VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO LADĚNÍ DEFINICE PLATEBNÍHO FORMÁTU

TOOL FOR PAYMENT FORMAT DEFINITION DEBUGGING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **RICHARD KUBA**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. JAROSLAV ZENDULKA, CSc.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Kuba Richard**
Program: Informační technologie
Název: **Nástroj pro ladění definice platebního formátu**
Tool for Payment Format Definition Debugging
Kategorie: Informační systémy

Zadání:

1. Seznamte se s architekturou platformy S/4HANA.
2. Seznamte se s problematikou definice platebního formátu v prostředí produktů společnosti SAP.
3. Seznamte se se současným řešením pro správu platebních formátů.
4. Seznamte se s požadavky kladenými na nástroj pro ladění definice platebního formátu.
5. Navrhněte koncepci řešení nástroje.
6. Proveďte detailní návrh, implementujte ho a ověřte na vhodných datech.
7. Zhodnoťte dosažené výsledky a diskutujte možná vylepšení.

Literatura:

- Silvia, P., Frye, R., Berg, B.: SAP HANA. SAP PRESS. 2017.
- Message Reference Guide, Category 1 - Customer Payments and Cheques. Online - https://www2.swift.com/knowledgecentre/rest/v1/publications/us1m_20170720/1.0/us1m.pdf
- Gahm, H. et al.: ABAP Development for SAP HANA. SAP PRESS, 2014.
- Bardhan D. et al.: SAP S/4HANA: An Introduction. RHEINWERK PUBLISHING, 2019.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zendulka Jaroslav, doc. Ing., CSc.**
Konzultant: Brázdil Jan, Mgr., SAP
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 16. října 2019

Abstrakt

Hlavním cílem této práce je vyvinout a demonstrovat nástroj pro ladění platebních formátů, který by uživatelům transakce (programu) DMEEX usnadňoval odhalování chyb v definičních stromech. Demonstrace nástroje pro ladění je implementována na platformě SAP S/4HANA. V první části práce jsou popisovány platformy SAP R/3 a SAP S/4HANA, s důrazem na vystižení rozdílů mezi nimi. Dále je pak rozebrána problematika platebních formátů a jejich podpora v rámci systémů SAP. V rámci návrhu je popsán sběr a práce s požadavky uživatelů na tento nástroj. Implementovaný produkt umožňuje uživatelům vizualizovat průběh zpracování definičního stromu transakce DMEEX, díky čemuž jeho uživatelům umožňuje jednodušší odhalování chyb v definici stromu nebo v jeho vstupních datech.

Abstract

The main goal of this thesis is to develop and demonstrate a tool for debugging payment formats that would make it easier for users of the DMEEX transaction (program) to detect bugs in definition trees. Demonstration of the debugging tool is implemented on the SAP S / 4HANA platform. The first part of this thesis describes the platforms SAP R / 3 and SAP S / 4HANA, with emphasis on capturing the differences between them. Furthermore, the purpose of payment formats and their integration within SAP systems is discussed. The design describes the collection and work with user requirements for this tool. The implemented product allows users to visualize the processing of the DMEEX transaction definition tree, thanks to which it allows its users to more easily detect errors in the definition of the tree or in its input data.

Klíčová slova

platební formáty, SAP, DMEE, DMEEX, ladění, ABAP, SAP GUI, SAP S/4HANA

Keywords

payment formats, SAP, DMEE, DMEEX, debugging, ABAP, SAP GUI, SAP S/4HANA

Citace

KUBA, Richard. *Nástroj pro ladění definice platebního formátu.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Jaroslav Zendulka, CSc.

Nástroj pro ladění definice platebního formátu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Jaroslava Zendulky CSc. Další informace mi poskytl Mgr. Jan Brázdil. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Richard Kuba
28. května 2020

Poděkování

Velice rád bych zde poděkoval panu doc. Ing. Jaroslavu Zendulkovi CSc. za veškeré konzultace, rady a doporučení, které mi v průběhu vytváření této práce dával. Rád bych vyzdvihl klidný a laskavý přístup pana docenta a jeho velkou trpělivost. Dále bych rád poděkoval svoji rodině za trvalou podporu při studiu a své přítelkyni za podporu při dokončování práce.

Obsah

Seznam obrázků

Seznam zkratek

- SAP - Systeme, Anwendungen, Produkte in der Datenverarbeitung - softwarová společnost se sídle v německém Walldorfu známá především jako dodavatel ERP systémů.
- ERP - Enterpise Resource Planning, informační systém pro plánování podnikových zdrojů
- DMEE - Data Medium Exchange Engine, transakce systému SAP sloužící pro definici formátu výstupního souboru.
- DMEEX - Data Medium Exchange Engine eXtended, rozšířená verze DMEE pro platformu SAP S/4HANA.
- PMW - Payment Medium Workbench, soubor nástrojů systémů SAP sloužící ke správě platebních formátů.
- ABAP - Advanced Business Application Programming, programovací jazyk nativní pro aplikační server systému SAP.
- AS ABAP - aplikační server ABAP, aplikační vrstva systémů SAP
- PAI - process after input, událost logiky toku obrazovky Dynpro vykonávaná po zadání vstupu
- PBO - process before output, událost logiky toku obrazovky Dynpro vykonávaná před zobrazením výstupu
- Dynpro - Dynamic Program, slouží k definici obrazovek pro SAP GUI

Kapitola 1

Úvod

Bakalářská práce se zabývá platebními formáty a jejich použitím v rámci systému SAP. Platební formáty jsou média, která se používají pro komunikaci mezi firmou (klientem) a bankou. Neexistuje jednotná autorita, která by sjednocovala platební formáty napříč jednotlivými státy, proto je nutné formáty lokalizovat. V prostředí SAP se k lokalizaci platebních formátů používají transakce (programy) DMEE a DMEEX. Pro definici formátu v DMEE/DMEEX se používá tzv. definiční strom. Jde o relativně expertní transakce a odhalit některé chyby v definičním stromu není snadně.

Cílem této bakalářské práce je navrhnout nástroj, který by umožnil uživateli jednoduše tyto chyby odhalit, bez zdlouhavého procházení cizích zdrojových kódů. Nástroj pro ladění by měl v první fázi být demonstrován na platformě SAP S/4HANA, ale již vytvářen s myšlenkou možného přenosu na starší verze systému SAP.

Bakalářská práce je členěna do celkem 5 kapitol. Po této úvodní kapitole následuje popis platformy SAP R/3, která je stále nejrozšířenějším ERP softwarem společnosti SAP. Zaměřuje se na charakteristiku každé z vrstev třívrstvé architektury s důrazem na popis komponent, které sehrávají důležitou roli v implementaci výsledného produktu. Nástupnický systém S/4HANA zasazuje do kontextu jeho předchůdce a vystihuje zásadní rozdíly mezi těmito platformami, především pak in-memory databázi SAP HANA a s ní spojené technologie.

Následuje popis problematiky platebních formátů. Zde se detailně se představují transakce DMEE a DMEEX, sloužící k vytváření výstupních souborů na základě definičního stromu. Transakce jsou popisovány primárně v rámci integrace podpory pro platební formáty v systémech společnosti SAP. Dále na praktických příkladech demonstruje použití transakcí DMEE a DMEEX pro vytváření výstupních platebních souboru včetně jednoho skutečného platebního formátu.

Ve 4. kapitole se práce zaměřuje na návrh a implementaci nástroje pro ladění. V úvodu kapitoly jsou představena současná řešení, které lze využít k ladění a důvody proč nejsou zcela vhodná. Náslehuje popis procesu získávání požadavků od cílových skupin transakcí DMEE a DMEEX a finalizace návrhu nástroje pro ladění na základě těchto požadavků. v části implementace jsou představeny použité technologie a jednotlivé součásti tvořící nástroj pro ladění. V závěru kapitoly je rozbírán proces testování.

V závěru dochází k hodnocení dosažených výsledků a možných vylepšení.

Kapitola 2

ERP systémy společnosti SAP

V této kapitole se bakalářská práce zabývá pojmem podnikový informační systém. Popisuje architekturu systémů SAP R/3 a SAP S/4HANA, kde se podrobněji zabývá technologiemi, které jsou relevantní pro tuto práci.

2.1 Podnikový informační systém

Podnikový informační systém, anglicky označovaný jako ERP (enterprise resource planning) je software používaný ve firmách pro integraci a řízení všech oblastí, které souvisí s provozem firmy. Jedná se typicky o oblasti jako finanční účetnictví, nákladové účetnictví, prodej a marketing, plánování výroby, materiálové hospodářství nebo řízení lidských zdrojů.

Společným rysem ERP systémů je společné využití dat. Všechny aplikace ERP systému tedy pracují nad jednou databází, což je základní předpoklad pro integraci všech komponent systému. Např. při plánování výroby využívá ERP systém data z materiálového hospodářství, vyplacení mezd v rámci řízení lidských zdrojů je nutné zohlednit v účetnictví apod.

Nasazení ERP systému do provozu ve společnosti vyžaduje obvykle důkladnou analýzu všech procesů v dané firmě a přizpůsobení informačního systému konkrétním potřebám. Takový systém proto musí být vysoce přizpůsobitelný.

ERP systém operuje v reálném čase, databáze podporuje všechny komponenty a aplikace sytému, moduly ERP systému vypadají a chovají se stejně. Mezi základní možnosti nasazení ERP systému patří:

- On-premise, kde je systém nasazen na vlastním harwaru firmy provozující ERP řešení. Poskytovatel ERP dodává licenci popř. může dodávat i podporu svého softwaru. On-premise řešení jsou obvykle velmi flexibilní a umožňují provozovateli dodatečné modifikace.

- Cloud nasazení, kde obvykle poskytovatel dodává jak software, tak hardware. Software bývá často licencován formou předplatného. V takovém případě se bavíme o SaaS (Software as a Service, česky software jako služba).

2.2 Platforma SAP R/3

Společnost SAP byla založena v roce 1972 s cílem vyvinout standardní software pro řízení podnikové ekonomiky. První platformou společnosti SAP se později stal systém SAP R/1

(R vychází z „Real Time-Datenverarbeitung“ – zpracování dat v reálném čase), který podporoval primárně oblast finančního účetnictví. Architektura systému R/1 byla jednovrstvá. Prezentační, aplikační i databázová vrstva se nacházela na jednom sálovém počítači.[6]

Prvním systémem, který lze na základě definice z předchozí části označit jako ERP, se stal následovník SAP R/2, který byl představen v roce 1979.[4] Za ERP systém jej považujeme, protože se oproti jeho předchůdci výrazně rozšířil počet dostupných modulů pro firemní procesy. Mimo finančního účetnictví podporoval dále: controlling, materiálové hospodářství, řízení zásob, plánování produkce, řízení kvality, údržbu zařízení, podporu pro prodej, distribuci a logistiku nebo modul pro personální management.[3] Z pohledu architektonického se jednalo o dvouvrstvý systém, kde je aplikační a databázová vrstva na jednom sálovém počítači. Prezentační vrstva je instalována na klientských stanicích.[7]

Platforma SAP R/3 byla představená v roce 1992. Mimo rozšíření nabídky komponent o další oblasti pokrývající rozlišné oblasti firemních procesů, implementovala třívrstvou architekturu. Později byla tato platforma přejmenována na SAP ECC (SAP ERP Central Component). Práce se v následující části bude zabývat každou z těchto vrstev samostatně. Bude se zaměřovat na popis klíčových součástí každé z vrstev s důrazem na oblasti související s touto prací.

2.2.1 Prezentační vrstva

Úlohou prezentační vrstvy je předkládat uživateli data v grafické reprezentaci. Realizuje vstupně/výstupní funkce systému SAP. Tyto služby jsou zprostředkovány uživateli pomocí aplikace SAP GUI. SAP GUI je tenkým klientem, sama o sobě obsahuje minimum logiky, jejím účelem je pouze prezentace a zadávání dat aplikační vrstvy.[6]

SAP GUI

SAP GUI implementuje podporu pro zobrazovaní kompletního uživatelského rozhraní, tedy tlačítka, přepínače, menu, dialogová okna, zaškrtávací políčka apod. Kompletní definice uživatelského rozhraní je však načítána z aplikační vrstvy, kde se definuje jako tzv. Dynpro (dynamic program). Díky oddělení prezentační a aplikační vrstvy je snazší zajistit nezávislost platforem. Samotné SAP GUI podporuje operační systémy typu Windows, macOS, Linux i Unix. Zároveň existují další verze SAP GUI např. pro použití na webových stránkách SAP WebGUI.

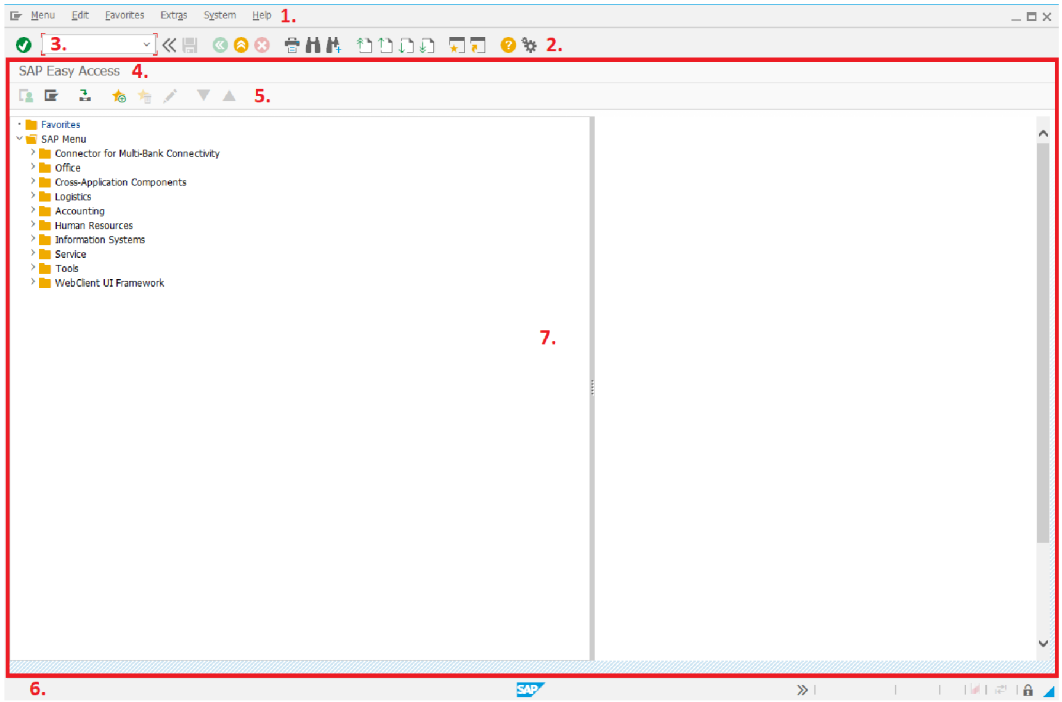
SAP GUI komunikuje se součástí aplikační vrstvy ABAP Dispatcher datovými balíky o velikosti 2,6-5,3kB na transakci v rámci transakčního zpracování.[6]

2.2.2 Aplikační vrstva

Aplikační vrstva systému SAP R/3, jmenovitě ABAP Application Server (dále AS ABAP), poskytuje běhové prostření pro programovací jazyk ABAP. Jádro aplikační vrstvy je napsáno v jazycích C/C++ a slouží jako základ pro aplikační programy napsané v jazyce ABAP. Dále poskytuje základní služby jako komunikaci s databázovým serverem, podporu XSL transformací, generování XML souborů z dat jazyka ABAP apod. Z výše zmíněného vyplývá, že jazyk ABAP je nezávislý na platformě. AS ABAP je podporován všemi zavedenými operačními systémy.[11]

Mezi hlavní úkoly jádra AS ABAP patří:

- Spouštění aplikací v jazyce ABAP na softwarových procesorech (virtuálních strojích)

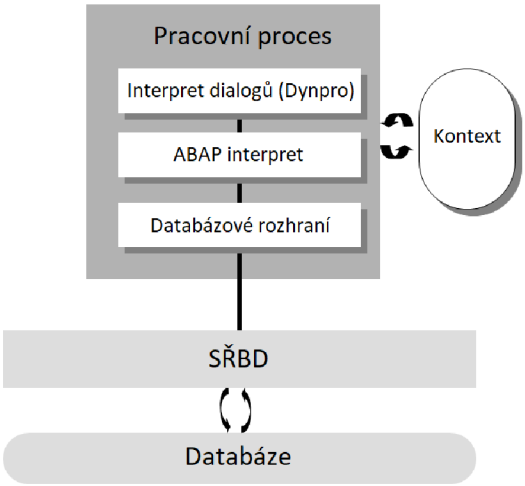


Obrázek 2.1: 1) Lišta menu 2) Standardní panel nástrojů 3) Příkazový řádek 4) Záhlaví, obvykle název programu 5) Aplikačně specifický panel nástrojů 6) Stavová lišta 7) Aplikačně specifický prostor

- Správa uživatelů a procesů. AS ABAP podporuje víceuživatelské prostředí a každý uživatel může spouštět určitý počet programů v jazyce ABAP. Správa uživatelů a procesů odstiňuje uživatele využívajícího služby AS ABAP od operačního systému operujícího pod AS ABAP.
- Každý AS ABAP je připojen k databázovému systému, který se skládá ze systému řízení báze dat (SŘBD) a samotné databáze. Programy v jazyce ABAP však nekomunikují přímo s databázovým serverem, ale přes prostředníka – administrativní služby jádra.[11]

Pracovní procesy (Work Processes)

Jak bylo zmíněno výše, správa procesů je jednou z klíčových funkcionalit AS ABAP. Pracovní procesy jsou zodpovědné za provádění jednotlivých dialogových kroků aplikačních programů v jazyce ABAP. Součástí každého pracovního procesu je interpret dialogů, který zodpovídá za zpracování obrazovek Dynpro, procesor jazyka ABAP a databázové rozhraní. Každý pracovní proces komunikuje s databází samostatně. Pracovní procesy jsou úzce spjaty s dispečerem. Dispečer slouží jako přístupový bod prezentační vrstvy k aplikační vrstvě. Přebírá dialogové kroky od prezentační vrstvy a předává je ke zpracování vhodnému pracovnímu procesu. Jeden dialogový krok je přiřazen právě jednomu pracovnímu procesu.[6] Pokud se na pracovní proces v AS ABAP díváme z pohledu operačního systému, na kterém je tato aplikační vrstva provozována, pak pracovní proces odpovídá právě jednomu procesu operačního systému.[6]



Obrázek 2.2: Schéma pracovního procesu [15]

Interpret dialogů poté, co je mu předán kontext od dispečera, volá konkrétní Dynpro resp. jeho „logiku toku“ (flow logic). Zajišťuje korektní předání všech položek prezentační vrstvy (textová pole, označení polí, GUI tabulky apod.) do logiky toku. AS ABAP obsahuje speciální jazyk pro definici logiky toku, ve které se definují moduly, které se při zpracování volají. Tyto moduly tvoří pomyslné pojítko mezi interpretem dialogů a interpretem jazyka ABAP. Interpret dialogů předává interpretu jazyka ABAP informaci o tom, který z modulů je na řadě na základě definice logiky toku a ten pak modul interpretuje. 2.2

Databázové rozhraní zajišťuje komunikaci mezi programovacím jazykem ABAP a databázovým systémem. Jazyk ABAP v sobě integruje standard Open SQL, který je podmnožinou standardního SQL. Databázové rozhraní je klíčovou součástí platformy SAP R/3, protože pro koncového uživatele zapouzdřuje databázi a to tak, že konvertuje všechny databázové požadavky do standardního jazyka SQL pro daný databázový systém. Tímto je zaručena nezávislost aplikací v jazyce ABAP na databázi, kterou využívá. Jazyk ABAP podporuje i nativní SQL, jeho používání však není doporučeno, právě z důvodu možné nekompatibility napříč databázovými systémy.[8]

Programovací jazyk ABAP

ABAP je strukturovaný, imperativní, procedurální, objektově orientovaný, vyšší programovací jazyk vytvořený společností SAP výhradně pro použití na jejích platformách. Poprvé se objevil již na platformě SAP R/2 v roce 1983. V současné době je jazyk ABAP základem pro drtivou většinu aplikací na platformách společnosti SAP. Zdrojové kódy jazyka ABAP jsou ukládány ve dvou formách v databázových tabulkách: ve formě zdrojového kódu odkud jsou dostupné uživateli pomocí nástroje ABAP Workbench a ve formě generovaného, přeloženého binárního kódu, který může být interpretován.

