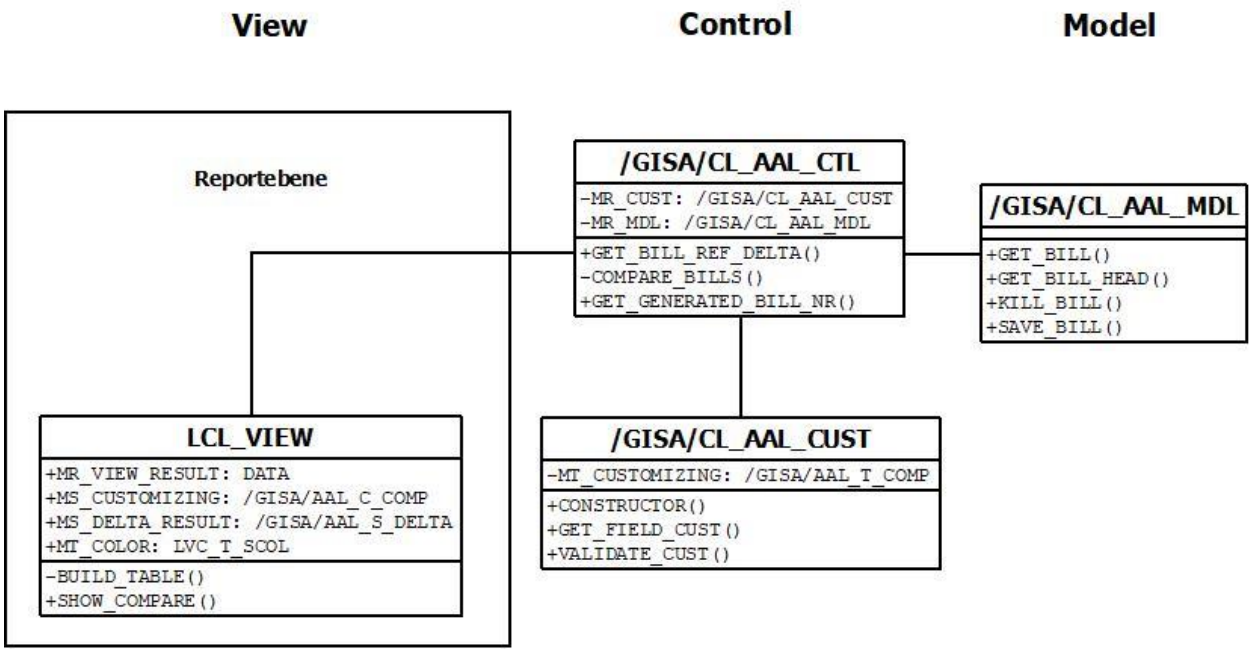


Abbildung 20 Klassendiagramm

Das Klassendiagramm wurde auf die relevanten Methoden reduziert. Damit ist eine bessere Übersicht über die Funktionen des Prototyps gegeben. Mit der View beginnt ein Beispielablauf des Vergleichs der Abrechnungsbelege. Dabei sollen die eingegebenen Daten an den Controller übergeben werden. Dieser lässt das Model die angefragten Daten von der Datenbank ermitteln. Im Weiteren stößt der Controller den Vergleich an und die Validierung des Customizingeintrags. Das Ergebnis des Vergleichs wird an die View übergeben und angezeigt. Dies ist der positive Fall des Vergleichs der Abrechnungsbelege, wenn keine Ausnahmen auftreten.

4.2.2 View

Die Klasse LCL_VIEW steuert innerhalb von MVC die Oberfläche und verarbeitet die Daten, die bereitgestellt werden müssen, um dem Nutzer ein korrektes Ergebnis präsentieren zu können. Darin werden die Benutzereingaben, das Customizing und das Ergebnis des Vergleichs beider Abrechnungsbelege, mithilfe von Memberattributen, gespeichert. Es gibt innerhalb der Klasse die Hauptmethode SHOW_COMPARE(), die für die Anzeige des Vergleichs der Abrechnungsbelege zuständig ist. Im Listing 3 ist zu sehen, dass die Memberattribute `ms_delta_result` und `ms_customizing` in den Zeilen 1 bis 7 zuerst gefüllt werden. Sie werden dann innerhalb von Zeile 8 `build_table( )` verwendet, die Farbinformation extrahiert und in eine interne Tabelle geschrieben. Diese wird schließlich in Zeile 9 visualisiert und dem Anwender präsentiert.

```
1  ms_delta_result = gr_ctl->get_bill_ref_delta(
2                    iv_bill_ref = gs_view-bill_ref
3                    iv_bill_comp = gs_view-bill_comp
4                    iv_variant = gs_view-cust_name ).
5
6  ms_customizing  = gr_ctl->get_customizing(
7                    iv_variant = gs_view-cust_name ).
8  build_table( ).
9  display_alv( ).
```