ABAP Workbench

ABAP Workbench je součástí každého systému SAP R/3 a umožňuje uživateli programování v jazyce ABAP, editaci databázových tabulek, editaci Dynper a další. Základním nástrojem programátora je Object Navigator. Mezi jeho důležité součásti patří: [1]

- ABAP Editor – pro psaní reportů
- ABAP Dictionary – pro práci s databázovými tabulkami, globálními typy, doménami apod.
- Menu Painter – pro editaci a vytváření některých prvků uživatelského rozhraní: lišta menu, standardní panel nástrojů, aplikačně specifický panel nástrojů apod.
- Screen Painter – pro editaci obrazovek dynpro jejich logiky toku
- Class Builder – pro práci s třídami v rámci ABAP Objects
- Function Builder – pro práci s funkčními moduly

Dynpro

Jak bylo zmíněno dříve, Dynpro implementuje svůj vlastní jazyk pro jeho logiku toku. Logika toku sestává v zásadě ze dvou události – PBO (proces before output) a PAI (proces after input). PBO se provádí vždy při zavolání obrazovky Dynpro a poté pokaždé ve fázi přípravy dat na aplikační vrstvě před odesláním dat prezentační vrstvě. PAI se provádí po zadání dat na prezentační vrstvě. PAI slouží ke zpracování těchto dat. V každé z těchto událostí uživatel definuje, které moduly mají data zpracovat.[13]

Pro lepší představu uvedu příklad: Předpokládejme program v jazyce ABAP sloužící k uchovávání informací o firemních vozidlech. V tomto programu je možné informace pouze zadávat. Uživatel musí v programu vyplnit základní údaje o vozidle - značku, typ, motorizaci, SPZ, firemního uživatele vozidla apod. Program také disponuje funkcí pro automatické vyplnění údajů o spotřebě paliva z databáze povolených aut ve firemní flotile, na základě značky, typu a motorizace.

Při spuštění programu se volá událost PBO, kde dochází k nastavení polí pro Dynpro tohoto programu. Všechna pole na obrazovce Dynpro jsou přístupná z jazyka ABAP, odkud je možné s nimi manipulovat. Protože jde o první spuštění, pole budou iniciální.

Uživatel vyplní všechna pole a pro automatické načtení spotřeby zmáčkne klávesu „Enter“. Tím se na aplikačním serveru vyvolá událost PAI, v níž dojde k přenesení dat z obrazovky Dynpro do polí v jazyce ABAP. Ihned po dokončení PAI se volá událost PBO. V ní program detekuje vyplněné hodnoty, provede vybrání příslušné spotřeby z databáze a tuto hodnotu uloží do pole v jazyce ABAP, které reprezentuje příslušné pole na obrazovce.

Uživatel zmáčkne tlačítko pro uložení, opět se volá PAI, dojde k vložení nového záznamu do databáze. V PBO by se mohla pole např. promazat, aby mohl uživatel ihned začít zadávat další firemní vozidlo.

Dynpro je možné zavolat buď pomocí příkazu jazyka ABAP „CALL SCREEN“ nebo pomocí kódu transakce. [10]

Pojem trasakce v prostředí SAP

Aplikace a zobrazovaní jejich obrazovek probíhají v tzv. transakcích. Transakce se skládá z množiny dialogových kroků uživatele (vyplnění pole, kliknutí na tlačítko apod.) a množiny dialogových kroků systému (PAI, PBO). V rámci jednotlivých dialogových kroků je nutné udržovat data v databázi v konzistentním stavu. K zajištění tohoto stavu slouží LUW (logical unit of work), což je jedna nedělitelná posloupnost databázových operací, na jejichž začátku a konci je databáze v konzistentním stavu.

Aby systém SAP dokázal zaručit konzistenci databáze z aplikačního pohledu přes více dialogových kroků systému pracuje v tzv. SAP LUW transakcích. Jednotlivé transakce SAP LUW jsou tak, jak jsou zpracovávány jednotlivé dialogové kroky ukládány do aktualizačních tabulek. Teprve po úspěšném dokončení všech dialogových kroků transakce, jsou data zapsána do produktivních tabulek. Pojmem transakce tedy v terminologii systému SAP označujeme úspěšný průběh SAP LUW. [6]

Transakce je zároveň ve světě SAP zaužívaná jako synonymum pro aplikaci nebo program.

2.2.3 Databázová vrstva

Jak bylo napsáno výše, aplikační server komunikuje s databázovou vrstvou pomocí rozhraní, které zajišťuje předklad standardu OpenSQL do nativního SQL pro databázi instalovanou na databázovém serveru.

Systém SAP v tomto kontextu podporuje databáze SAP DB/MaxDB, Microsoft SQL Server, Oracle, IBM a SAP HANA. [14]

2.3 Platforma SAP S/4HANA

V této podkapitole se práce zaměří na platformu SAP S/4HANA. Hlavním tématem bude popis klíčových vlastností in-memory databáze SAP HANA, která je pevně spjata s databázovou vrstvou platformy. Z hlediska aplikační vstvy se bude zabývat především změnami oproti SAP R/3, z nichž většina souvisí s optimalizacemi provedenými za účelem plného využití nových možnosti in-memory databáze.

Platforma SAP S/4HANA byla představena společností SAP v roce 2015 jako přímý následník platformy SAP R/3, ze které vychází. Databáze SAP HANA byla představena o 5 let dříve. Výraznou změnou oproti platformě R/3 je fixace platformy výhradně na databázi SAP HANA.

2.3.1 SAP HANA

Z výše napsaného vyplývá, jaký význam předsavuje databáze SAP HANA pro platformu SAP S/4HANA. Níže se práce zaměří na její klíčové vlastnosti.

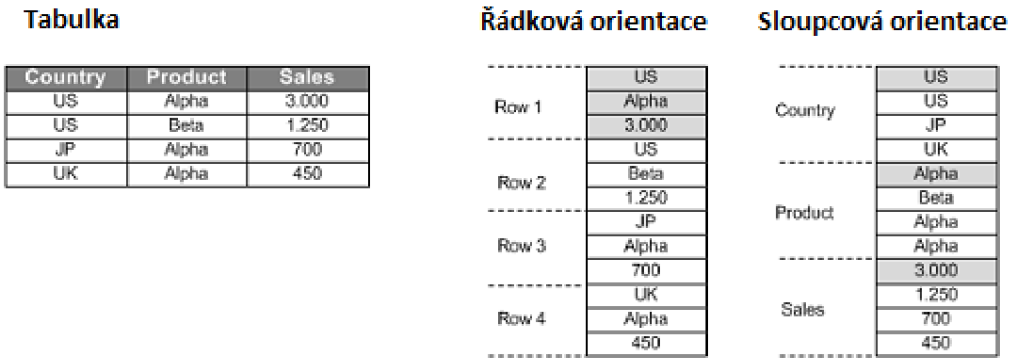
In-memory databáze

S postupně snižující se cenou volatilní paměti RAM našla myšlenka databáze zcela přístupné z operační paměti široké uplatnění v praxi. K databázi v operační paměti může být oproti pevnému disku přistoupeno až 100 000krát rychleji. Je to dáno nejen typem paměti, ale i sběrnicemi, které zpomalují datový tok při použití konvenční databáze uložené na pevném disku. [5]

Sloupcová orientace

Relační databáze mohou organizovat data buď v řádcích nebo sloupcích. SAP S/4HANA využívá, s výjimkou systémových tabulek, ve většině případů sloupcovou orientaci. [5]

Orientace se odráží v uspořádání dat v paměti počítače. Zajímco v řádkově orientované databázi jsou záznamy ukládány po řádcích, ve sloupově orientovaných se ukládají po sloupcích viz obr. 2.3.



Obrázek 2.3: Porovnání řádkové a sloupcové orientace [9]

Síla sloupcově orientovaných databází se nejvíce projeví při práci s obrovským množstvím dat. Pro ilustraci můžeme použít tabulku obsahující 10^7 záznamů o pěněžních transakcích. Nad těmito daty chceme provést operaci agregace, která se týka pouze sloupce udávající částku dané transakce. Při řádkové orientaci by SŘBD musel provést operaci čtení pro každý řádek, aby mohl přistoupit k položce částky. Oproti tomu při sloupcové orientaci bude stačit pouze jedna operace čtení, pro načtení celého sloupce. Sloupcová orientace je velice vhodná pro OLAP (online analytical processing). [5]

Operace typu insert-update jsou z logiky věci ve sloupcově orientované databázi složitější. SAP HANA náročnost těchto operací kompenzuje právě in-memory přístupem. Dále pro operace typu insert-update využívá rozdílové tabulky, které se s fyzickou databází periodicky synchronizují. Mezi další optimalizace patří paralelní zpracování jednotlivých sloupců a fragmentace databáze. [5]

Optimalizace operací typu insert-update umožňují použití databáze SAP HANA nejen pro OLAP, ale i pro OLTP (online transaction processing), což je z hlediska platformy SAP S/4HANA klíčové.

Komprese dat

Základním typem komprese na úrovní sloupců je v databázi SAP HANA tzv. slovníková komprese. Hodnoty v příslušném sloupci jsou přidány do slovníku a ze sloupce jsou odkazovány [5] viz obr. 2.4. Tímto jsou odbourány redundantní hodnoty ve sloupci.

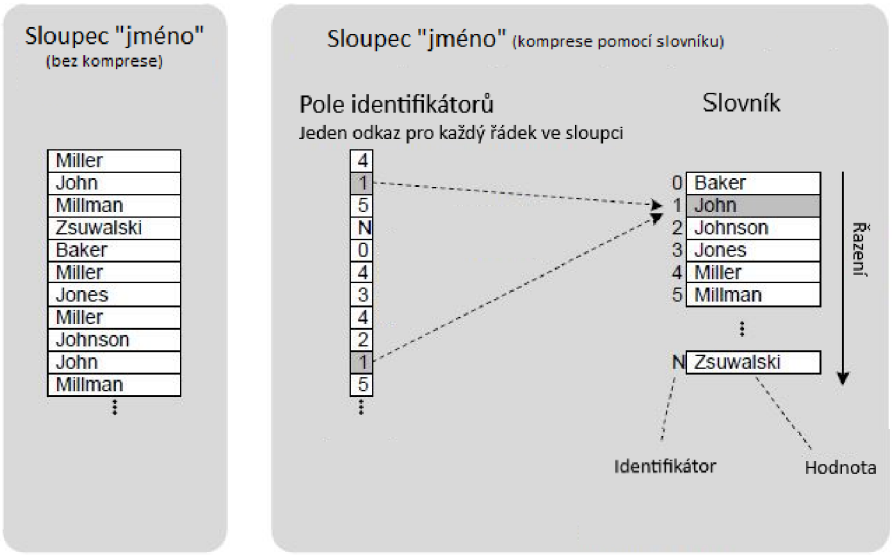
Mimo této základní jsou na sloupce algoritmicky aplikovány také další vhodné komprese na základě povahy dat v těchto sloupcích. [5]

Paralelní zpracování

SAP HANA masivně využívá možností miltiprocesorových systémů. Požadavky na zpracování jednotlivých sloupců jsou přidělovány volným procesorovým jádrům k jejich paralelnímů vykonání. [5]

Fragmentace

Fragmentaci v tomto případě chápejme jako rozdělení fyzické databáze mezi více výpočetních clusterů. SAP HANA implicitně omezuje maximální počet řádků tabulky na dvě



Obrázek 2.4: Slovníková komprese v SAP HANA [2]

miliardy záznamů. V případě dosažení tohoto limitu musí být dotčená tabulka fragmentována na více serverů.

Tabulky mohou být samozřejmě fragmentovány i bez nutnosti dosažení zmíněného limitu, fragmentace do oddílů pod více výpočetních clusterů výrazně napomáhá snížení odezvy databáze. Oddíly mohou být determinovány hashovací funkcí, algoritmem round-robin, rozděleny podle rozsahu určitých hodnot nebo vhodnou kombinací těchto přístupů. [5]

2.3.2 Aplikační vrstva SAP S/4HANA

Masivní změna na úrovni databázové vrstvy otevřela možnosti optimalizacím aplikací aplikační vrstvy, aby maximálně benefitovaly z možností in-memory databáze. Nová technologie vytvářející sémantickou vrstvu mezi databází a aplikační vrstvou se nazýva CDS (Core Data Services). Byť tato technologie figuruje na pomezí mezi těmito dvěma vrstvami, častěji je řazena mezi služby aplikační vrstvy, kde je i definována.

Architektura aplikační vrstvy byla v zásadě převzata z platformy SAP R/3.

CDS (Core Data Services)

Technologie CDS umožňuje programátorům vytvářet znovupoužitelné sémantické pohledy nad tabulkami nebo databázovými pohledy databáze SAP HANA. Obsahuje také podporu pro operace využitelné při vytváření modelů pro OLAP jako agregace, spojení více selekcí do jedné projekce apod.

2.3.3 Prezentační vrstva

Z hlediska prezentační vrstvy je stále nejčastěji používáno rozhraní SAP GUI, popisované v kapitole 2.2. U nově vznikajících aplikací se však čím dál častěji používá rozhraní SAP Fiori.

2.3.4 SAP Fiori

SAP Fiori je nové uživatelské rozhraní společnosti SAP, představené roku 2013. Využívá technologii HTML5 a vlastní framework sloužící k definici uživatelského rozhraní UI5. Elementy tohoto frameworku umožňují snadné zakomponování datového modelu definovaného pomocí CDS pohledů.

S novým uživatelským rozhraním přibyla na straně aplikačního serveru podpora pro komunikaci pomocí protokolu oData. Zajišťuje ji služba SAP Gateway.

Rozhraní Fiori je možné využívat i na novějších verzích platformy R/3.

Kapitola 3

Platební formáty v prostředí SAP

Platebním formátem označujeme soubor pravidel, kterým musí odpovídat výstupní soubor s informacemi v našem případě o platbách. Termín se používá především v souvislosti s automatickými a hromadnými platbami. Slouží pro obousměrnou komunikaci mezi finanční institucí a klientem této finanční instituce. Pro příchozí a odchozí platby (směr v této práci bude vždy udáván z pohledu klienta) se zpravidla používají rozdílné typy formátů.

Situace s platebními formáty není jednoduchá. Existují sice instituce zaštiťující jednotlivé formáty, např. SWIFT, ale jednotlivé bankovní instituce si formáty modifikují pro svoje potřeby a vydávají svá vlastní pravidla. Formáty se ovšem začínají lišit již na úrovni konkrétního státu. Každá země má trochu jinou legislativu, týkající se plateb resp. informací nutných pro jejich uskutečnění. Klient finanční instituce je nucen striktně dodržovat tato pravidla, pro odchozí soubory musí být formát přesně v požadovaném formátu, oproti tomu pro příchozí soubory musí být klient schopen data od finanční instituce zpracovat a převést do účetnictví.

Z výše napsaného jasně vyplývá potřeba lokalizace platebních formátů.V prostředí SAP se pro tyto účely používá transakce DMEE (Data Medium Exchange Engine), především na platformě R/3. Pro S/4HANA existuje vylepšená transakce DMEEX (eXtended).

V této kapitole se bude práce zabývat popisem integrace výše zmíněných transakcí do procesu plateb v prostředí SAP. Po popisu integrace a obecné funkcionality transakcí bude následovat ukázka definice reálného platebního formátu a následně jeho implementace v transakci DMEEX

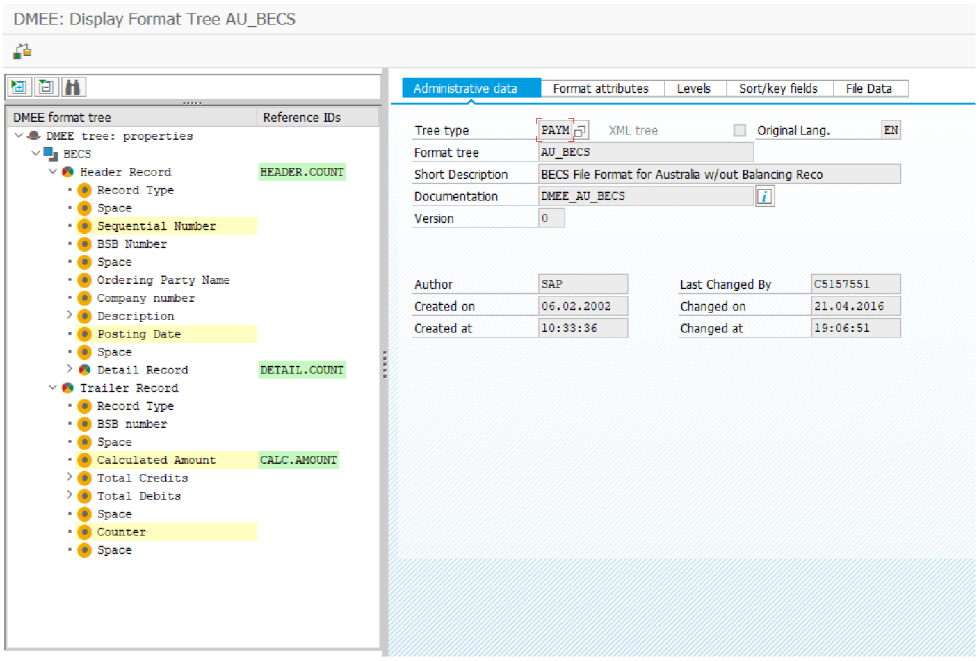
3.1 Payment Medium Workbench

Payment Medium Workbench (PMW) je soubor nástrojů, které slouží k vytvoření, úpravám a správě platebních médií. Uživateli umožňuje vytvoření velice komplexní definice zpracování platebního formátu. [12] Pro účel této práce je popis velmi obsáhlé funkcinality těchto nástrojů nadbytečný. Je důležité PMW zmínit z toho důvodu, že integruje funkcionalitu transakcí DMEE a DMEEX. Uživatel systému SAP si pro každé platební médium může nadefinovat zpracování pomocí DMEE popř. DMEEX. V případě této volby PMW předá vstupní data pomocí pevně definovaného rozhraní transakci DMEE resp. DMEEX. Transakce poté vrátí vygenerovaný soubor zpět PMW, který soubor uloží a popř. odešle definovanou cestou do bankovní instituce.