Listing 3 Ablauf des Vergleichs im View

In der Methode `build_table( )` werden die Spalten der internen Tabelle mittels RTTS und der Daten aus dem Customizing zusammengebaut.

4.2.3 Datenermittlung

Die Klasse /GISA/CL_AAL_CTL ist der Controller und steuert beziehungsweise initialisiert das Model, das Customizing und den Vergleich der Abrechnungsbelege. Als Memberattribute werden die Objekte des Customizings und des Models gehalten. GET_BILL_REF_DELTA() stößt dabei den Vergleich der Abrechnungsbelege an und vergleicht diese mithilfe von COMPARE_BILLS(). Dabei werden in Listing 4 in den Zeilen 1 und 2 die beiden Abrechnungsbelegsnummern aus dem View übergeben. In Zeile 3 wird der Bezeichner der Zeile im Customizing ebenfalls aus dem View übertragen.

In Zeile 4 wird der eingegebene Eintrag aus dem Customizing validiert.

```
1  DATA(lt_bill_1) = get_bill_from_db( iv_bill_number = iv_bill_ref ).
2  DATA(lt_bill_2) = get_bill_from_db( iv_bill_number = iv_bill_comp ).
3  DATA(ls_field_cust)=mr_cust->get_field_cust( iv_variant = iv_variant ).
4  mr_cust->validate_cust( is_field_cust = ls_field_cust ).
5  rs_delta_result = compare_bills(  it_bill_ref     = lt_bill_1
6                                    it_bill_comp    = lt_bill_2
7                                    is_field_cust   = ls_field_cust ).
```

Listing 4 Ablauf des Vergleichs im Controller

Im nächsten Schritt wird dann in Zeile 5 das Delta entgegengenommen und die Struktur aus Abbildung 21 erwartet.

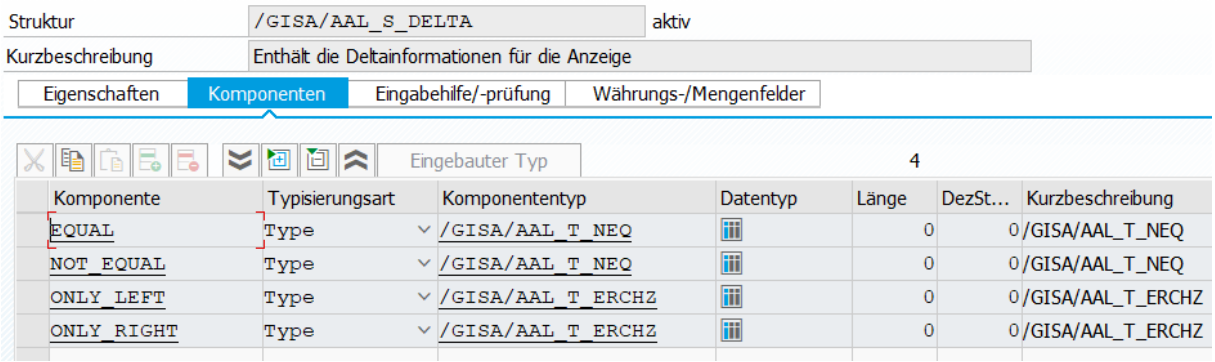


Abbildung 21 Struktur des Deltas

In Anlehnung an die Abbildung 21 werden im Vergleich 4 Fälle unterschieden:

- Zeilen sind gleich (EQUAL),
- Zeilen sind unterschiedlich (ähnliche Spalten sind mitenthalten, NOT_EQUAL),
- Zeile ist nur links vorhanden (im Referenzbeleg, ONLY_LEFT),
- Zeile ist nur rechts vorhanden (im Testbeleg, ONLY_RIGHT).

Die Zeilen, ob gleich oder unterschiedlich, sind in einer bestimmten Struktur, welche in Abbildung 22 dargestellt ist. In der linken Spalte Komponenten sind die Felder LEFT, RIGHT und FIELDS. Diese Komponenten der Struktur /GISA/AAL_S_NEQ werden benötigt, um gleiche und ähnliche Zeilen zu unterscheiden. In der Komponente LEFT werden die Zeilen gespeichert, die sich im Referenzbeleg befinden. In der Komponente RIGHT werden die Zeilen gespeichert, die sich im Testbeleg befinden.

In der Komponente FIELDS werden bei dem Fall, dass die Zeilen sich nur ähneln, die unterschiedlichen Felder gespeichert.

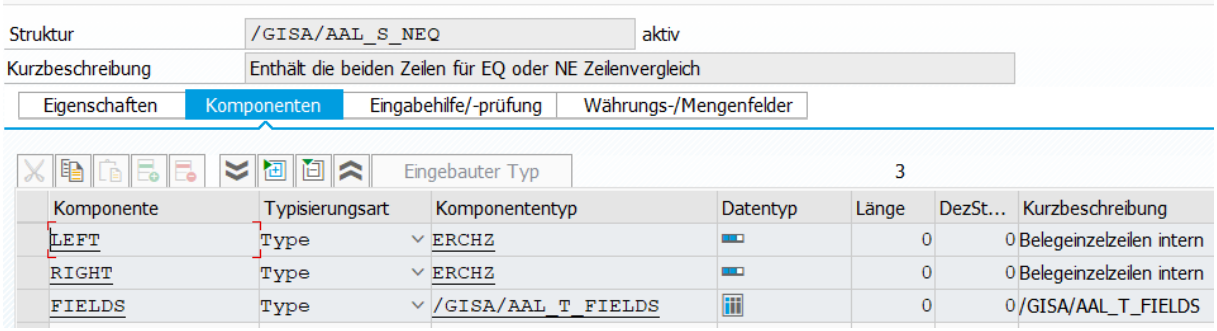


Abbildung 22 Struktur für Zeilenvergleich

EQUAL ist das Synonym, welches anzeigt, dass beide Zeilen gleich sind, NOT_EQUAL gibt an, dass beide Zeilen ähnlich sind, ONLY_LEFT steht dafür, dass die Zeilen ausschließlich im Referenzbeleg und ONLY_RIGHT gibt wiederum an, dass die Zeilen ausschließlich im Testbeleg enthalten sind.

Die Methode COMPARE_BILLS() fängt an mit einem dynamischen Aufbau einer Struktur inklusive einer internen Tabelle. Mithilfe von RTTS und dem Customizing, was die vergleichenden Spalten enthält, wird eine Basis für nachfolgende Schritte erstellt. Das Schreiben von Daten der korrespondierenden Spalten von beiden internen Tabellen (Abrechnungsbeleg und neue interne Tabelle) ist nötig, um auf die relevanten Spalten den Abrechnungsbeleg zu kürzen. Danach erfolgt der Vergleich von beiden Abrechnungsbelegen, welche in den neuen internen Tabellen geschrieben wurden. Dabei finden die vier unterschiedlichen Fälle Beachtung EQUAL, NOT_EQUAL, ONLY_LEFT und ONLY_RIGHT. Der aufwändigste Fall ist dabei NOT_EQUAL.

4.2.4 Algorithmus für Ähnlichkeitsprüfung

Das Listing 5 zeigt, wie zwei Zeilen miteinander verglichen werden. Bei einer gefundenen Übereinstimmung (Zeile 4) wird diese Zeile in die Zielstruktur hineinkopiert und die beiden Zeilen aus den internen Tabellen gelöscht (Zeile 6 und Zeile 15). In den Zeilen 12 und 13 wird dabei der Befehl MOVE-CORRESPONDING verwendet, welcher die Zeilen aus einer internen Tabelle in eine andere interne Tabelle kopiert.