Transakce DMEE a DMEEX slouží v rámci PMW jako nástroj umožňující lokalizaci či přizpůsobení platebních formátů potřebám konkrétní bankovní instituce.

3.2 Transakce DMEE

Transakce DMEE slouží k definování exaktních pravidel pro generování výstupních souborů a to buď souborů typu „flat-file“ nebo ve formátu XML. Její účel je naprosto obecný a může být použita pro vygenerování souboru jakéhokoli účelu nebo formátu. DMEE tedy umožňuje koncovému uživateli systému SAP definovat i formáty pro finanční instituce bez znalosti programování a jazyka ABAP. Jak již bylo řečeno, tato potřeba vyplývá z neexistence jednotného přístupu k platebním formátům nejen celosvětové, ale i na úrovni jednotlivých států. Vzhledem k tomu, že se tato práce soustředí převážné na oblast plateb, bude se zabývat pouze definičními stromy typu PAYM (3.2.1) v transakci DMEE nebo DMEEX. Transakce DMEE, která vznikla v roce 2000 na platformě R/3 a je stále velmi populární, se skládá ze dvou stěžejních částí. První z nich je grafické uživatelské prostředí (obrázek 3.1), ve kterém si může uživatel nadefinovat formát pomocí definičního stromu (dále DME strom). To umožňuje flexibilitu, která je nutná, aby mohli klienti finančních



Obrázek 3.1: Grafické uživatelské rozhraní aplikace DMEE

institucí vytvářet svoje vlastní stromy nebo upravovat ty standardně dodávané společností SAP. SAP dodává standardní stromy z toho důvodu, aby pro jejich lokalizaci stačily dílčí změny a zákazník nebyl nucen celý strom vytvářet od začátku. Grafické rozhraní se skládá v levé části z grafické reprezentace definičního stromu reprezentovaného uzly různých typů a v pravé části z detailní konfigurace jednotlivých uzlů.

Druhou částí je engine, který slouží pro generování samotného výstupního souboru.

3.2.1 Typ stromu

Každému DMEE stromu musí být při vytvoření přiřazen určitý typ v závislosti na tom, pro jaký účel bude využíván. Od typu stromu se odvíjí vstupní struktury, které jsou při volání DME stromu (generování souboru) naplněny daty ze systému SAP. Typ PAYM, kterým se tato práce výhradně zabývá, využívá struktury FPAYH, FPAYHX a FPAYP. Struktura

FPAYH obsahuje pole sloužící pro detailní popis platby, FPAYHX tato data rozšiřuje. Záznam v FPAYP popisuje placenou položku. Struktury jsou velice rozsáhlé FPAYHX a FPAYH mají přibližně po 150 položkách, FPAYP pak 79.

3.2.2 Verze stromů

DMEE pracuje v základu se dvěma základními verzemi stromů. Každý strom má verzi aktivní (databázově 000) a verzi pro údržbu (maintenance, databázově 001). Aktivní verze se využívá produkčně, při požadavku na vygenerování výstupního souboru pomocí DMEE. Ve chvíli, kdy uživatel potřebuje provést nějaké změny v mapování (ve stromu), pracuje výhradně s verzí 001, aby mohla být verze 000 dále produktivně využívaná v platebních bězích. Uživatel vytvoří novou verzi 000 aktivací verze 001, jejíž součástí je důkladná kontrola konzistence stromu, který musí splňovat určitá syntaktická pravidla. Jakýkoliv strom je možné manuálně verzovat přes správu verzí. Číslo nové manuální verze se vždy inkrementuje o 1 počínaje 002. Uživatel může manuálně vytvořené verze obnovit. Při obnovení dojde k přepsání verze 001 (maintenance).

3.2.3 Jednoduchý příklad výstupního souboru

Pro lepší demostraci funkcí transakce DMEE použijeme jednoduchý příklad výstupního souboru. Tento příklad bude obsahovat právě jednou nadpis v hlavičce a aktuální datum. Dále se bude až n-krát opakovat blok s číslem účtu odesílatele, uníkátní ID platby, číslo účtu příjemce, částka a měna. Definice pomocí vlastní jednoduché syntaxe se nachází níže.

```
Demo DMEE stromu Datum: <datum>
{<c.u. odesilatele>[11]
<UID>[17]<c.u. prijemce>[11]<castka>[7]<mena>[2]}[n]
```

Výpis 3.1: Příklad definice výstupního souboru

V tomto demonstračním příkladu údaje ohraničené < > označují data, která se mají na tomto místě v souboru vyskytovat. [ ] ohraničují definici fixní délky pole, v případě použití za ohraničením bloku { } pak kardinalitu tohoto bloku. Prostý, statický text není nijak speciálně označen.

3.2.4 Data hlavičky stromu

Kořenový uzel stromu obsahuje technická nastavení, která jsou aplikována na celý výstupní soubor.

Atributy

Mezi tato nastavení patří volba struktury specifické pro daný formát v případě, že strom vyžaduje i jiná vstupní data než z výše vyjmenovaných standardních struktur pro strom typu PAYM. V případě XML stromů je možné nastavit také XSLT program, který po vygenerování výstupního XML souboru provede definovanou XSL transformaci. Pro stromy typu „flat file“ si uživatel může nakonfigurovat oddělovač oddělující výstupní pole v souboru.

Úrovně (Levels)

Číslo „Level“ udává kolik úrovní z pohledu definičního stromu má daný formát. Level je přiřazován uzlům typu Segment a Segment Group v flat-file stromech a uzlům typu element

ve stromech XML. Pro každý Level se definuje maximální počet opakování ve výstupním souboru. Pokud dojde v přesažení limitu na nejvyšší úrovni, vygeneruje se další soubor. Pokud se tak stane na některé z nižších úrovní, předcházející úroveň se zopakuje.

Demonstrační příklad 3.1 obsahuje dvě úrovně. Úroveň `1` je hlavička výpisu a vyskytuje se v souboru právě jednou. Druhá úroveň je detailní výpis platby, jejíž maximální počet opakování je 9999999, což je i maximální hodnota, kterou DMEE umožní uživateli zadat.

Sort/Key Fields

„Sort/Key Fields“ má velmi úzkou souvislost s úrovněmi. Pomocí tohoto nastavení je možné definovat pravidla pro sestavování jednotlivých úrovní výstupního souboru. Pokud je nějaké pole nastaveno jako „sort field“ Payment Medium Workbench může tuto informaci využít a setřídit data předtím, než jsou předána DMEE. Pokud dané pole označíme jako „key field“ generování dané úrovně bude ukončeno v okamžiku změny této hodnoty a generování úrovně začíná znovu. Každý „Key field“ musí mít přiřazen právě jednu úroveň definující maximální počet opakování hodnot s tímto klíčem.

Kdybychom v demonstračním příkladu vybrali jako klíčové pole pro úroveň 1 `datum` s maximálním počtem opakování 1, došlo by při změně datumu ve vstupní datech k vygenerování nového souboru. Např. pro tři různé datumy bychom získali tři soubory s platbami provedenými v konkrétní den.

Klíčové pole pro úroveň `2` by v našem příkladu 3.1 bylo `UID platby`. Pro každou platbu by se vygeneroval nový blok.

File Data

Pod nastavení „File Data“ je možné specifikovat typ ukončení řádku ve výstupním souboru. V případě, že z nějakého důvodu uživatel potřebuje specifikovat délku výstupních dat z mapovacích uzlů typu element (3.2.5) v bajtech, nikoliv počtem znaků, může si zde tuto variantu zvolit. Některé bankovní instituce také odmítají speciální znaky v soboru. Např. soubor, který by obsahoval jméno dodavatele „Procter & Gamble“ by byl touto bankovní institucí zamítnut. DMEE proto obsahuje funkcionalitu, která všechny znaky definované jako speciální vymaže z výstupního souboru.

3.2.5 Strom typu flat-file

Segment Group

Typ uzlu, který je používán výhradně pro „flat-file“ strom. Pro lepší představu si můžeme Segment Group představit jako logický celek či odstavec výstupního souboru. Skládá se tedy obvykle z více řádků. Uzly typu Segment Group obvykle definují novou úroveň ve výstupním souboru a mohou se opakovat právě na základě definice úrovně. Slouží k seskupení dalších uzlů typů Segment Group nebo Segment.

Z popisu vyplývá, že uzel typu Segment Group definuje soubor bloků. V příkladu výše 3.1 by tedy byl definovaný na úrovni `1`.

Segment

Typ uzlu Segment reprezentuje právě jeden záznam ve výstupním „flat-file“ souboru. Obvykle je tímto záznamem jeden řádek skládající se z různých hodnot, které se mapují pomocí

uzlů typu Element. Segment musí mít mezi svými potomky alespoň jeden uzel typu Composite nebo Element, v opačném případě zahlásí kontrola konzistence chybu.

Segment bychom v příkladu 3.1 použili dvakrát. Jeden na definování hlavičky, druhý na definování opakujícího se bloku s detaily o platbě. Druhý segment by byl definován na úrovni `2`, mohl by v případě více `UID` plateb iterovat.

Composite

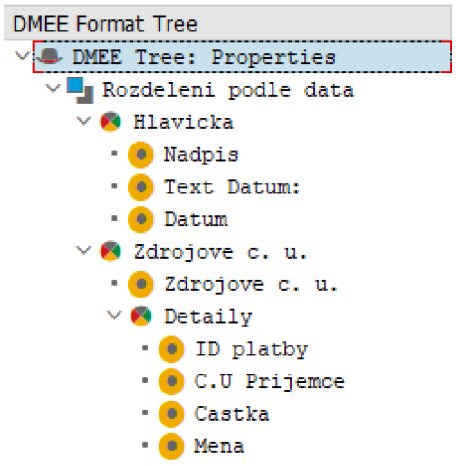
Potomci uzlu typu Composite mohou být pouze uzly typu Element. Composite využíváme v případě, kdy se na za sebou následující uzly vztahuje tatáž podmínka.

Element

Uzel typu element je jedním z uzlů, které umožňují mapování. Pod mapováním si představme přenesení zdrojové hodnoty uzlu na místo ve výstupním souboru na základě definice stromu. Zdrojové hodnoty uzel typu element může získat vícero cestami, které popisuje tabulka 3.1.

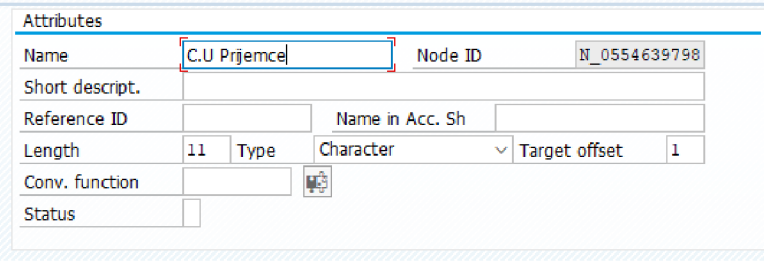
V reálných souborech se ovšem mohou vyskytnout případy, kdy mapování jedna ku jedné nestačí. Na tomtéž místě ve výstupním souboru mohou být různé hodnoty v závislosti na určité podmínce (např. měna v závislosti na zemi). Aby bylo možné této variability docílit, byl zaveden uzel typu atom (3.2.5).

Uzel typu Element v definovaném příkladu zajišťuje samotné mapovaní na příslušná místa ve výstupním souboru. Jedná se tedy o pole `Datum, Zdrojové Č.Ú, UID, Č.Ú Příjemce, Částka a Měna`. Dále by se pomocí elementu mapovala konstantní hodnota `"Demo definice DMEE stromu Datum:"` Finální struktura je zachycena na obrázku 3.2, kde `Rozdeleni podle data` je uzel typu Segment Group, `Hlavicka`, `Zdrojove c. u` a `Detaily` jsou uzly typu Segment, ostatní uzly jsou typu Element.



Obrázek 3.2: Demonstrační příklad v DMEE

Na elementu je možné nastavit další parametry důležité pro mapování. Pole `Length` určuje délku, kterou daný element bude představovat ve výstupním souboru. V souborech typu flat-file, kde jsou pozice určovány pomocí pevných délek a ofsetů, jde o klíčovou funkcionalitu. Pole `Target Offset` určuje ofset výstupu dalšího uzlu. Funkčnost těchto polí můžeme demonstrovat na uzlu `C.U. Prijemce` (3.3) z demonstračního příkladu.



Obrázek 3.3: Nastavení uzlu `C.U. Prijemce`

Výstup při této konfiguraci uzlu je na obrázku níže (3.4). Za povšimnutí stojí mezera mezi výstupem uzlu `C.U. Prijemce` a uzlem `Castka`. Ta je vytvořena ofsetem o hodnotě 1. Kdybychom hodnotu zvýšili na 3, mezery budou 3. Dále kdybychom např. snížili délku o 2, čísla učtu se přenesou do výstupního souboru bez údajů za symbolem lomeno.

```
Demo DMEE stromu Datum: 2020011
154441/145
00000000000000111499111117/44 544CZK
154441/145
00000000000000035474444587/44 221USD
154441/145
00000000000000021322124154/49 111EUR
845511/145
00000000000000045874747474/88 448EUR
845511/145
00000000000000011541231312/77 118USB
777844/412
00000000000000045412123518/04 488GBP
777844/412
00000000000000022581647445/15 151CZK
```

Obrázek 3.4: Nastavení uzlu `C.U. Prijemce`

Atom

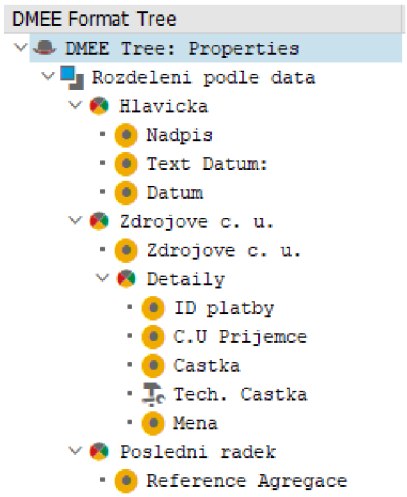
Jediná pozice ve stromě, na které se atom může objevit, je jako potomek uzlu typu element. Existuje několik případů, kdy je vhodné atom použít. Pokud potřebujeme do nadřazeného elementu přenést právě jednu hodnotu z různých zdrojových polí na základě podmínky, můžeme pod takovým elementem definovat atomy s podmínkami. DMEE Engine poté vybere hodnotu z prvního atomu, který splnil definovanou podmínku a přenese hodnotu do elementu. Další příklad může být konkatenace více zdrojových polí do jednoho nadřazeného elementu a to buď s mezerou nebo bez ní.

Technický uzel

Technický uzel je ve své podstatě elementem, jehož hodnota mapovaná ze zdroje není součástí výstupu do výstupního souboru. Hodnota technického uzlu může být referována pomocí Reference ID. Reference ID může být použito v Elementu jako zdroj hodnoty, v podmínkách či agregacích (3.2.7).

V našem příkladě by našel uplatnění pokud bychom například chtěli agregovat všechny částky a jejich sumu poté zapsat do výstupního souboru jako poslední údaj. Technický uzel by v iteracích pod segmentem `Detaily` postupně sčítal všechny částky. Tento technický

uzel by poté sloužil jako zdroj jinému elementu, který výsledek agregace vypíše. Rozšířeny demonstrační příklad je na obrázku 3.5.



Obrázek 3.5: Demonstrační DMEE strom s technickým uzlem

3.2.6 Strom typu XML

Element

Uzel typu element reprezentuje element výstupního XML souboru, též můžeme říci jeden řádek. Potomky tohoto uzlu mohou mít další elementy, které se ve výstupním souboru projeví jako pod-uzly, uzel typu XML atribut a Atom, jehož funkce je naprosto stejná jako v případě „flat-file“. Na rozdíl od elementu ve „flat-file“ stromu je možné definovat úroveň, a tedy docílit iterací XML elementů. Značky XML ve výstupním souboru odpovídají názvům uzlů v DMEE stromu.

XML Atribut

Jako potomek určitého elementu přidá ve výstupním souboru do těla počáteční značky XML atribut nebo atributy. Identifikátor atributu odpovídá názvu uzlu v DMEE stromu.

3.2.7 Mapování

Mapování budeme chápat jako přenesení hodnoty z uzlu na jasně definované místo ve výstupním souboru. Z výše napsaného vyplývá, že ne všechny uzly slouží k mapování. Uzly můžeme rozdělit do dvou skupin. První z nich jsou řídící uzly, které dodávají výstupnímu souboru strukturu a je za jejich pomocí možné vytvářet skupiny či opakovaní (segment, segment group). V XML stromech nejsou tyto řídící uzly potřeba, protože struktura výstupního souboru je jasně daná již samotnou definicí XML. Jediným řídícím uzlem v XML stromu může být z logiky XML struktury element. Hodnota mapovaná do výstupního souboru může pocházet s různých zdrojů.

Volba mapování	Popis funkcionality konkrétní volby
Konstanta	Konstantní hodnota.
Reference	Reference na jiný uzel.
Agregace	Sumarizace hodnot předaných referencí.
Exit modul	Uživatelské mapování, zprostředkované voláním funkčního modulu v jazyce ABAP.
Atomy	V případech, kdy na jednom místě ve výstupním souboru vybíráme z množiny možností nebo hodnoty chceme konkatenovat.
Žádné mapování	Pro elementy v XML, jejichž podstrom obsahuje další elementy.

Tabulka 3.1: Možnosti mapování v aplikaci DMEE.

Reference

Reference slouží k odkázání se na hodnotu uzlu s určitým referenčním identifikátorem. Nejčastěji se tato funkcionalita používá se spojitosti s technickými uzly. Typickým důvodem pro použití referencí je odkázání se na hodnotu uzlu v podmínce či její použití v rámci agregace. Referenční uzel musí být při běhu zpracován před samotným odkazováním, aby již uzel měl načtenou svou hodnotu. V opačném případě dojde k chybě v běhovém prostředí.

Podmínky

Podmínky jsou vyhodnocovány při běhu na začátku zpracování konkrétního uzlu. V případě, že je výsledek po vyhodnocení podmínek `false`, uzel, na němž jsou tyto podmínky definovány, se neprovádí. Podmínka podporuje standardní porovnávací operátory a operaci negace. Zdrojem operandů podmínek mohou být reference, zdrojová pole a konstanty.

Agregace

Agregace je dalším z typů zdrojových hodnot pro uzel. Agregují se výhradně hodnoty předané referencí s tím, že uživatel má na výběr několika typů agregace. Nejčastěji používaným typem je suma všech hodnot (např. suma všech částek), popř. suma všech absolutních hodnot (např. pro výpočet obratu na určitém účtu) předaných referencí. Třetím typem je počet výskytů hodnot vstupujících do agregace (např. počet všech provedených bankovních příkazů).

3.2.8 DMEE Engine

DMEE Engine je označení funkcionality, která zajišťuje interpretaci uživatelem definovaného stromu. Vzhledem k faktu, že DMEE je nástroj obecný a znovupoužitelný, je dodáván se standardním rozhraním, pomocí kterého je možné DMEE funkcionalitu integrovat do takřka libovolného procesu. Děje se tak zaintegrováním tří funkčních modulů:

- `DMEE_START`
- `DMEE_PUT_ITEM`
- `DMEE_END`

Funkční moduly musí být volány v přesně daném sledu. V integraci musí být zajištěno volání funkčního modulu `DMEE_START` jako prvního se všech zmíněných. Zajišťuje totiž kompletní nastavení běhového prostředí. Nastavují se globání data nutná pro zpracování stromu na základě předaného typu a jména stromu. Dochází z sestavení běhové reprezentace stromu. Ta se skládá ze samotné hierarchie uzlů, kterou definoval uživatel a generovaných podprogramů pro každý uzel. Podprogramy jsou vytvořeny automaticky opět na základě definice a též se generují pro zpracování podmínek.

Volání funkčního modulu `DMEE_PUT_ITEM` slouží k předání vstupních dat do DMEE. Děje se tak předáním právě jednoho řádku ze vstupních tabulek. `DMEE_PUT_ITEM` musí tedy být volán v iteraci nad vstupními tabulkami a počet jeho volání odpovídá počtu řádků ve vstupních tabulkách. Tento funkční modul tedy zajišťuje postupné volání podprogramů vygenerovaných v rámci funkčního modulu `DMEE_START` a to ve správném pořadí se zohledněním definice řídících struktur stromu `Sort/Key Fields`. S každým dalším voláním funkčího modulu dochází k obohazení pracovní verze výstupního souboru o předané hodnoty a to jak v případě flat-file, tak XML stromu.

`DMEE_END` Ve funkčním modulu `DMEE_END` dochází k finalizaci dat, která byla vytvořena postupným voláním funkčního modulu `DMEE_END`. Finalizace probíhá přesunem pracovních dat do tabulky, která je definována rozhraním funkčního modulu. V případě XML se ještě před tímto krokem renderuje jeho textová podoba na základě objektové reprezentace DOM objektu a pokud je definována, aplikuje se XSL transformace.

3.3 Transakce DMEEX

S příchodem platformy S/4HANA bylo v roce 2016 rozhodnuto o vydání rozšířené verze DMEE. Písmeno „X“ pochází z anglického slova „extended“. Hlavním důvodem k tomuto rozšíření bylo nabídnout uživatelům na novější platformě snadnější údržbu definičních stromů. Jak bylo zmíněno výše, diverzita platebních formátů je velká a s velikostí organizace také logicky vzrůstá počet spravovaných definičních stromů. Pří jakékoli centrální změně platebního formátu, za příklad si vezměme formát MT101 vydávaný organizaci SWIFT, musí být všechny definiční stromy implementující tento formát samostatně aktualizovány. Tento problém se podařilo v DMEEX vyřešit zavedením hierarchie stromů a synchronizací změn napříč všemi stromy v hierarchii směrem shora dolů.

Při vývoji transakce DMEEX se počítalo s nasazením jejího derivátu na Cloud verzi platformy S/4HANA. Funkcionalita byla proto upravena i takovým způsobem, aby omezila nutnost používání uživatelských rozšíření (mapování exit modulem viz tab. 3.1), která na platformě S/4HANA Cloud nemohou být implementována.

3.3.1 Nová funkcionalita oproti DMEE

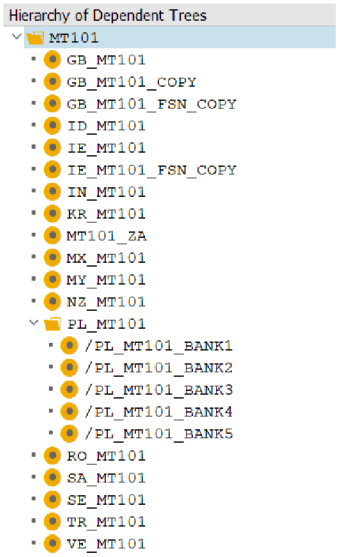
Deaktivace uzlu

Uzel může být deaktivován. V takovém případě se uzel při zpracování stromu neprovádí, ale jeho záznam je stále uložen v databázi a může být kdykoli reaktivován.

Stromová hierarchie

Hierarchie vzniká při vytváření nového stromu, pevným nastavením záznamu o rodičovském stromě. Hierarchie tedy vzniká při vytváření zcela nového stromu nebo při kopírování. Počet úrovní hierarchie je teoreticky neomezený. V praxi se však používají nejčastěji tři úrovně.

Je základním stavebním kamenem již zmíněné funkcionality – synchronizace. V hierarchii se totiž přesně odráží vztahy mezi rodičovskými stromy a jejich potomky. Obrázek 3.6 zobrazuje hierarchii stromu MT101. Kořenový formát odpovídá přesné definici MT101, tak jak je definována organizací SWIFT. Jeho potomci odpovídají specifikacím, vydávaným na úrovni jednotlivých států. Například pod stromem PL_MT101 se nacházejí potomci, kteří implementují formáty pro fiktivní polské bankovní instituce.



Obrázek 3.6: Stromová hierarchie v DMEEX

Synchronizace

Synchronizace v DMEEX je uživatelsky intuitivní způsob pro zavádění změn z vyšších úrovní stromové hierarchie do úrovní nižších. Synchronizaci podléhají všechna data stromu vyjma těch hlavičkových, která se v dané hierarchii nesynchronizují. Synchronizaci nelze v žádném případě spustit přes více než jednu úroveň – tedy je možné synchronizovat vždy z určité úrovně na úroveň o jedna vyšší. Při synchronizaci je potřeba počítat se situacemi, kdy změny přenášené z vyšší úrovně budou kolidovat se změnami provedenými uživatelem na nižší úrovni. Při synchronizaci je dostupný log se zprávami, poskytujícími informace o každé jednotlivé změně, která bude po jejím odsouhlasení přenesena do podřízeného stromu. Zprávy jsou klasifikovány podle vlivu na uživatelské změny v poděděném stromu barvami zelenou, žlutou a červenou. Jako zelené jsou obecně značeny změny, které žádným způsobem neovlivňují uživatelem definované či redefinované (uzel existuje v rodičovském stromu, ale v potomkovi byla jeho definice změněna) uzly a jejich parametry.

Z uživatelského pohledu existují dva základní módy synchronizace. Prvním z nich je synchronizace ze strany potomka. Ta je uživateli nabídnuta ve chvíli, kdy do takového stromu vstoupí, přičemž rodič tohoto stromu byl změněn. Uživateli je nabídnut seznam změn. Druhým módem je hromadná synchronizace všech přímých potomků jednoho rodičovského stromu v jeden okamžik.

Kalkulační uzel

Přídáním kalkulačního uzlu bylo odstraněno velké množství případů použití, při kterých se uživatel musel spoléhat na mapování pomocí uživatelského rozšíření v jazyce ABAP. Mezi tyto případy patří např. číslování řádků, operace odečítání, složitější sumace, na které nestačí funkcionalita agregací apod.

Uživatel DMEEX dostává možnost definovat si promenné a kalkulační uzly ve kterých může nad těmito operacemi definovat základní matamatické operace.

3.3.2 DMEEX Engine

Z hlediska výstupu i rozhraní je DMEEX Engine stejný jako DMEE Engine. Hlavní změnou je univerzálnost jeho kódu, protože se již negenerují podprogramy pro každý uzel. Zpracování každého uzlu je prováděno pomocí stejného zdrojového kódu, což významně zvýšilo jeho přehlednost a udržovatelnost. Tímto krokem došlo mj. ke zrychlení zpracování stromů v průměru o přibližně 50%.

3.3.3 Cloudová verze DMEEX

V roce 2017 bylo rozhodnuto o vytvoření sesterské aplikace DMEEX pro S/4HANA Cloud. Na rozdíl od standardních on-premise dodávek systému SAP, kdy je celý systém provozován na vlastním hardwaru zákazníka, kterému jsou umožněny jakékoli modifikace (včetně zdrojových kódů) a rozšíření, v Cloud řešení si zákazník prostor pouze pronajímá od společnosti SAP. Systém je tedy dodáván tak, jak je, a jakékoliv modifikace nejsou možné.

3.3.4 Příklad definice platebního formátu

Pro demonstaci vytvoření platebního formátu v transakci DMEEX jsem zvolil na internetu volně dostupný dokument od Rabobank[1]. Vzhledem v obsáhlosti dokumentu jsem vybral pouze malou skupinu uzlů XML, na které je však možné dobře demonstrovat, jak se DMEEX používá v praxi. Oproti předchozímu příkladu jde v tomto případě o typ stromu XML.

Dokument od Rabobank definuje tři základní skupiny uzlů, ze kterých se skládá výsledný souboru. První skupinou jsou hlavičková data souboru, ze kterých jsem pro demonstraci vybral uzly zaznamenané v tabulce 3.2.

Jméno	XML-tag	Úroveň	Datový typ	Délka	Popis
Document	<Document>	Top			
	<CstmrCdtTrfInitn>	1			
Group Header	<GrpHdr>	2			
Message Identification	<MsgId>	3	Alphanumeric	35	Unikátní ID souboru
Creation Date Time	<CreDtTm>	3	Date+time		Datum a čas vytvoření souboru Např. 2012-02-03T11:20:45
Control Sum	<CtrlSum>	3	Amount	18	Suma všech transakcí (částek) v souboru

Tabulka 3.2: Definice hlavičkových dat platebního formátu. Každá informace se vyskytuje v souboru právě jednou.

Druhá skupina představuje informaci o platbě, např. informace o typu platby nebo debetní straně. Tato skupina se může opakovat a zahrhuje v sobě inforamaci o transakcích. Vybrané uzly viz tabulka 3.3.

[1] https://www.rabobank.com/en/images/Format_description_CT-XML.pdf

Jméno	XML-tag	Úroveň	Výskyt	Datový typ	Délka	Popis
Payment Information	<PmtInf>	3	[1..n]			
Payment Information Identification	<PmtInfId>	4	[1..1]	Alphanumeric	35	Unikátní ID platby
Control Sum	<CtrlSum>	4	[1..1]	Amount	18	Suma všech transakcí (částek) v platbě

Tabulka 3.3: Definice informací o platbě v platebním formátu

Poslední skupinou jsou informace o jednotlivých transakcích, které se vyskytují v rámci informace o platbě. Vybráné uzly z této skupiny jsou uvedeny v tabulce 3.4.

Jméno	XML-tag	Úroveň	Výskyt	Datový typ	Délka	Popis
Payment Information	<CdtTrfTxInf>	4	[1..n]			
Payment Information Identification	<PmtId>	5	[1..1]			
Amount	<Amt>	5	[1..1]			
Instructed Amount	<InstdAmt>	6	[1..1]	Amount		Částka transakce

Tabulka 3.4:

Při vytváření stromu na základě podobné definice je vhodné na začátku identifikovat pole vstupních struktur FPAYH, FPAYHX a FPAYP, která budou potřeba pro mapování. Každé pole z těchto struktur má svůj jasně daný účel, na který je potřeba při sestavovaní stromu pamatovat. Zároveň je vhodné si stanovit, která pole mohou iterovat a nad kterými poli vstupních struktur viz tabulka 3.5.

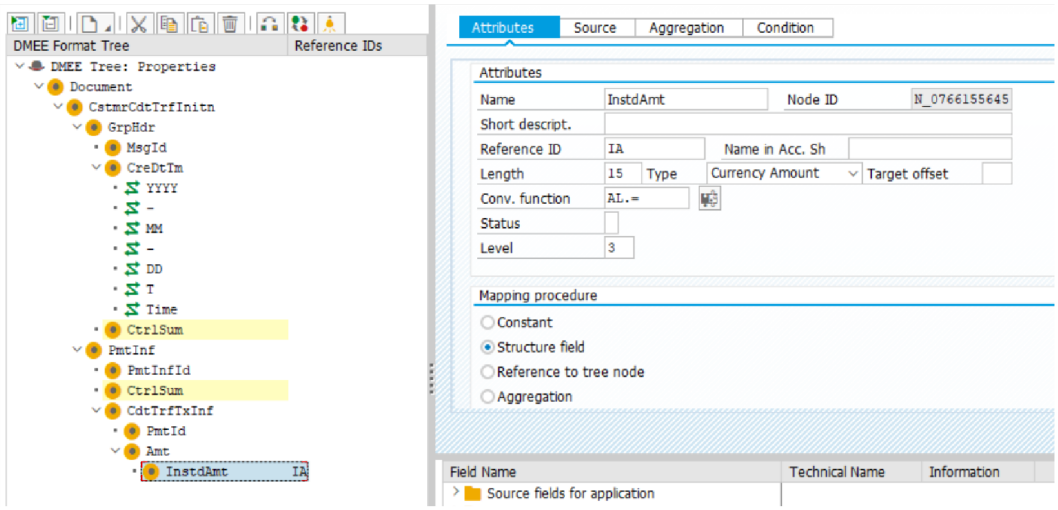
Struktura-pole	Význam	Uzel	Klíčové pole pro úroveň
FPAYH-ZBUKR	Účetní okruh		1
FPAYHX-RENUM	Referenční číslo	MsgId	
FPAYH-HKTID	ID pro podrobnosti o účtu	PmtInfId	2
FPAYH-DOC1R	Reference na platební doklad	PmtId	3
FPAYH-RWBTR	Částka placená v měně transakce	InstdAmt	

Tabulka 3.5: Zdrojová pole vstupních struktur, uzly odpovodné za jejich mapování, definice úrovní (nezaměňovat za úroveň zanoření uzlu) a jejich klíčových polí.

Na základě tabulek 3.2 3.3 a 3.4 můžeme vytvořit strukturu stromu v transakci DMEEX, nastavit správně úrovně pro iterace a parametry jednotlivých uzlů - délky, ofsety, konverze apod. Za pomoci tabulky 3.5 přiřadíme příslušným úrovním jejich klíčová pole a jednotlivým uzlům zdrojová pole. Poté stačí doplnit `Reference ID` k uzlu `InstAmt` a použít ho v agregaci na uzlech pro kontrolní sumy `CtrlSum`.

Uzel `CreDtTm`, tedy uzel zobrazující čas a datum ve specifickém formátu, můžeme definovat pomocí množiny uzlů typu atom, které mapují ze systémových struktur pro datum a čas.

Výsledná struktura stromu s otevřeným uzlem `InstAmt` je na obr. 3.7. Další screenshoty nastavení DMEEX, které se vážou k tomuto demonstračnímu příkladu jsou v přílohách A.1.



Obrázek 3.7: Definovaný strom s detailem uzlu `InstAmt`

Kapitola 4

Aplikace pro ladění definic platebních formátů

Návrh finálního řešení je ovlivňován mnoha faktory. Pro nástroje typu DMEE již existují nástroje, které umožňují testování či ladění. Výsledná funkcionalita navrhované aplikace musí zohlednit existenci těchto nástrojů a mít oproti nim výraznou připadanou hodnotu. Jak bylo demonstrováno, proces vytváření definičních stromů není jednoduchý a uživatelsky příliš přívětivý proces. Je nutné si uvědomit, jak již bylo zmíněno, že koncovým uživatelem je zřídka kdy programátor, který by rozuměl všem technickým pojmům a souvislostem, které vstupují do procesu vytváření stromů.

V následující části bude práce popisovat nástroje, které se v současné době používají pro ladění, a poté předpokládané cílové skupiny této aplikace a jejich možné požadavky. Z této vlastní analýzy vychází prvotní návrh aplikace. Dále se práce zaměří na popis průběhu získávání požadavků a zpětné vazby od cílových skupin. Pomocí toho budou také výsledky prvotní (mojí vlastní) analýzy potvrzeny a rozšířeny.

Dále se práce bude zabývat kontinuální spoluprací mezi mnou jako vývojářem a cílovými skupinami za aplikace metodiky vývoje software Scrum po dobu implementace a testování nástroje pro ladění.

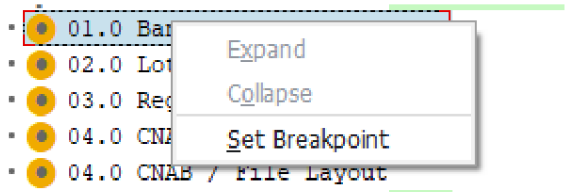
4.1 Analýza současné situace

Práce se zabývá návrhem řešení pro všechny tři platformy. Na platformách S/4HANA a R/3 společnost SAP nabízí použití breakpointů a testovacího nástroje. Na platformě S/4HANA Cloud řešení pro testování a ladění neexistuje.

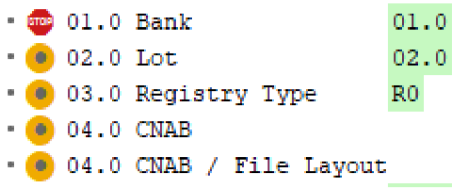
Breakpoint

Uživatel jak DMEE, tak DMEEX disponuje možností umístit breakpoint na konkrétní uzel (viz obr. 4.1 a 4.2) definičního stromu. V DMEEX je tato možnost přímo integrována do aplikace (transakce DMEEX), uživatel starého řešení DMEE musí pro zviditelnění této možnosti přistupovat přes expertní transakci DMEE_DEBUG.

Při nastavení breakpointu dojde k zápisu informace o typu stromu, jméně stromu, ID uzlu, uživatelském jméně a času do tabulky DMEE_DEBUG. Obsah této tabulky je kontrolován na začátku zpracování konkrétního uzlu v DMEE Engine. V případě, že dojde ke shodě v uživatelském jméně a ID uzlu, je pomocí standardního klíčového slova BREAK-POINT programovacího jazyka ABAP vyvolán ABAP Debugger. V tomto místě může uživatel pře-



Obrázek 4.1: Nastavení breakpointu v DMEEX



Obrázek 4.2: Uzel s nastaveným breakpointem v DMEEX

vzít kontrolu a krokovat zpracování stromu příkaz po příkazu. Na obrázku 4.3 můžeme vidět zastavení zpracování v místě uzlu 01.0 Bank (N_5515897960) stromu BR_FEBRABAN.

Testovací nástroj

Testovací nástroj umožňuje provedení zcela nezávislého běhu DMEE stromu. Jak bylo již řečeno, DMEE je nástroj, který vyžaduje integraci do určitého procesu, např. pro platby je integrován do produktu Payment Medium Workbench, jehož součástí je transakce FBPM, která DMEE Engine nad platebními daty spouští. V rámci těchto procesů často dochází k úpravě dat před voláním DMEE Enginu i po něm. Pokud tedy uživatel po spuštění platebního běhu nalezne v souboru chybná data, nelze jednoduše zjistit, v jaké části práce s daty došlo k chybě. Ulehčením v této situaci může být testovací nástroj DMEE.