Passende Spalten werden dabei automatisch erkannt und damit ein Kopiervorgang ermöglicht, ohne direkt auf die Spalten zugreifen zu müssen.

```
1   LOOP AT <lt_ref_bill> ASSIGNING FIELD-SYMBOL(<ls_ref_bill>).
2     CLEAR: lv_found.
3     LOOP AT <lt_comp_bill> ASSIGNING FIELD-SYMBOL(<ls_comp_bill>).
4       IF <ls_ref_bill> EQ <ls_comp_bill>.
5       lv_found = 'X'.
6       DELETE <lt_comp_bill>.
7       EXIT.
8       ENDIF.
9     ENDLOOP.
10    IF lv_found EQ 'X'.
11    CLEAR: ls_equal, lv_found.
12    MOVE-CORRESPONDING <ls_ref_bill> TO ls_equal-left.
13    MOVE-CORRESPONDING <ls_ref_bill> TO ls_equal-right.
14    APPEND ls_equal TO rs_delta_result-equal.
15    DELETE <lt_ref_bill>.
16    ENDIF.
17  ENDLOOP.
```

Listing 5 Vergleich von zwei Zeilen

Nachdem die gleichen Zeilen herausgenommen wurden, verbleiben in den beiden neuen internen Tabellen noch Zeilen, die auf Ähnlichkeit untersucht werden müssen und in NOT_EQUAL gespeichert werden. Dieser Schritt erfolgt sobald zwei ähnliche Zeilen gefunden wurden. Der Algorithmus für Ähnlichkeitsprüfung funktioniert folgendermaßen:

1. Die Zeilen vom Testbeleg werden in eine interne Tabelle `lt_similar` auf Basis einer lokalen Struktur gespeichert. Die Struktur hat als Komponenten die Datenbankzeile eines Abrechnungsbelegs, den Ähnlichkeitszähler `similar_cnt` und die Felder welche unterschiedlich sind.

2. Danach wird über die Zeilen vom Referenzbeleg eine Schleife durchgeführt und als Abbruchbedingung die Prüfung, ob `lt_similar` leer ist, festgelegt.

3. In dieser Schleife wird eine weitere Schleife mit der gespeicherten Anzahl von Spalten durchgeführt. Es werden hier die Spalten mithilfe einer Index-Variablen (welche inkrementiert wird) durchlaufen.

4. Innerhalb der zweiten Schleife wird der Testbeleg mithilfe einer dritten Schleife durchlaufen. Hier wird in den Zeilen beider Testbelege die referenzierte Spalte verglichen, welche in der zweiten Schleife aktuell ist. Wenn der Inhalt der Zelle in den Spalten gleich ist, wird in `lt_similar` der `similar_cnt` (Ähnlichkeitszähler) für diese Zeile erhöht.

5. Sind die Zellen nicht gleich, wird der Name der Spalte in der `lt_similar` hinterlegt.

6. Wenn die zweite und dritte Schleife fertig ist, wird in `lt_similar` das erste Ergebnis mit der höchsten Übereinstimmung genommen und in der internen Tabelle als Zeile der Komponente NOT_EQUAL gespeichert.

7. Zuletzt werden bei `lt_similar` und den beiden internen Tabellen der Abrechnungsbelege die betreffenden Zeilen gelöscht. Es wird bis zum Abbruch wieder bei zweitens angefangen.

Im Listing 6 werden die nachfolgenden beiden letzten Fälle abgearbeitet.

```
1   LOOP AT <lt_ref_bill> ASSIGNING FIELD-SYMBOL(<ls_ref_bill_03>).
2     CLEAR: ls_only_left.
3     MOVE-CORRESPONDING <ls_ref_bill_03> TO ls_only_left.
4     APPEND ls_only_left TO rs_delta_result-only_left.
5     DELETE <lt_ref_bill>.
6   ENDLOOP.
7   LOOP AT <lt_comp_bill> ASSIGNING FIELD-SYMBOL(<ls_comp_bill_03>).
8     CLEAR: ls_only_right.
9     MOVE-CORRESPONDING <ls_comp_bill_03> TO ls_only_right.
10    APPEND ls_only_right TO rs_delta_result-only_right.
11    DELETE <lt_comp_bill>.
12  ENDLOOP.
```

Listing 6 Letzte Fälle

In Zeile 1 bis 6 werden dabei die Zeilen, die nur im Referenzbeleg vorhanden sind, verarbeitet. Es wird mit dem Befehl `MOVE-CORRESPONDING` (Zeile 3) der Inhalt der Zeile in die erzeugte

Struktur übertragen und an die Zielstruktur (Zeile 4) angehängt. Danach wird aus der internen Tabelle die Zeile gelöscht (Zeile 5). Analog dazu funktioniert Zeile 7 bis 12 mit dem Testbeleg.