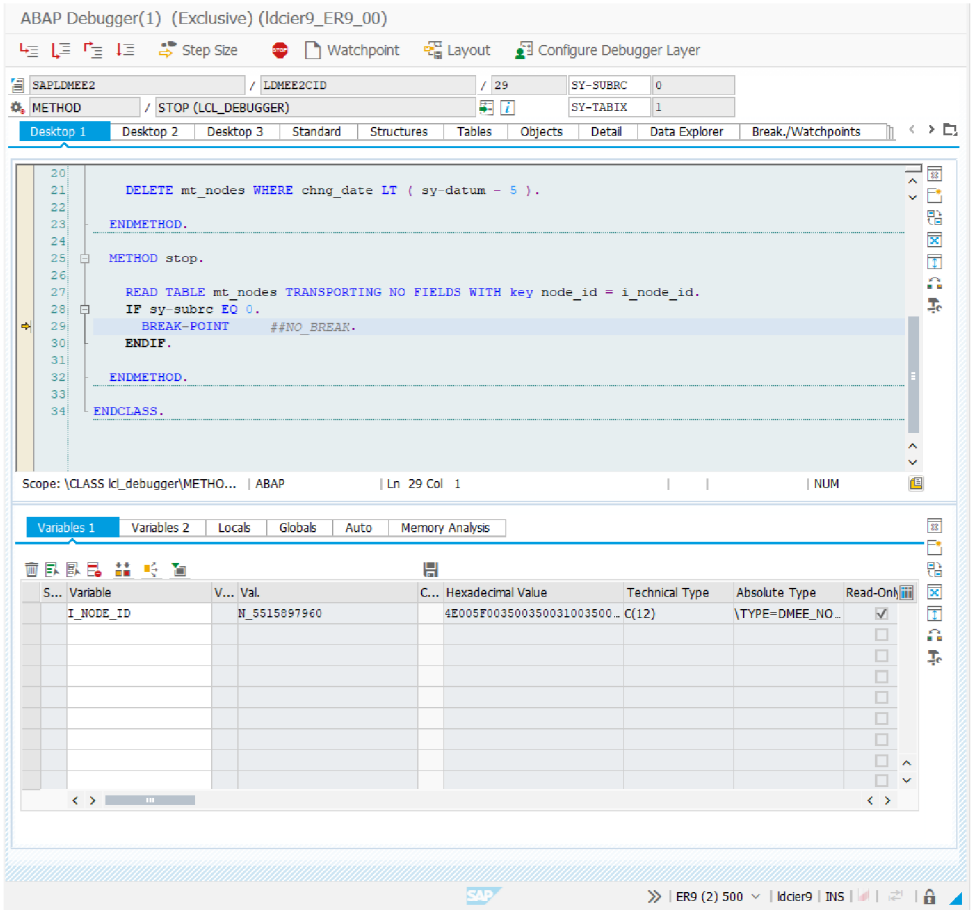
Testovací nástroj DMEE využívá pouze standardní funkční moduly DMEE – `DMEE_START`, `DMEE_PUT_ITEM` a `DMEE_END`. Je specifický pro každý typ stromu, protože jak již bylo řečeno, každý typ stromu má jiné vstupní struktury. Po zadání vhodných testovacích dat do GUI tabulek, v našem případě reprezentující struktury FPAYH, FPAYHX a FPAYP, je stiskem tlačítka Execute zahájeno generování výstupního souboru. Jednoduchý případ použití testovacího nástroje je znázorněn na obrázcích níže. Nejdříve se generuje strom CGI_CT bez dat ve vstupních strukturách (obr. 4.4).

Poté je do tabulky reprezentující strukturu FPAYHX vložena hodnota pro pole RENUM, reprezentující referenční číslo, ale na volbě pole v tomto případě nezáleží. (obr. 4.5).

Na výstupu se tato hodnota objevuje mezi XML značkami <MsgID> (obr. 4.6).

Mezi další významné funkcionality testovacího nástroje patří možnost generování souboru do souborového systému na straně klienta a ukládání variant. Varianta je standardní funkcionalita systému SAP, která umožňuje uložit hodnoty vyplněné v polích na SAP GUI obrazovce. Díky tomu uživatel nemusí po každém spuštění testovacího nástroje vyplňovat tabulky struktur znovu, ale pouze je obnovit z uložené varianty. Kombinace těchto dvou funkcí se nabízí např. při vytváření automatických regresních testů pro DMEE stromy.

Testovací nástroj se však v rámci ladění neuplatní, protože nepokrývá všechny možné integrace DMEE a dále odpovídá pouze na otázku „jak“ vypadá výstup, ale nikoliv „proč“ tak vypadá.



Obrázek 4.3: Zastavení zpracování DMEE stromu na konkrétním uzlu.

Map Payment Format Data (Fiori)

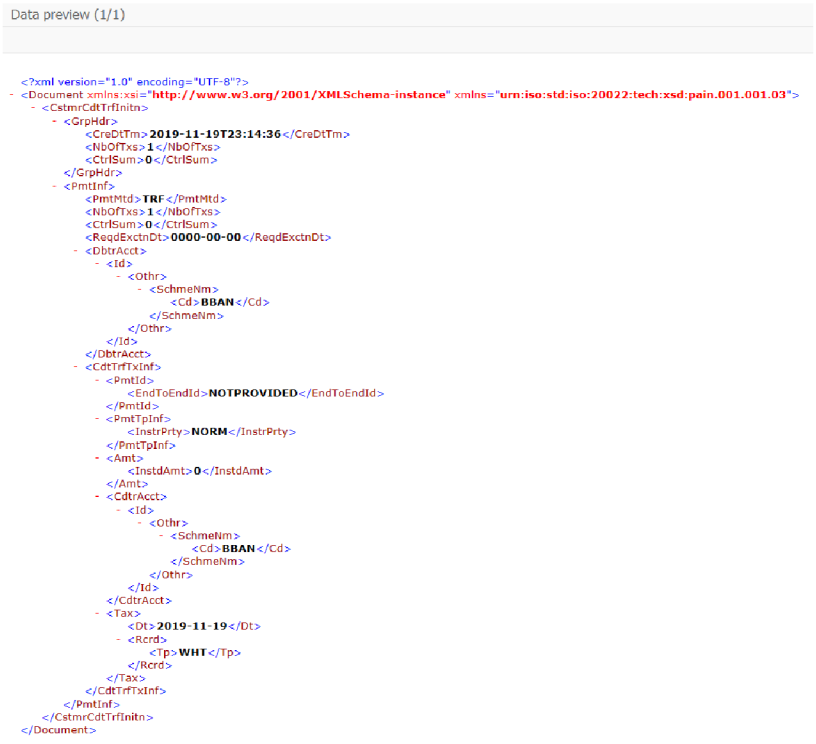
Pro cloud aplikaci Map Payment Fomat Data neexistuje žádný dedikovaný testovací nástroj. Uživatelé jsou tak odkázáni na testovaní v rámci integrace DME Enginu do procesu plateb. Tento fakt je potřeba zohlednit v rámci návrhu nástroje pro ladění. Zahrnutí této myšlenky do návrhu usnadní případnou dodávku nástroje pro ladění na platformu S/4HANA Cloud.

4.2 Údržba DMEE a DMEEX

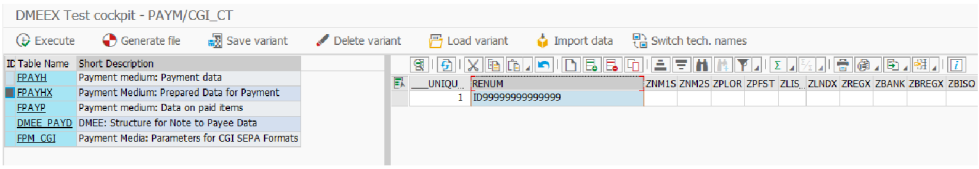
DMEE a DMEEX jsou jako řešení dodávané společností SAP z hlediska životního cyklu softwaru ve fázi údržby. Mimo průběžného ladění a reakcí na především interní požadavky tvoří největší část řešení zákaznických problému. Zákazníci mají několik možností, jak kontaktovat produktovou podporu k produktům společnosti SAP. Standardní cestou jsou zákaznické incidenty.

Zákaznické incidenty mají dvě úrovně podpory. První úroveň řeší základní uživatelské problémy při konfiguraci, používání, odpovídá na dotazy ohledně funkcionality apod.

Druhá úroveň je podpora zákazníka přímo od vývojářů v dané oblasti. Obvykle řeší složitější problémy, chyby v logice chování aplikace, nestabilní chování aplikace a obecně řečeno problémy, které první úroveň podpory nemá možnost pokrýt.



Obrázek 4.4: Výstup stromu CGI_CT s prázdnými vstupními strukturami

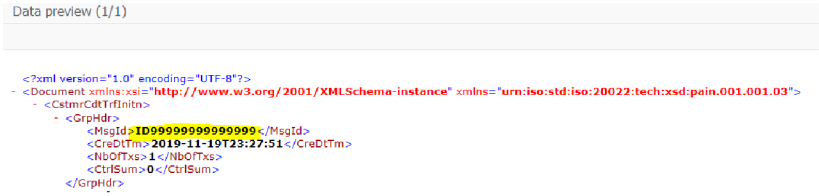


Obrázek 4.5: Vložení hodnoty do FPAYHX-RENUM

V rámci návrhu je nutné odpovědět na otázku, zda nový nástroj přinese zásadní usnadnění v hledání často se opakujících chyb.

Chybějící uzel ve výstupním souboru

Příčin chybějícího uzlu na výstupu může být mnoho. Mezi nejtriviálnější patří nevyplněná hodnota zdrojového pole. Pokud by uživatel používal testovací program popsaný výše, nebylo by složité si nevyplněné hodnoty všimnout pouhým okem. Aplikace založené na DMEE jsou vždy integrovány do nějakého procesu. V rámci tohoto procesu jsou data předpřipravena pro DMEE a uložena do struktur FPAYP, FPAYH a FPAYHX. Uživatel nemá možnosti za běhu DME Enginu vidět obsah těchto struktur jiným způsobem než použitím nástroje ABAP Debugger. Další častou příčinou je nesprávně nastavená délka nebo ofset zdrojového pole. Typicky k tomu dochází při použití tzv. REF polí struktury FPAYHX. REF1-15 pole mohou obsahovat libovolné hodnoty specifikované daným případem použití nebo zákazníkem. Jsou typu CHAR o délce 132 a obvykle se skládají z více údajů, které mají v rámci pole pevně danou maximální délku a ofset. Stává se však, že uživatel omylem zvolí délku nebo ofset chybně. DMEE potom přistupuje do prázdného místa v REF poli. Bez použití



Obrázek 4.6: Část výstupu stromu CGI_CT s hodnotou RENUM (zbytek výstupu stejný jako na obr. 4.4)

nástroje ABAP Debugger je opět velmi obtížné si tuto chybu uvědomit. V lepším případě dojde při zadání chybného ofsetu nebo délky pouze k nevhodnému oříznutí hodnoty, která je poté součástí výstupního souboru. Dalším případem je nesprávné použití podmínky. Jak by napsáno výše, u každého uzlu může být definována podmínka, za které uzel bude součástí výstupního souboru. Uživatel nemá jinou možnost jak analyzovat výsledek podmínky než prostřednictvím nástroje ABAP Debugger.

Chybějící blok ve výstupním souboru

V současné chvíli, bez potřebného nástroje velmi složitě odhalitelný problém. Jak bylo popsáno v podkapitole o transakci DMEE (3.2), iterace pod uzly fungují na principu definice klíčového pole („Key Field“) pro iteraci, úrovně a limitu opakování. Iterace probíhá, pokud existuje další hodnota klíčového pole, která ještě nebyla zpracovaná. Chybnou volbou klíčového pole může být způsobeno to, že se od určité iterace bloky přestanou zpracovávat. Situaci lze demonstrovat na jednoduchém příkladu: Mějme pole o délce 10 s tím, že uživatel si zvolí poslední tři číslice tohoto pole jako klíčová: 0000000**000**. Uvažujme, že uživatel toto pole používá jako jednoznačný identifikátor platby, který se inkrementuje o 1 od 0 výše. Předpokládejme, že v našem příkladě je počet plateb 1001. V okamžiku zpracování platby 1000 by však nedošlo ke změně klíčového pole, protože si uživatel zvolil jen poslední tři číslice, totéž pro platbu s identifikátorem 1001. Dvě platby by tedy ve výstupním souboru chyběly. Uvažovaný příklad je velmi zjednodušený oproti použití v praxi, kdy se klíčové pole může skládat z kombinací více polí a jejich částí vymezenými posuny a délkou.

Jediné možné řešení pro nalezení tohoto problému je použiti nástroje ABAP Debugger nebo velmi zdlouhavá manuální kontrola vstupních dat resp. polí vstupních dat, která jsou v DMEE/DMEEX označena jako klíčová.

Problémy s konverzemi

DMEE i DMEEX nabízejí velké množství konverzí, sloužící ke změně formátu vstupních dat v průběhu zpracování těchto dat na konkrétním uzlu. Hodnoty, které jsou přeneseny do výstupního souboru odpovídají stavu po konverzi. Uživatelé často z důvodu slabé znalosti nebo nepozornosti konverze přehlížejí. Výsledkem pak může být hodnota jiná, než uživatel očekává. Jako příklad můžeme použít vstupní hodnotu částky v amerických dolarech 3.7. Uživatel očekává na výstupu hodnotu 3.7, tedy vstupní hodnotu beze změny. Výstupní soubor však obsahuje hodnotu 3.70, protože je aplikována implicitní konverze podle měny - americký dolar je standardně reprezentován na dvě desetinná místa.

Dalším častým příkladem může být situace, kdy je v konverzích zvolena možnost odebrání speciálních znaků, mezi které se řadí všechny symboly mimo alfanumerických. Uživatel může být potom překvapen, že vstupní hodnota `R&D Company` na vstupu neodpovídá ře-

tězci `RD Company` na výstupu. K odebrání speciálních znaků může docházet i v jiné fázi zpracování výstupního souboru např. v rámci integračně specifického zpracování.

Jediná exaktní cesta pro odhalení problému v chybně zvolené konverzi je použití nástroje ABAP Debugger.

4.3 Cílové skupiny uživatelů

V předchozí podkapitole byly popsány časté chyby, které se vyskytují při definici stromu. Tyto problémy, jako při vytváření jakéhokoliv složitější koncepce, se vyskytují u několika kategorií uživatelů, více či méně.

První skupinou, a z pohledu společnosti vytvářející uživatelský software nejkritičtější, jsou samotní firemní zákazníci. Konfiguraci systému SAP zpočátku a po dobu života projektu konfigurují a udržují specialisté zákazníka, kteří disponují znalostmi nejen technickými, ale také mimořádnými znalostmi systému SAP a firemních procesů. V případě programů postavených na DMEE vstupují tito konzultanti, jak je toto pracovní zaměření nejčastěji titulováno, do prvotní konfigurace stromu a do jeho případných aktualizací podle požadavků autority platebního formátu či bankovní instituce. Důležité je v rámci návrhu aplikace brát ohled na fakt, že konzultanti nejsou programátoři, byť často základní principy programování ovládají. Výsledný nástroj pro ladění by měl umožňovat graficky, přehledně zobrazit hodnoty vstupující do jednotlivých uzlů, výsledky podmínky na uzlech, hodnotu vystupující z uzlu a posloupnost provádění jednotlivých uzlů. Další požadavky budou předmětem „průzkumu mezi uživateli“.

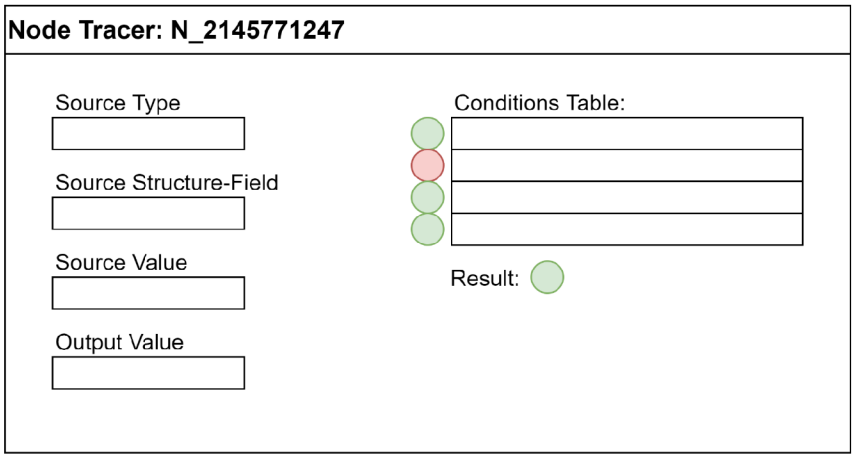
Každý software vstupuje do fáze údržby. Základní podporu pro DMEE a aplikace z ní vycházející zajišťuje produktová podpora (angl. Product Support). Jedná se o první instanci, která se stará o problém zákazníka a hledá pro něj vhodné řešení. Lidé pracující na softwarové podpoře jsou experty v dané oblasti. Jejich expertízu vymezují aplikační komponenty pod jejich odpovědností. Zřídka kdy mají byť jen základní znalost programování. I pro tuto cílovou skupinu je značně nevhodné používat jako jediný možný nástroj pro ladění klasický ABAP Debugger. Stejně jako v případě firemních zákazníků by pro tyto uživatele bylo zjevně užitečnější mít dostupný jednoúčelový nástroj pro ladění stromů.

Poslední cílovou skupinou jsou vývojáři platebních formátů. Tato cílová skupina disponuje nejvyšší znalosti, jak aplikací typu DMEE, tak požadavkům na platební formát. Zároveň má exemplární znalost programování a programovacího jazyka ABAP. Jejich znalost jazyka ABAP jim umožňuje provádět ladění a hledání chyb ve stromě za použití současného řešení. Očekává se od nich největší zpětná vazba k návrhu nového řešení ze všech zmíněných skupin.

4.4 Prvotní návrh

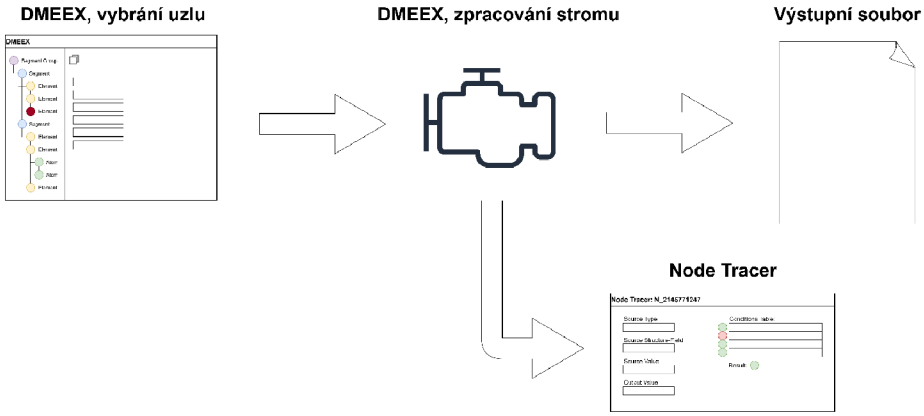
Prvotní návrh vychází z předpokladu, že uživatel ocení možnost vizuálně a přehledně analyzovat data v době zpracování zvoleného uzlu. Vycházel jsem v tomto návrhu z mých vlastních zkušeností. Záměrně jsem prvotní mock-up nakreslil tak, aby ne zcela směřoval zmíněné uživatele určitým směrem. Nástroj pro ladění by měl být svým způsobem pouze rozšířením nástroje ABAP Debugger. Uživatel by si stejným způsobem mohl u uzlu nastavit breakpoint, při jehož zpracování by se uživateli místo ABAP Debuggeru zobrazilo okno nástroje. Zde by měl na výběr mezi zobrazením dat zpracovaných uzlem pomocí nástroje pro ladění nebo by mohl použít staré řešení v podobě ABAP Debuggeru. V prvotním ná-

vrhu se počítá se zobrazením typu zdroje, zdrojové hodnoty uzlu, výstupní hodnoty uzlu a výsledkem podmínky, viz 4.7.