Da gleiche und ähnliche Zeilen nun aussortiert sind, können die jeweils vorhandenen Abrechnungsbelegszeilen kopiert werden. Also die Zeilen, die nur jeweils in dem Referenzbeleg und in dem Testbeleg vorhanden sind. Damit ist der Vergleich abgeschlossen und die Ergebnisstruktur `rs_delta_result` gefüllt. Die Information über die gleichen, ähnlichen und jeweiligen Zeilen wird an die View übergeben und ausgewertet.

4.2.5 Simulation

In der Methode GET_GENERATED_BILL_NR wird eine Abrechnungsbelegsnummer erwartet und aus dieser die Vertragsnummer sowie der Abrechnungszeitraum aus den Kopfdaten der Abrechnungsbelegstabelle entnommen. Somit wird eine laufende Nummer generiert und mit den folgenden Daten ein simulierter Beleg erzeugt. Die Vertragsnummer, der Beginn des Abrechnungszeitraums, das Ende des Abrechnungszeitraums und eine laufende Abrechnungsnummer sind wichtig, um einen Beleg simulieren zu können.

Im Listing 7 ist zu sehen, dass in Zeile 3 die Kopfdaten ausgelesen und in eine lokale Struktur übergeben werden. In dem Funktionsbaustein Zeile 9 wird eine laufende Abrechnungsnummer erzeugt.

```
1  DATA: ls_bill_head      TYPE erch,
2        lv_billingrunno   TYPE ERCH-BILLINGRUNNO.
3        ls_bill_head = mr_mdl->get_bill_head( iv_bill_nr = iv_bill_ref ).
4  DATA: co_number_object TYPE inri-object VALUE 'ISU_BIRUN'.
5  CALL FUNCTION 'ISU_NUMBER_GET'
6    EXPORTING
7      object                    = co_number_object
8    IMPORTING
9      number                    = lv_billingrunno
```

Listing 7 Kopfdaten

Danach wird der Funktionsbaustein ISU_SIMULATION_PERIOD_BILL angestoßen und die vorangegangenen Ergebnisse übergeben. Konkret wird die Vertragsnummer, der Beginn der Abrechnungsperiode, das Ende der Abrechnungsperiode und die laufende Abrechnungsnummer übergeben.

Als vorletzter Schritt wird die Abrechnungsbelegsnummer in die RETURNING Variable kopiert. Nun müssen noch die Änderungen in die Datenbank geschrieben werden. Dazu wird ein Commit gesetzt. Die Änderungen sind nun wirksam und es kann beispielsweise der Vergleich des Referenzbelegs mit dem simulierten Beleg erfolgen.

4.2.6 Validierung und Customizing

/GISA/CL_AAL_CUST enthält das Customizing, welches in dieser Klasse von der Datenbank gelesen wird.

Die Validierung der angeforderten Zeile des Customizings wird auf Basis des Bezeichners vorgenommen, welcher vom Benutzer eingegeben wurde. Es wird geprüft ob sich keine Lücke zwischen den Eingaben befindet. Es wird vorausgesetzt, dass keine Spalte ausgelassen wurde im Customizing und danach weiter ausgefüllt.

In Listing 8 wird in der DO Schleife ab Zeile 1 das Customizing überprüft und bei einem Fehler eine Ausnahme ausgelöst.

```
1  DO.
2    lv_counter = lv_counter + 1.
3    DATA(lv_field_number) = |FIELD_{ lv_counter }|.
4    ASSIGN COMPONENT lv_field_number OF STRUCTURE is_field_cust
5    TO FIELD-SYMBOL(<ls_field>).
6    IF sy-subrc EQ 0 AND <ls_field> IS NOT INITIAL.
7    lv_compare = lv_counter - lv_found.
8    IF lv_compare GT 1.
9      lv_counter = lv_counter - 1.
10     RAISE EXCEPTION TYPE /gisa/cx_aal
11     MESSAGE ID '/GISA/AAL_CAST' NUMBER '005' WITH lv_counter space
12     space space.
13   ENDIF.
14   lv_found = lv_counter.
15   ELSE.
16     IF lv_counter GT 10.
17       EXIT.
18     ENDIF.
19   ENDIF.
20 ENDDO.
```

Listing 8 Validierung des Customizings

Dies erfolgt mit dazugehöriger Fehlermeldung. Die Ausnahme wird in Zeile 10 geworfen. Eine dazugehörige Nachricht befindet sich in Zeile 11. Hinzu kommt, dass die Variable

`lv_counter` mitgegeben wird, um die Position der fehlenden Spalte in der Zeile weiterzugeben. Die Validierung erfolgt zunächst nur, unter dem Aspekt, dass die Felder gefüllt sind und es keine Lücken in der Zeile gibt.