Obrázek 4.7: První návrh nástroje pro ladění. Obsahuje pole pro typ zdroje (reference, pole struktury apod.), hodnotu zdroje a výstupní hodnotu. Tabulka podmínek obsahuje dílčí výhodnocení podmínky - zelená = pravda, červená = nepravda

DMEEX Engine by integroval nástroj tak, že by se v případě zpracování uzlu, který je označený breakpointem, zpracování uzlu zastavilo a uživateli by se místo klasického ABAP Debugger zobrazil tento nový nástroj s možností přepnutí se zpět do prostředí ABAP Debugger, viz 4.8.



Obrázek 4.8: Návrh integrace nástroje pro ladění

4.5 Práce s cílovými skupinami

S každou ze tří cílových skupin jsem zorganizoval meeting, který měl sloužit pro získání kompletního přehledu o každé skupině uživatelů. Základním úkolem bylo zjistit, zda jsou původní předpoklady o cílových skupinách správné a zda skutečně podobný nástroj potřebují. Dalším cílem bylo vytvořit typickou personu uživatele vývojáře, pracovníka produktové

podpory a zákazníka. Každá persóna by měla zachycovat shodné či pro danou skupinu typické znaky: zkušenosti se systémem SAP, znalost programovacího jazyka ABAP, zkušenost s vytvářením DMEE stromů apod. Údaje o počtu zúčastněních z každé cílové skupiny jsou zaneseny v tabulce níže.

Cílová skupina	**Počet účastníku**
Vývojář v oblasti plateb	6
Pracovník produktové podpory	10
Zákazník znalý oblasti plateb	3

Tabulka 4.1: Zastoupení cílových skupin

Pro jasnou strukturu meetingů jsem měl připraveny jasně dané body a otázky na každou z cílových skupin. Průběh meetingu je popsán níže

Představení účastníků, agenda (10min.)

Vzájemné představení, představení agendy meetingu v bodech.

Otázky sloužící pro definici persóny (15min.)

- Jaké jsou vaše zkušenosti se systémy SAP, jazykem ABAP apod.? (otevřená otázka)
- Jak často používáte DMEE/DMEEX?
  - denně, týdně, měsíčně apod.
- Používáte standardní stromy používané společností SAP nebo vytváříte vlastní od začátku?
  - Pokud ano, za jakým účelem?
  - Pokud ano, kolik jste jich vytvořili?
- Máte vlastní nástroj na testování či ladění chyb v DMEE stromech?

Nástroj pro ladění (1h)

- Představení všech cílových skupin, aby bylo jasné, že cílových skupin je více.
- Jaké jsou důvody vytvoření pro vytvoření nástroje pro ladění ze strany vývojového týmu DMEE/DMEEX.
- Jaký je očekávaný přínos pro cílové skupiny.
- **Diskuze nad častými problémy při ladění a odhalování chyb v DMEE stromech.**
  - Časté problémy při odhalování chyb.
  - Úprava prvotního návrhu podle požadavků.

Definování požadavků na základě častých problémů (1h)

V této fázi jsem přistoupil k jasné definici požadavků na základě problémů, se kterými se cílové skupiny často setkávají. Byly zapisovány na virtuální tabuli, kterou mohli vidět i účastníci připojeni distančně.

4.5.1 Persóna vývojáře v oblasti plateb

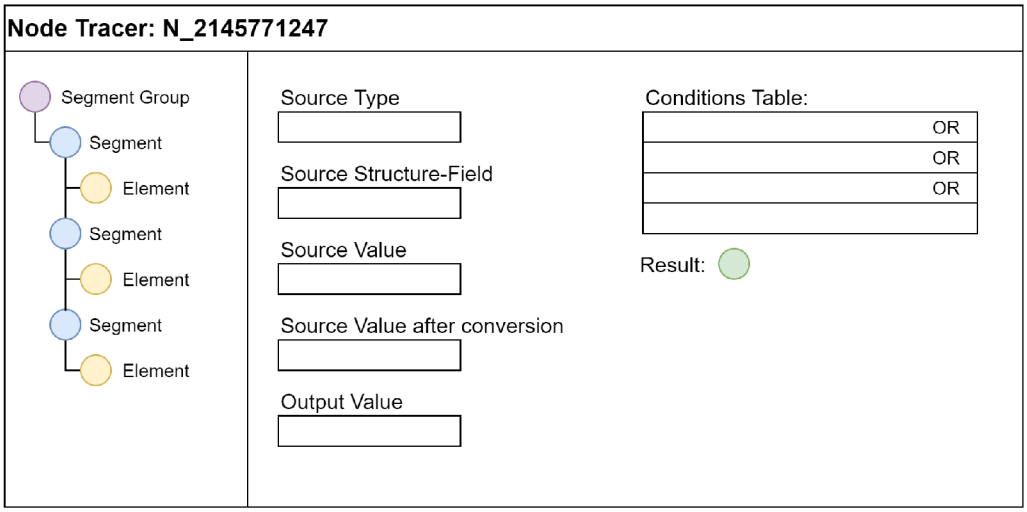
Průměrná zkušenost se systémem SAP u vývojářů činila 15 let. Konkrétněji vývojářská činnost v oblasti plateb průměrně 5 let. Tým těchto vývojářů zajišťuje lokalizaci platebních formátů pro více než 25 zemí světa.

V transakcích DMEE/DMEEX vytvářejí celé stromy a jejich verze specifické pro jednotlivé země. A to v rámci požadavku na lokalizaci, který se objevuje několikrát za rok.

Každodenně však DMEE/DMEEX používají při podpoře zákazníků resp. řešení zákaznicích incidentů.

Z diskuze nad požadavky ze strany vývojářů vyplynulo, že navržený nástroj by skutečně výrazně urychlil analýzu celé řady problémů, především těch, které byly popsány v kapitole 4.2. Nejdůležitějším výstupem této diskuze byl požadavek na zobrazení hodnot na uzlu v kontextu přímých předků tohoto uzlu. Pro lepší představu uvedu příklad. Mějme uzel typu element, který chceme analyzovat nástrojem pro ladění z mého prvotního návrhu. V případě, že některý z přímých předků tohoto uzlu obsahuje podmínku, která není splněna, tento uzel typu element se nebude zpracovávat. Uživatel tedy vůbec nemá přehled o tom, z jakého důvodu se tak stalo, protože o předcích nástroj nezaznamenává žádné informace. Uživatel by tedy měl mít možnost označit uzel pro sledování podobným způsobem jako je tomu u bodu přerušení, avšak s tím rozdílem, že v prvním případě by se automaticky zaznamenávaly i všichni přímí potomci sledovaného uzlu.

Dále padla shoda, že funkcionalita bodu přerušení by měla zůstat netknutá. Z toho důvodu bude výstup z nástroje pro ladění dostupný nikoli při zpracování uzlu, ale až po dokončení zpracování celého stromu. Výstup bude reprezentován graficky, podobně jako v původním návrhu s tím rozdílem, že na levé straně budou zobrazeny uzly v pořadí v jakém byly zpracovány. Po připomínkování se návrh změnil do podoby na obr. 4.10.



Obrázek 4.9: Návrh po připomínkování vývojářů

4.5.2 Persóna pracovníka produktové podpory

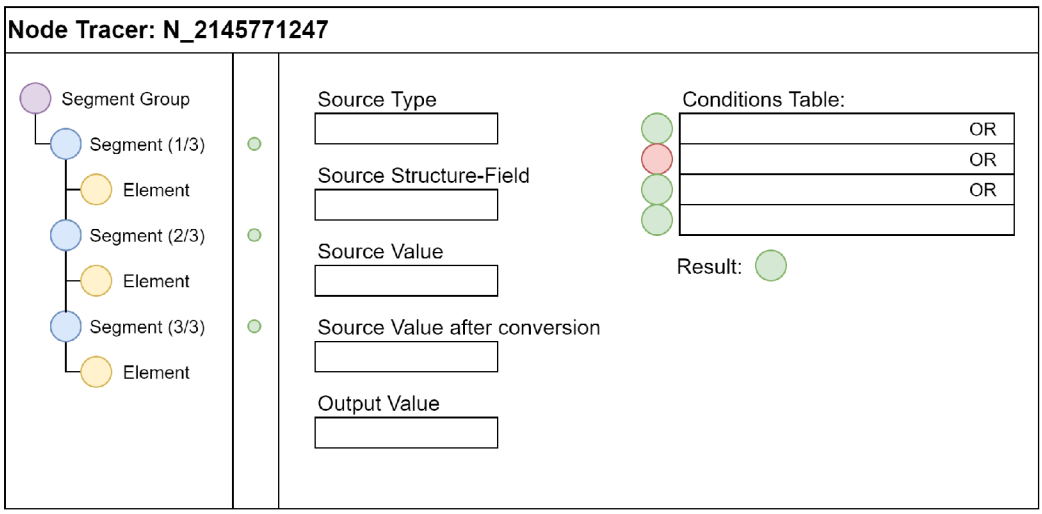
Průměrná zkušenost zúčastněných pracovníků produktové podpory se systémem SAP činila 7 let, z byly toho průměrně 3 roky podpory komponent souvisejících s platbami v systému SAP.

Mezi úkony v transakcích DMEE/DMEEX, které pracovnící produktové podpory vykonávají při řešení zákaznických incidentů, patří následujicí:

- Kontrola syntaktické správnosti stromu.
- Vyhledání a analýza nastavení uzlů.
- Úprava stromu na základě požadavků zákazníka.
- Analýza příčin chybného výstupu.

Pracovníci produktové podpory pracují s transakcemi DMEE/DMEEX téměř denně. Stromy zásadně nevytvářejí, pouze modifikují stromy, které jsou relevantní k problému, který se zákazníkem řeší. Vlastní nástroj ani způsob ladění nemají. Často používají dostupný testovací nástroj standardně dodávaný jako součást DMEE/DMEEX, který však podle jejich slov postrádá integračně specifickou práci s daty, která je obsažena v PMW. Z toho důvodu jim nástroj v mnoha případech nepomůže odhalit problém, který hledají. Z deseti zúčastněných pouze jeden disponuje znalostí programovacího jazyku ABAP, avšak z mého pohledu nedostatečnou pro hledání chyb laděním zdrojového kódu enginu DMEE/DMEEX. Předpoklady z podkapitoly 4.3 se tímto potvrdily.

Pracovníkům produktové podpory byl poté předložen návrh nástroje ve stavu po připomínkování od vývojářů. Doplnili, že by bylo vhodné, aby byl výsledek podmínky jednoduše viditelný ve struktuře uzlů na levé straně aplikace s tím, že by tato možnost výrazně urychlila orientaci v podmínkách. Dále bylo navrženo, číslovat iterace jednotlivých uzlů pro zlepšení orientace ve stromě. Zároveň by uvítali možnost exportování a importování záznamu, pro případnou zálohu a budoucí použití. Upravený návrh je na obrázku níže.



Obrázek 4.10: Návrh po připomínkování vývojářů

4.5.3 Persóna zákazníka znalého oblasti

Vzhledem k omezenému počtu účastníků a návrhu, který byl už připomínkovaný dvěma předchozími skupinami, nevzešlo z jejich strany žádné konkrétní vylepšení návrhu. Co je však důležité, všichni tři zástupci se shodli, že současné možnosti testování, ladění a ověřování stromů nejsou dostačující. V návrhu, který jsem předložil, všichni shledali smysl a vnímají ho, jako velmi dobrý příslib do budoucna.

Mimo samotného nástroje vzešlo z meetingu několik zajímavých návrhů na vylepšení transakcí DMEE/DMEEX. Meeting považuji na přínosný, byť samotný návrh nástroje pro ladění nijak neovlivnil.

4.6 Zvolená metodika vývoje

Proces návrhu v případě nástroje pro ladění nekončí začátkem jeho vývoje a to vzhledem k použité agilní metodice Scrum. Zvolená metodika vychází ze standardního nastavení vývojových týmů ve společnosti SAP.

Scrum je iterativní metodika vývoje software. Vývoj produktu je rozdělen do časově pevně daných bloků - sprintů [1]. Na konci každého sprintu je dodáván tzv. inkrement produktu, tedy funkcionalita, která má pro zákazníka reálnou hodnotu a použitelnost. Metodika Scrum počítá s úzkou spoluprácí vývojáře nebo vývojového týmu a zákazníka. Důležitým pojmem metodiky Scrum je tzv. Product Backlog. Product Backlog obsahuje všechny dosud známé požadavky na produkt, které musí být dodány. O důležitosti každé z položek rozhoduje zejména Product Owner 4.6

Sběr požadavků a zpětná vazba jsou pravidelné události probíhající kontinuálně. Jde o naprostou podstatu této metodiky, což dokazuje to, že pro tyto účely jsou v samotné definici Scrumu definovány následující události periodicky se opakující v každém sprintu:

- Review
- Planning
- Retrospective

Review se koná vždy na konci sprintu. Vývojový tým představuje zákazníkům inkrement produktu. Tým díky tomu může získat včasnou zpětnou vazbu na dosavadní vývoj produktu. Běžně se stává, že zákazníci si vyvíjenou funkcionalitu představují jinak popř. se priority zákazníků změní. Vývojový tým tak může na tyto okolnosti pružně zareagovat a další sprint podle toho přizpůsobit.

Na planningu se vývojový tým na základě požadavků z backlogu a jejich priorit rozhoduje, které z těchto požadavků bude schopný dodat jako inkrement produktu v následujícím sprintu. Plánuje se kapacita jednotlivých členů týmu a zvažují se i možná rizika jednotlivých požadavků.

Retrospective neovlivňuje přímo zákazníky. Slouží týmu jako nástroj pro kontinuální zlepšování sebe sama. Hodnotí se, jak tým pracoval minulý sprint, kde má tým slabá místa, kde a z jakého důvodu nastaly problémy, jak podobným problémů předcházet a další interní otázky. Zdůrazňují se ale i úspěchy týmu, které by bylo vhodné přenést i do následujících sprintů.

Metodika Scrum stanovuje v týmu tři základní role:

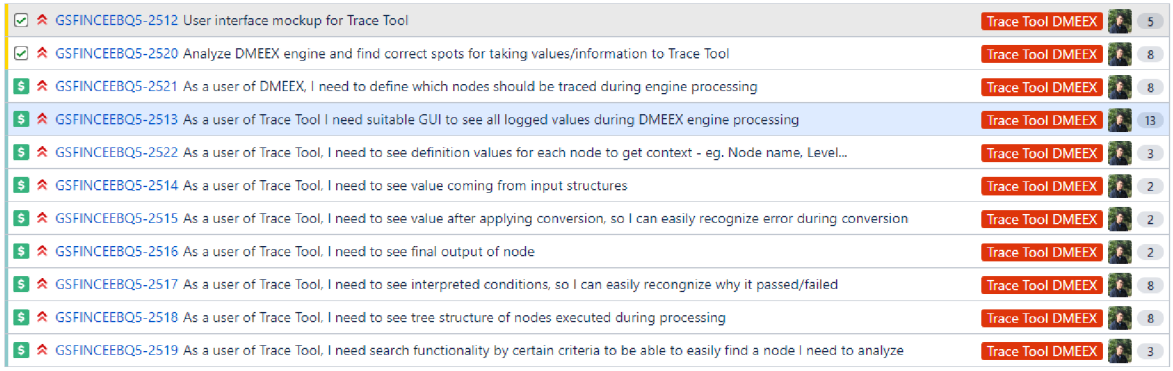
[1] Metodika Scrum používá specifickou anglickou terminologii, která je v českých textech běžně zaužívaná, proto ji používám také.

- Product Owner - zodpovídá za komunikaci se zákazníky, sběr požadavků, prioritizaci backlogu
- Vývojový tým - vyvíjí inkrement produktu
- Scrum Master - zodpovídá za správnou aplikaci metodiky scrum, odbourává překážky ve vývoji

Jako vývojář jsem součástí týmu 8 dalších spolupracovníků, vývojářů a testerů. Tým je zodpovědný za aplikace DMEE, DMEEX a další nástroje pro mapování z portfolia společnosti SAP. V rámci vývoje nástroje pro ladění jsem byl tomuto projektu vyhrazen pouze já (vyjma manuálního testování). Zároveň jsem pro tento konkrétní projekt částečně zastupoval Product Ownera v rámci komunikace se zákazníky a sběru požadavků a definice produktu.

4.7 Definice produktu

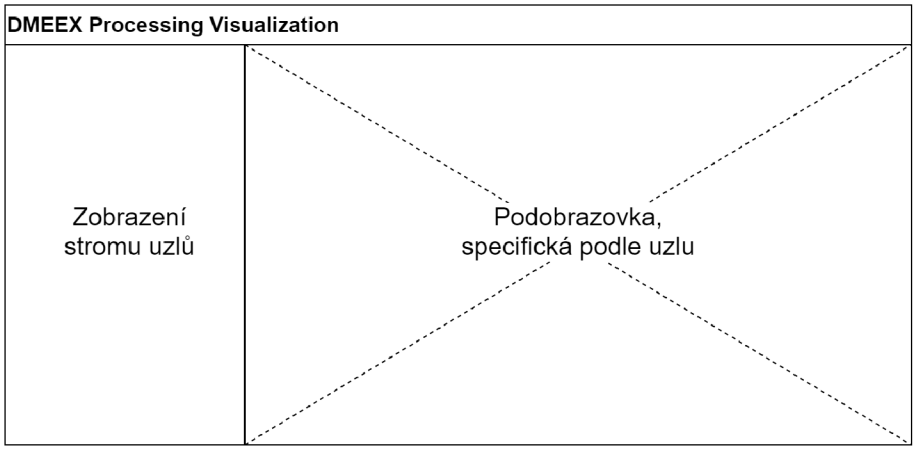
Grafický návrh prezentovaný cílovým skupinám musel být rozšířen o všechny typy uzlů, další podobrazovky apod. Dále jsem na základě získaných znalostí definoval požadavky na produkt v podobě položek backlogu. Vzhledem k inkrementálnímu vývoji je bylo nutno definovat po malých logických částech, kde každá položka přináší přidanou hodnotu uživateli. Počítá se se skutečností, že položky se budou v průběhu času měnit, konkretizovat se. Backlog ve stavu před začátkem vývoje je zachycen na obr. 4.11



Obrázek 4.11: Backlog pro nástroj pro ladění. Na obrázku je zachycen stav před začátkem vývoje především z produktového, nikoli technického hlediska.

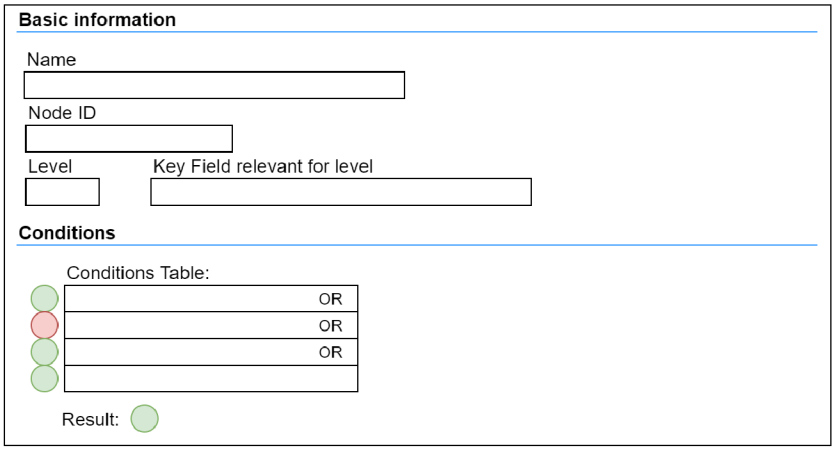
4.7.1 Finalizace návrhu

Návrh představovaný cílovým skupinám jsem účelově příliš nekonkretizoval, aby nebyly ovlivněny mými představami o tom, co by nástroj pro ladění měl umět a jak by měl vypadat. Musel být tedy před začátkem vývoje upřesněn. Jak již víme, GUI se podle návrhu skládá ze dvou základních částí - vlevo je prostor pro zobrazení uzlů stromu, které jsou zpracovávány v rámci běhu DMEEX, vpravo pak prostor, který by měl zobrazovat uživateli data relevantní pro zvolený uzel, viz obr. 4.12.



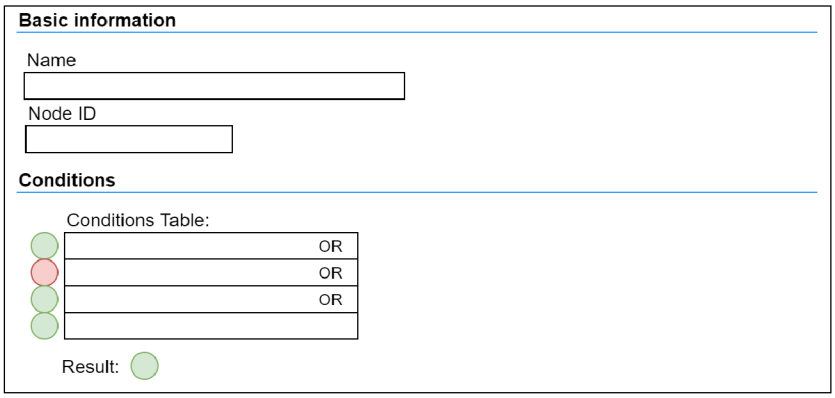
Obrázek 4.12: GUI nástroje zahrnuje dvě klíčové části - prostor pro zobrazení stromu uzlů (vlevo) a podobrazovku relevantní pro každý uzel.

Každý uzel v DMEEX má svoje specifika, která je potřeba zohledňovat z hlediska hodnot, které se budou v nástroji pro ladění zobrazovat. Na obr. 4.13, 4.14 a 4.15 můžeme vidět návrhy polí obrazovek pro uzly typu segment group, segment, composite, element, technický uzel, XML attribute a atom. Každá podobrazovka obsahuje blok *Basic information*, který by měl poskytovat téměř vždy údaje převzaté z definice stromu resp. jeho uzlů. Jedná se o hodnoty jako jméno, různé identifikátory, nastavení délek, ofsetů apod. Blok *Conditions*, udávající výsledek podmínek, je obsažen v podobrazovkách všech typů uzlů. Pro uzly zajišťující mapování se objevuje blok *Mapping*, který by uživateli měl předkládat všechny důležité informace od vstupní hodnoty přebírané ze zdroje (jiný uzel, konstanta, pole struktury apod.), přes konverzi, až po zobrazení výstupní hodnoty.

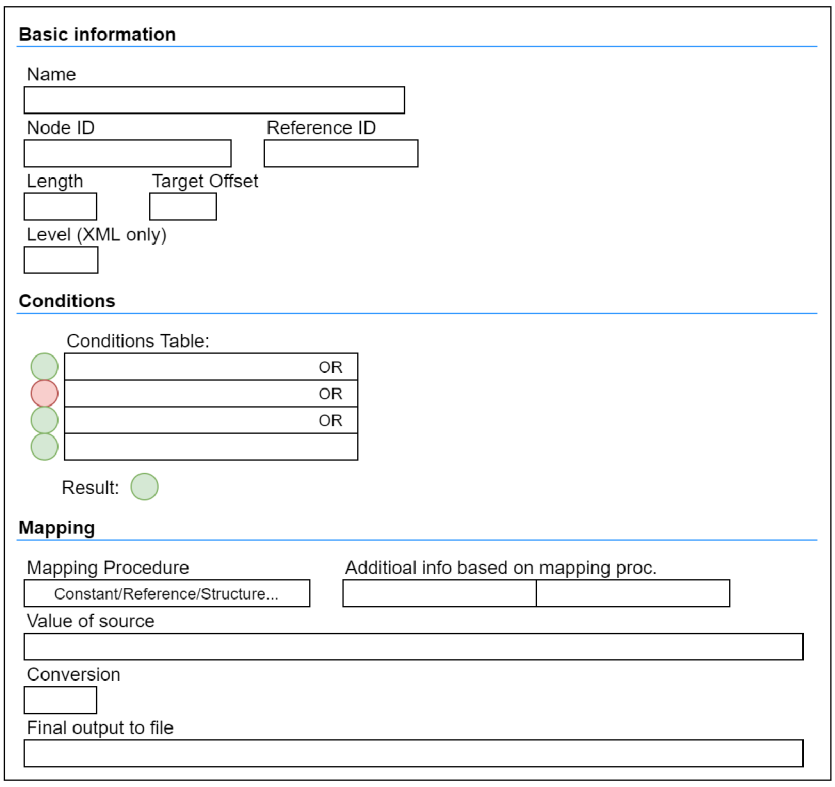


Obrázek 4.13: Návrh podobrazovky pro uzly typů *segment group* a *segment* s relevantními poli.

Úvodní obrazovka nástroje pro ladění (obr. 4.16) by měla zobrazovat obecné údaje o běhu stromu a dále statistická data, kterými jsou např. doba běhu a počet zpracovaných uzlů. Tento počet udává všechny uzly stromu, nikoliv pouze ty, které jsou označeny „pro ladění“.



Obrázek 4.14: Návrh podobrazovky pro uzel typu *composite* s relevantními poli.

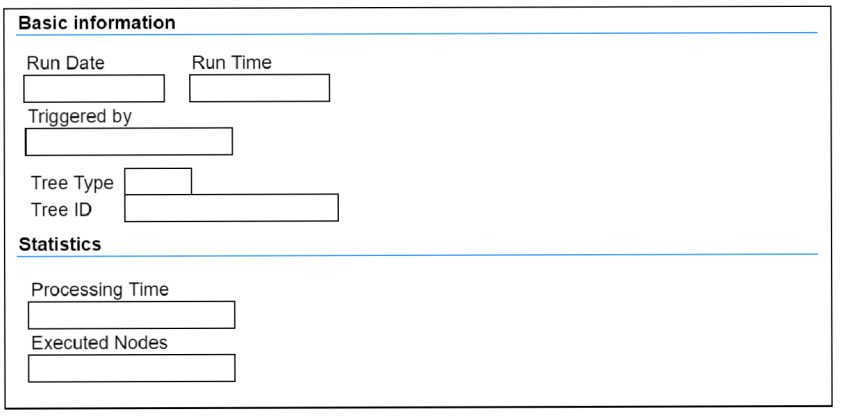


Obrázek 4.15: Návrh podobrazovky pro uzly typů *element*, *technický uzel*, *XML atribut* a *atom* s relevantními poli.

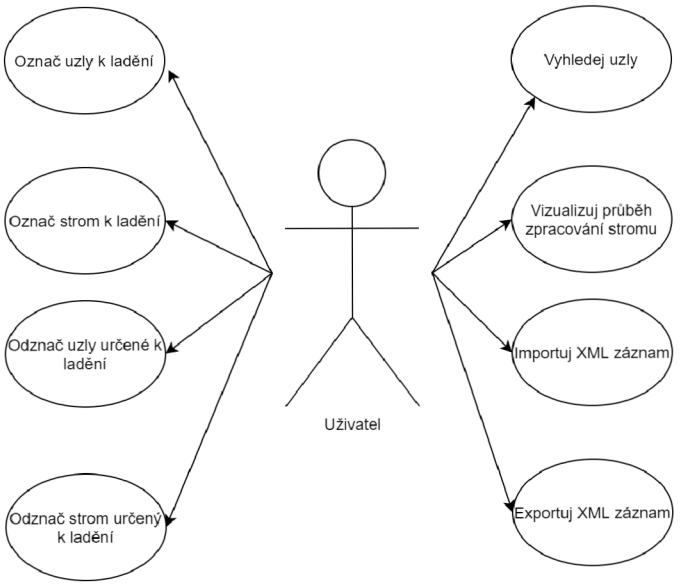
Model případů použití

Je užitečné, aby byl součástí také diagram případů použití, vytvořený se zohledněním požadavků od cílových skupin, viz obr. 4.17.

Uživatel označuje jednotlivé uzly a vybírá je tak k ladění. Stejným způsobem může tyto uzly odznačit. Uživatel také může vybrat k ladění i celý strom, tedy všechny jeho uzly resp. celý strom odznačit. Uživatel vizualizuje průběh zpracování stromu, v rámci kterého může vyhledávat uzly na základě různých kritérií. Vizualizaci zpracování může exportovat a opětovně importovat.



Obrázek 4.16: Návrh úvodní podobrazovky nástroje pro ladění.



Obrázek 4.17: Diagram případů použití nástroje pro ladění.

4.8 Zvolené technologie pro implementaci

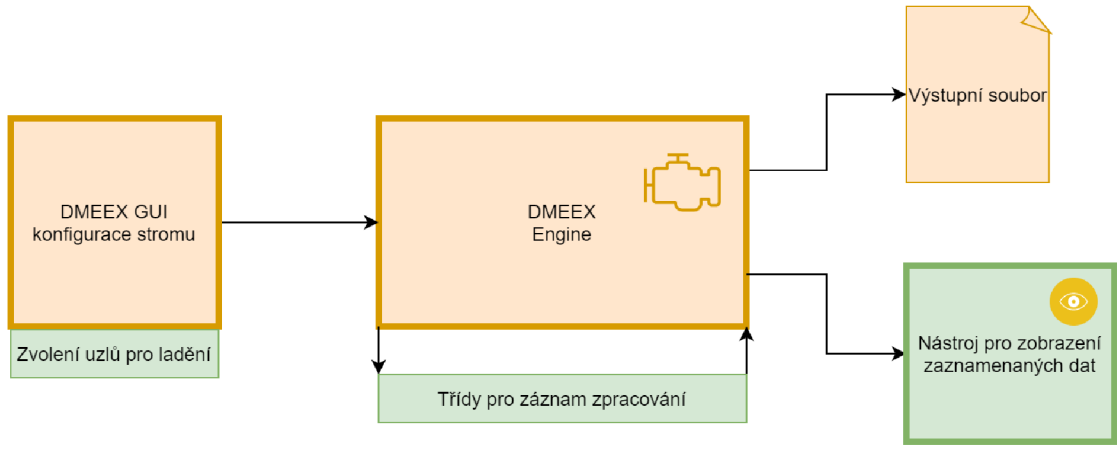
Aplikace pro ladění je v této fázi určena pro interní nasazení na on-premise instalacích. Nejprve na platformu S/4HANA (DMEEX), později se počítá s migrací nástroje na platformu R/3 (DMEE). Z důvodu usnadnění migrace jsem zvolil uživatelské rozhraní kompatibilní se SAP GUI, nikoliv Fiori, protože starší verze platformy R/3 Fiori vůbec nepodporují. Jako programovací jazyk byl přirozeně zvolen pro platformy nativní ABAP.

Pro implementaci jsou používal výhradně nástroje ze skupiny ABAP Workbench, který sdružuje různé vývojářské nástroje na platformě SAP a o němž jsem se rozepsal v kapitole 1.

4.9 Implementace

Implementace aplikace pro ladění probíhala ve dvoutýdenních sprintech. Vzhledem k použité metodice Scrum došlo v průběhu vývoje k průběžným úpravám vstupních požadavků. Všechny změny budou zmíněny jako celek na konci této kapitoly.

Implementace nástroje pro ladění se skládá ze tří velkých celků. Prvním z nich je integrace nástroje pro ladění do stávajícího řešení DMEEX tak, aby uživatel mohl na základě potřeb volit uzly relevantní pro ladění. Druhou významnou součástí je třídní model zajišťující sběr dat v průběhu zpracování stromu v DMEEX a integrace metod těchto tříd do existujícího kódu DMEEX Enginu. Celý třídní model je navržen jak, aby integrace jeho metod co nejméně narušila původní kód DMEEX Enginu. Posledním celkem je nástroj zajišťující vizualizaci dat, které byly zachyceny v třídním modelu, podle návrhu z předchozí kapitoly. Architektura a integrace do stávajícího řešení je zachycena na obr. 4.18.



Obrázek 4.18: Zelené části obrázku představují nové součásti, implementované v rámci práce na nástroji pro ladění.

4.9.1 Rozšíření uživatelského rozhraní DMEEX

S přihlédnutím k uvedené architektuře jsem implementaci nástroje započal integrací možnosti označení uzlů pro ladění ve stávajícím uživatelském rozhraní DMEEX. Zde jsem se přesně snažil kopírovat požadavky od cílových skupin. Implementována byla možnost označit si celý strom nebo jednotlivé uzly s automatickým označením cesty (neboli všech přímých předků).

V souvislosti s tím jsem musel vytvořil jednoduchý datový model, pro ukládání označených uzlů. V zásadě jde o jednu tabulku obsahující typ stromu, ID stromu, ID uzlu, jméno uživatele, který uzel označil a příznak, zda je uzel označen přímo uživatelem nebo je dopočítán na základě cesty k uživatelem označenému uzlu. Pro speciální případ, kdy uživatel označí celý strom (resp. všechny jeho uzly) jako relevantní pro ladění, používám ID uzlu „ROOT“ jako identifikátor této volby. Právě kvůli potřebě tohoto datového modelu v dalších fázích vývoje jsem se rozhodl začít úpravou GUI DMEEX.

Z hlediska UX (user experience) jsem zde narazil na problém, jak vyřešit případnou kolizní situaci, kdy se uživatel zvolí uzel jako relevantní pro ladění a zároveň si na stejném uzlu nastaví i bod přerušení. Rozhodl jsem se proto přesunout ikony reprezentující tyto volby za jméno uzlu, viz obr.

Z technického hlediska se v tomto případě jednalo u drobné úpravy chování GUI komponenty `CL_GUI_LIST_TREE` a to tak, aby zobrazovala ikony i za jménem uzlu, tak jak je zachyceno na obr.

Dále bylo potřeba změnit kontextovou nabídku zobrazovanou po kliknutí pravého tlačítka na určitý uzel. Zde přibyla položka nastavení uzlu „pro ladění“.

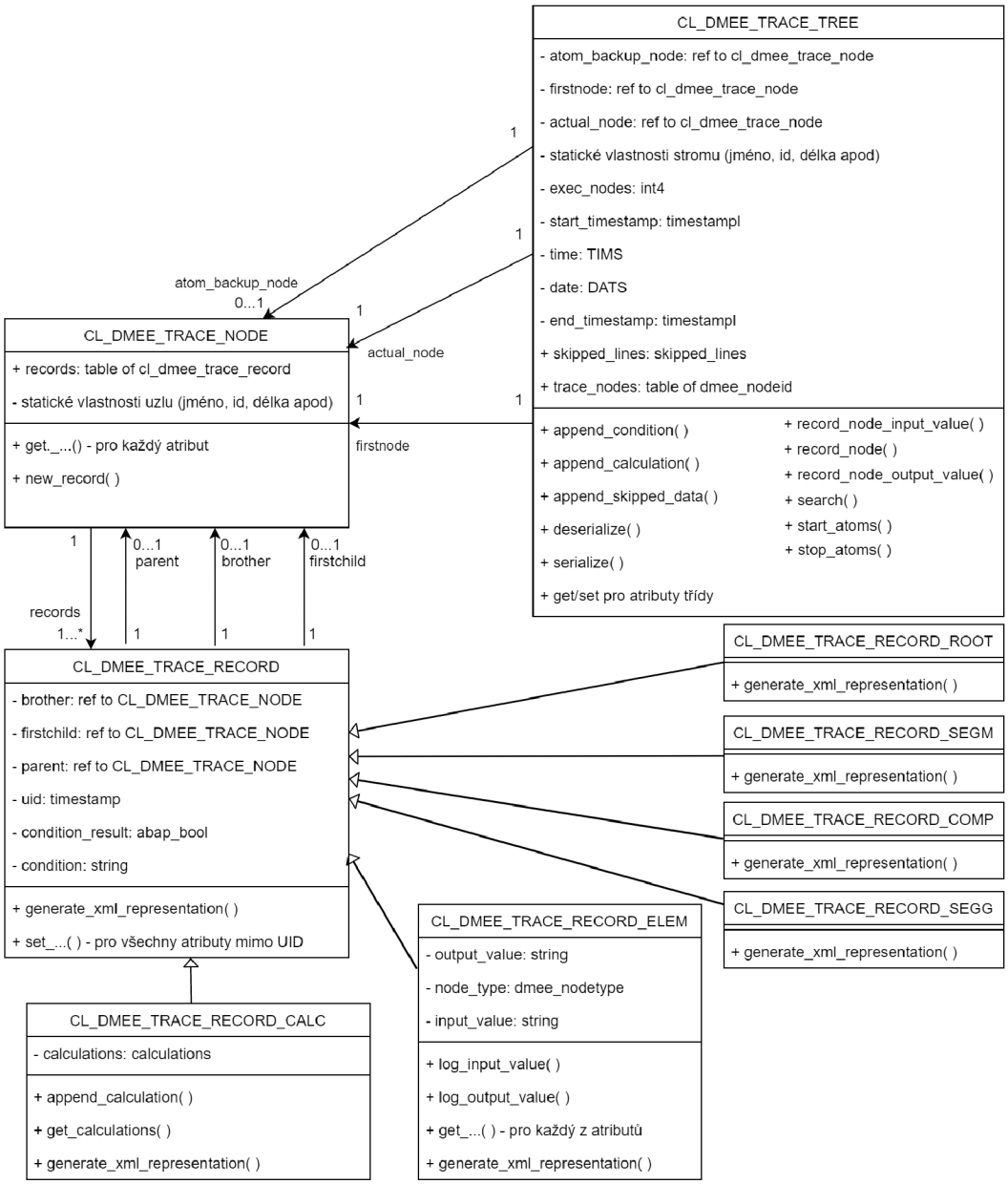
4.9.2 Třídy pro záznam zpracování a jejich integrace

Základem funkcionality pro záznam a uchování hodnot ze zpracování stromu pomocí DMEEX engine je skupina tříd, které zajišťují sběr statických dat (jména uzlů, ID uzlů apod.), dynamických dat (vstupní, výstupní hodnota, podmínky apod.) a korektní vytváření hierarchie zpracování uzlů. Třída `CL_DMEE_TRACE_TREE` implementuje vytvoření struktury zaznamenaných dat za pomocí dalších tříd a pomocí svých metod vytváří rozhraní pro programátora, které umožňuje integraci do DMEEX Engine. Zaznamenávané hodnoty jsou ze dvou kategorií. První kategorie jsou statická data, která se v průběhu zpracování nemění. Typicky jsou to data, která jsou součástí definice stromu např. ID uzlu, délka, typ mapování apod. Druhou kategorií jsou data dynamická, která se pro každý průchod uzlem obvykle liší, jako např. zdrojová hodnota, výstupní hodnota, výsledek podmínky apod. Rozdělil jsem statická a dynamická data do dvou tříd. `CL_DMEE_TRACE_NODE` implementuje prostředky pro uchování statických dat, `CL_DMEE_TRACE_RECORD` pak spravuje data dynamická. Pro každý typ uzlu je implementována specifická třída, kde `CL_DMEE_TRACE_RECORD` je třídou rodičovskou. Pro lepší ilustraci viz obr. 4.19.