4.2.7 Model

Die Klasse /GISA/CL_AAL_MDL ist dafür zuständig, die Daten der Abrechnungsbelege von der Datenbank zu lesen und zu löschen. SQL Statements werden hierzu direkt in den Quellcode eingegeben oder die Daten per Funktionsbaustein geholt. Um aus den Tabellen die Belegzeilen der eingegebenen Abrechnungsbelegsnummer zu bekommen, wird der Funktionsbaustein ISU_DB_ERCHZ_SELECT_BILL verwendet und in einer internen Tabelle gespeichert.

4.3 Probleme und Lösungen während der Umsetzung

Das MVC Modell ist in ABAP nach folgendem Schema umgesetzt. (Siehe hierzu Abbildung 23).

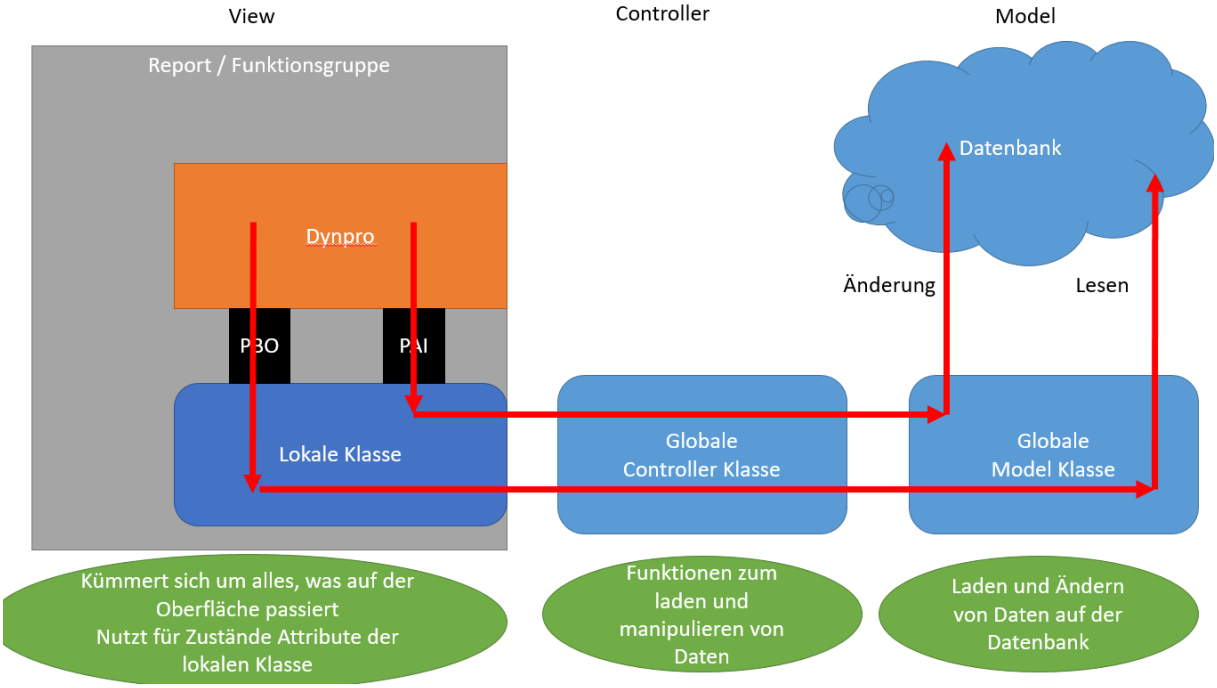


Abbildung 23 MVC in ABAP

Da der Report mit den Dynpro Elementen zuerst über eine lokale Klasse gekapselt und die Ereignisse im Dynpro (PBO, PAI) ebenfalls gekapselt werden müssen, gibt es hier einen Umweg, um MVC konform zu handeln.

In der SAP finden sich teilweise festgelegte Begrenzungen bestimmter Systemvariablen. Diese sind nötig, um zum Beispiel eine Nachricht in einer Ausnahme zu behandeln. Die sogenannten Message-Variablen sind auf 50 Zeichen begrenzt. Eine mögliche Lösung war es, die Nachricht auf die 4 vom System bereitgestellten Platzhalter zu verteilen und diese als Variable mitzugeben.

4.4 Bewertung der Ergebnisse

In einem GISA GmbH internen Termin wurden die Ergebnisse des Prototyps gezeigt und von zwei Abrechnungsexperten bewertet. Die Bewertung wurde zusammengefasst und entspricht nicht dem genauen Wortlaut der Abrechnungsexperten.

Anforderung 1: Vergleich zweier Abrechnungsbelege

Der Vergleich zweier Abrechnungsbelege mithilfe des entwickelten Prototyps erfüllt die Anforderungen. Durch die Zeilen in der ALV Tabelle wird schnell eine Übersicht über Unterschiede und Gemeinsamkeiten der Abrechnungsbelege gegeben, die manuell nur mit höherem Aufwand ermittelt werden könnten. Durch das vom Anwender eingestellte Customizing werden nur relevante Felder gezeigt, wobei eine schlankere Oberfläche entsteht als bei einem direkten Vergleich der Abrechnungsbelege auf Basis der Tabelleneinträge.

Anforderung 2: Verwalten von Referenzabrechnungsbelegen

Das Speichern, Löschen und Auswählen von Referenzabrechnungsbelegen wird mit dem Customizing und der Feldwertehilfe für die zuvor ausgesuchten Abrechnungsbelege adäquat umgesetzt. Da diese nicht benutzerspezifisch, sondern mandantenspezifisch gespeichert werden, gibt es eine saubere Trennung von Daten in dem Entwicklungsmandanten und Testmandanten.

Anforderung 3: Erzeugen eines simulierten Abrechnungsbelegs (Basis Referenzbeleg)

Der Referenzabrechnungsbeleg bildet die Basis des simulierten Abrechnungsbelegs. Die Nummer dessen wird an das Programm übergeben und daraus wird ein simulierter Abrechnungsbeleg wie gewünscht erzeugt. Nach der Generierung kann dieser mit dem Referenzbeleg verglichen werden. Mögliche Unterschiede, zwischen dem Referenzbeleg aus der Datenbank und dem simulierten Beleg nach der Ablauflogik der Abrechnung, werden hierbei aufgezeigt.

Zusatz: Absprung in die Abrechnungsbelegsübersicht

Als zusätzliches Feature wurde der Abrechnungsbeleg-Analyser, um die Funktionalität des Wechsels in die Übersicht der Abrechnungsbelegszeilen erweitert. Mittels Doppelklick auf ein schon gefülltes Feld der Abrechnungsbelege wird der Absprung ermöglicht. Dieses wurde ebenfalls als adäquat bewertet.

5 Fazit und Ausblick

Ziel dieser Arbeit war es, ein prototypisches Analysewerkzeug für Abrechnungsbelege vor und nach Änderungen zu entwickeln. Dieses besitzt drei Kernfunktionalitäten. Diese wurden erfolgreich gemeinsam umgesetzt und abgenommen. Als Zusatz ist noch der Absprung in die Abrechnungsbelegszeilenübersicht per Doppelklick implementiert. Der Bereich der Regressionstests und des Testens allgemein wurde nur angeschnitten.

Die Ergebnisse wurden von zwei Abrechnungsexperten der GISA GmbH positiv bewertet, da der Abrechnungsbeleg-Analyser den personellen Aufwand kostensenkend reduzieren kann. Das Werkzeug wurde einem Großkunden im Energiemarkt erfolgreich präsentiert und der auch beabsichtigt es zu kaufen, damit dieser seinen Arbeitsaufwand reduzieren und zukünftig leichter und verständlicher mit der komplexen Thematik der IS-U-Abrechnung umgehen kann.

Da nur die Kernfunktionalitäten und eine Zusatzfunktion eingebaut werden konnten, sind noch Wünsche offen geblieben, die von den zwei Abrechnungsexperten der GISA GmbH angemerkt wurden. Um diese zusammenzufassen beginnt dieser Abschnitt mit den wichtigsten Features und hört bei eher optionalen Erweiterungen auf.

Die Simulation benutzt einen SAP Standard Funktionsbaustein. Dieser konnte aber im Rahmen eines Tests keinen Abrechnungsbeleg simulieren, wenn der Abrechnungsbeleg selbst fehlerhaft war. Hier könnte noch eine genauere Weitergabe der Fehlerinformation über den Abrechnungsbeleg-Analyser erfolgen. Die Funktionalität bei korrekten Abrechnungsbelegen ist gegeben.

Zudem wurde angemerkt, dass es Massentests geben könnte. Mit dem Ausführen von Massentests auf Basis einer Liste von Referenzbelegen oder einer Kategorie von Referenzbelegen könnten schnell viele Abrechnungsbelege verglichen werden. Der Vergleich soll dabei selbst anpassbar sein in der Spaltenzahl die verglichen werden können und welche Spalten angezeigt werden sollen in der Liste.

Dazu gehört eine Übersichtsdarstellung mit Ampel. Das heißt der Massentest wird mit einer Tabelle visualisiert und in dieser Tabelle gibt es pro Zeile eine Spalte mit Ampelanzeige. Diese wird in Abb. 24 aufgezeigt.

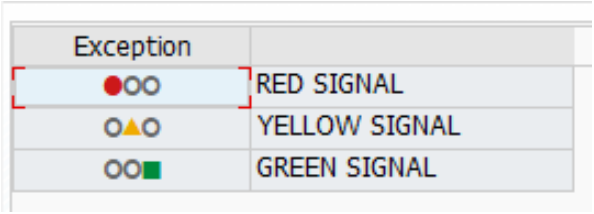


Abbildung 24 SAP Ampel

Es soll außerdem Vergleiche mit Priorität geben. Das heißt, dass einzelne Spalten eine höhere Gewichtung beim Vergleich haben sollen als andere. Als größerer Punkt einer Erweiterung würde es in Richtung RLM[15], also Großkundenanpassungen gehen. Außerdem wurden noch Punkte zur Usability angemerkt, wie beispielsweise Feldwertehilfe im Customizing und anpassbare Farben der Zeilen des Vergleichs.

[15] Die registrierende Leistungsmessung erfolgt ab einem Jahresverbrauch von ca. 100.000 kWh.

Literaturverzeichnis

Frederick, Jörg und Tobias Zierau (2011): SAP for Utilities. Das umfassende Handbuch für Energieversorger, 1. Aufl., Bonn: Galileo Press

Mindsquare AG, SAP IS-U, in: Mindsquare, [online] https://www.mindsquare.de/knowhow/sap-is-u/ 19.12.19

Wolff, Dipl.-Geogr. Ingo, Nutzung der SAP IS-U Verbrauchsdaten im Smallworld GIS der Stadtwerke Iserlohn GmbH, in: Its-service, [online] https://its-service.de/assets/img/projekte /stadtwerke-iserlohn-2011/screen-5_full.png 27.03.2020

Possel, Dr. Heiko, SAP – Übersicht, in: Stichpunkt, [online] https://www.stichpunkt.de/sap /module.html 23.12.2019

SAP AG, Abrechnungsbeleg, in: Help.SAP, [online] https://help.sap.com/viewer/efcbb384e71c48b99 370ea411db1cf58/6.00.31/de-DE/20f5c5536a51204be10000000a174cb4.html 05.03.2020

Testbirds GmbH, Regressionstest, in: Testbirds, [online] https://www.testbirds.de/leistungen/quality-assurance/regressionstest/ 05.03.2020

Allen, Barry, Structure in SAP ABAP, in: Gocoding [online] https://www.gocoding.org/ structure-in-sap-abap 15.04.2020

SAP AG, Run Time Type Services, in: Help.SAP [online] https://help.sap.com/doc /abapdocu_752_index_htm/7.52/de-DE/abenrun_time_type_services_glosry.htm 07.01.2020

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich vorliegende Masterarbeit selbstständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der von mir angegebenen Literatur und Hilfsmittel angefertigt habe.

Merseburg, den 22.05.2020 Sascha Fehst

Einverständniserklärung

Ich erkläre mein Einverständnis zu einer Veröffentlichung der vorliegenden Arbeit im Internet.

Ja Nein

Merseburg, den 22.05.2020 Sascha Fehst