Volání konstruktoru třídy `CL_DMEE_TRACE_TREE` jsem zaintegroval do funkčního modulu `DMEE_START`. Po instanciaci třídy jsou do paměti načteny všechny uzly právě zpracovávaného stromu, které jsou označeny pro ladění. Úspěšné vytvoření instance je také prvkem, který rozhoduje o tom, zda se budou zpracovávat rozšíření DMEEX Enginu. Díky tomuto kroku prakticky zaniká vliv nástroje pro ladění na rychlost či paměťovou náročnost zpracování v případě, že žádný uzel pro ladění není zvolen. V rámci volání konstruktoru se také nastavují časové značky, sloužící k pozdějšímu výpočtu doby zpracování stromu.

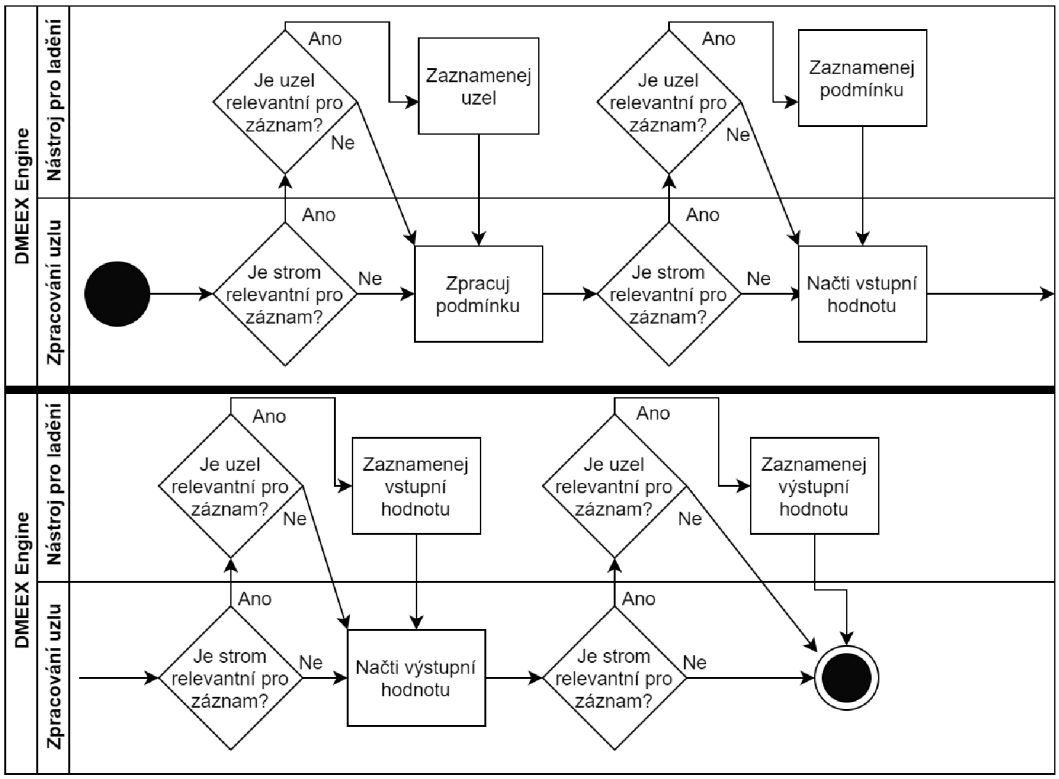
Metody třídy `CL_DMEE_TRACE_TREE` jsou poté integrovány v rámci zpracování jednoho řádku vstupních struktur v rámci funkčního modulu `DMEE_PUT_ITEM`, vždy na patřičných místech v rámci kódu DMEEX Enginu - začátek zpracování uzlu, vyhodnocování podmínek, načtení vstupní hodnoty, zpracování konverze apod.

Metody třídy `CL_DMEE_TRACE_TREE` zajišťují nejen správné uložení hodnot, ale i posloupnosti zpracování uzlů se zachováním jejich hierarchie v rámci stromu. Jako příklad formou diagramu aktivity uvádím zpracování uzlu element (bez atomů), viz obr. 4.20.



Obrázek 4.19: Diagram tříd. `CL_DMEE_TRACE_TREE` obsahuje odkaz na první uzel záznamu reprezentovaný třídou `CL_DMEE_TRACE_NODE`, která může obsahovat n záznamů pro daný uzel. Tento záznam je reprezentovaný třídou `CL_DMEE_TRACE_RECORD` resp. jednou ze tříd, které z ní dědí. Součástí záznamu je i informace o poloze uzlu ve stromě. Diagram je kvůli úspoře místa a přehlednosti zjednodušený - neobsahuje jmenovitě všechny metody, atributy nebo jejich rozhraní.

Při integraci jsem si kladl za cíl minimalizovat zásahy do již existujícího kódu DMEEX Engine, především z důvodu přehlednosti a určité logiky. Rád bych z diagramu 4.19 výše vypíchl metodu `record_node` třídy `CL_DMEE_TRACE_TREE`, která hraje klíčovou roli ve vytváření záznamu zpracování stromu. Je integrována na přesná místa v DMEEX Enginu – začátek zpracování určitého uzlu a implementuje logiku nalezení správné pozice v záznamu



Obrázek 4.20: Diagram aktivity pro uzel typu element bez atomů.

pro právě zpracovávaný uzel. Základem tohoto záznamu jsou uzly reprezentované jako instance třídy `CL_DMEE_TRACE_NODE`. Tato třída, jak je možné vidět z diagramu, obsahuje seznam odkazů na `CL_DMEE_TRACE_RECORD`. Tento seznam je naprosto klíčový pro zaznamenání cyklů pod určitým uzlem - počet zaznamenaných iterací určitého uzlu koresponduje s počtem záznamů z tomto seznamu. `CL_DMEE_TRACE_RECORD` je potom zasazen do hierarchie pomocí odkazů na další instance třídy `CL_DMEE_TRACE_NODE` (atributy `brother_node`, `firstchild_node` a `parent_node`).

K ukončení záznamu dochází ve funkčním modulu `DMEE_END`, kde se na základě relevance pro daný strom rozhoduje o zavolání nástroje pro vizualizaci zaznamenaných hodnot. Zde jsem narazil na poměrně limitující problém, protože jazyk ABAP nenabízí možnost předání reference jinému programu. Vzhledem k tomu, že nástroj pro ladění resp. jeho vizualizační část musí být samostatným programem, musel jsem najít jinou cestu, jak dosáhnout korektního předání instanciované třídy `CL_DMEE_TRACE_TREE`. První z možností byl koncept sdílené paměti, který ovšem nelze použít pro hluboké struktury, kterou záznam je.

Nakonec jsem zvolil cestu serializace a deserializace. Serializace je proces, který umožňuje konverzi složitých zanořených struktur do sériové podoby - sekvence bajtů. Díky tomu je možné tyto struktury snadno přenášet v různých programových prostředích nebo přes internet.

K serializaci dochází, jak již bylo nastíněno, ve funkčním modulu `DMEE_END`, ihned poté se asynchronně a v novém okně zobrazí vizualizační program nástroje pro ladění. Serializovaná podoba je uložena do databázové tabulky k tomu určené. Obsahuje pouze klíč a onen záznam.

4.9.3 Vizualizační program

Vizualizační program je implementován jako standardní program jazyka ABAP. Umožňuje uživateli graficky zobrazit hodnoty a uzly zaznamenané v průběhu zpracování stromu. Skládá se z hlavního Dynpra, které zobrazuje uzly v zaznamenané hierarchii, a několika podobrazovek, které přísluší jednotlivým typům uzlu. Dále je v rámci vizualizačního programu implementována podpora vyhledávání na základě kombinace hodnot ID uzlu, názvu uzlu, vstupní a výstupní hodnoty uzlu viz tabulka 4.2.

Dynpro číslo	**Popis**	**Typ**
0100	Oblast pro komponentu stromu, oblast pro subscreen	Screen
0210	Subscreen pro zobrazení obecných informací o zpracování stromu	Subscreen
0230	Subscreen pro uzly typu segment, segment group, composite	Subscreen
0240	Subscreen pro uzly typu element, tech. uzel, XML attr., atom	Subscreen
0260	Subscreen pro uzly typu calculation node	Subscreen
0410	Dialogové okno vyhledávání	Dialog
0710	Dialog pro zobrazení dlouhých hodnot	Dialog

Tabulka 4.2: Tabulka představující jednotlivá Dynpra tvořící vizualizační program

Program pro vizualizaci je, jak již bylo napsáno volán asynchronně z funkčího modulu `DMEE_END`.

První volání, které provádí je načtení a deserializace záznamu z databázové tabulky. Tím se znovu vytvoří instance třídy `CL_DMEE_TRACE_TREE`.

Toto řešení, byť velmi neefektivní, jsem konzultoval i se svými kolegy a i přesto, že se jedná o zbytečný krok navíc, ani oni neměli lepší nápad, jak situaci vyřešit.

Vizualizace stromu

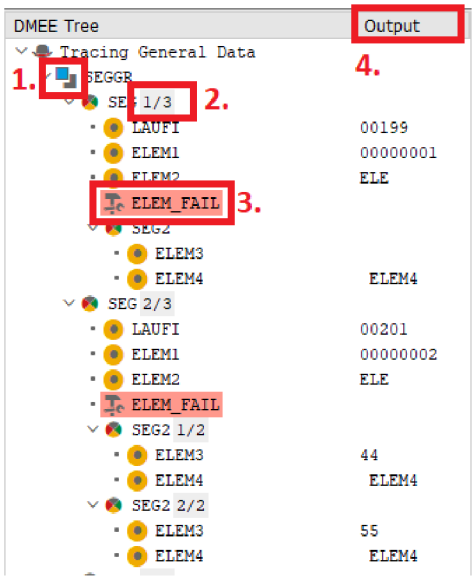
Pro vizualizaci stromu jsem zvolil standardní komponentu uživatelského rozhraní SAP `CL_GUI_LIST_TREE`. Ta je mimo jiné používána i v rámci DMEEX, takže volba byla o to snazší, kvůli zachování stejné uživatelské zkušenosti.

`CL_GUI_LIST_TREE` umožňuje široké spektrum nastavení, které jsem v rámci implementace potřeboval a využil. Jedná se např. o zobrazení různých druhů ikon, zobrazení dodatečných informací o každém uzlu, možnost podbarvení či zvýraznění konkrétního uzlu a mnohé další. viz obr. 4.21.

Tento GUI strom je zároveň základním ovládacím prvkem celého programu. Slouží totiž k přepínání kontextu mezi jednotlivými uzly. Při kliknutí na konkrétní uzel se vyvolá událost, která vede ke změně kontextu a překreslení podobrazovky pro zvolený typ uzlu a načtení relevantních dat.

Při implementaci jsem narazil na problém, se kterým jsem nepočítal při implementaci tříd pro záznam zpracování stromu. Uzly, které jsou vidět např. na obrázku 4.21, odpovídají hierarchicky záznamům třídy `CL_DMEE_TRACE_RECORD`. Avšak abych mohl jednoznačně provázat záznam třídy `CL_DMEE_TRACE_RECORD` s GUI uzlem komponenty `CL_GUI_LIST_TREE` musel jsem dodatečně ve třídě `CL_DMEE_TRACE_RECORD` doplnit unikátní ID pro každý záznam. ID se přiřazuje při vytvoření instance třídy `CL_DMEE_TRACE_RECORD` a získává se metodou `get_uid()`.

Do této chvíle žádný unikátní identifikátor jednoho záznamu neexistoval. ID uzlu jsem použít nemohl, protože se uzly mohou v rámci iterací opakovat.



Obrázek 4.21: 1. Ikony jsou převzaté z DMEEX, 2. Počítadlo iterací, 3. Červené podbarvení uzlu s nesplněnou podmínkou, 4. Sloupec „Output“ zobrazuje výstupní hodnoty uzlu

Implementace obrazovek Dynpro se skládá ze dvou základních částí. První fáze je nakreslení obrazovek, podobrazovek a dialogových oken. K tomu slouží integrovaný nástroj ze skupiny ABAP Workbench, který se nazývá Layout Editor. Umožňuje uživateli definovat aktivní plochu obrazovky, vložit do ní textová pole, tabulky, oblasti pro podobrazovky, štítky a další standardní prvky uživatelského rozhraní.

Jak bylo napsáno v kapitole 1, obrazovky jsou obsluhovány pomocí tzv. „flow logic“, skládající se ze dvou základních událostí: PBO a PAI. Druhým krokem implementace vizualizačního programu tedy bylo naprogramovat obsluhu obrazovek a všech prvků uživatelského rozhraní.

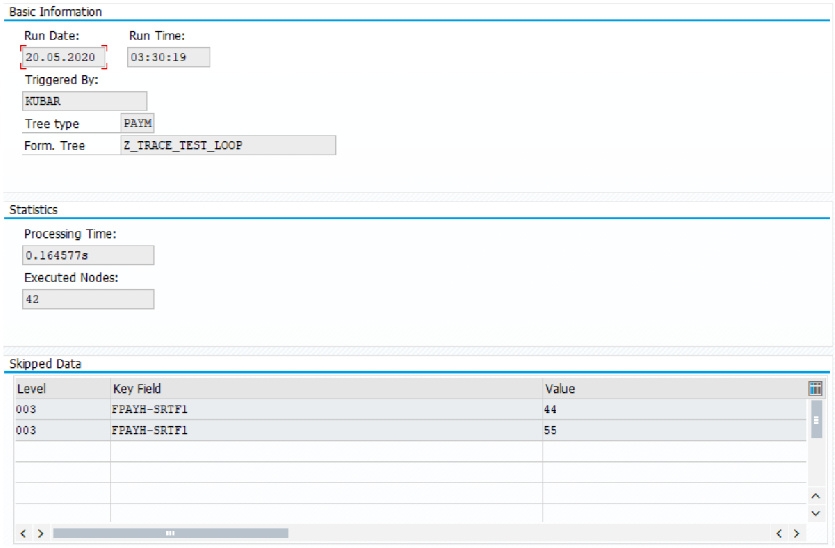
Vzhledem k tomu, že Vizualizační nástroj slouží pouze pro zobrazení zachycených hodnot, událost PAI (proces after input) pro jeho implementaci nesehrála až na výjimky, které rozeberu níže, významnou roli. Všechny obrazovky nebo podobrazovky Dynpro mají společné načítání dat z instance třídy `CL_DMEE_TRACE_TREE` resp. `CL_DMEE_TRACE_NODE` a `CL_DMEE_TRACE_RECORD` v rámci události PBO do příslušných polí uživatelského rozhraní.

Nyní se zaměřím na popis podobrazovek jednotlivých typů uzlů.

Podobrazovka pro kořen stromu

Podobrazovka kořene stromu, viz obr. 4.22, v zásadě vychází z finálního návrhu 4.16. Zobrazuje hodnoty z instance třídy `CL_DMEE_TRACE_TREE`.

V průběhu vývoje jsem však studiem zdrojových kódů DMEEX Engine zjistil, že jeden z problémů, který má nástroj pomoci odhalovat, konkrétně šlo o hledání příčin chybějícího bloku ve výstupním souboru, nebude možné v současném návrhu nijak zobrazit. O přeskočení úrovně na základě zopakovaní klíče se totiž rozhoduje (a zpětným pohledem zcela logicky) ještě před zpracováním uzlů dané úrovně. Doimplementoval jsem atribut `skipped_data`, metody `append_skipped_data( )` a `get_skipped_data( )`, které slouží k podpože tohoto typu záznamu a integroval je na patřičné místo v DMEEX Engine. Přeskočená data se nyní zobrazují v tabulce v sekci „Skipped Data“ a obsahuje údaj o úrovni, klíči a hodnotě klíče.



Obrázek 4.22: Podobrazovka pro kořenový uzel včetně tabulky zobrazující přeskočená data na základě shody klíče.

Podobrazovka pro element, technický uzel, atom a XML atribut
Vzhledem ke shodě zaznamenávaných hodnot pro tyto uzly jsem bylo na místě použít jeden typ obrazovky pro všechny z nich. Zobrazované hodnoty opět vycházejí z návrhu s výjimkou zobrazování podmínek.

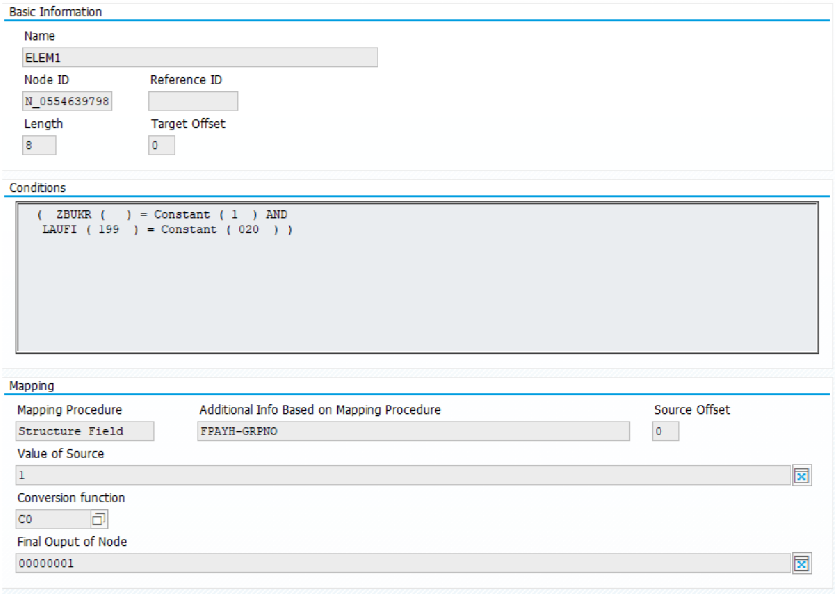
Při získávání požadavků od cílových skupin i při finalizaci návrhu všichni opomenuli, jakým způsobem jsou zpracovávány podmínky v DMEEX. O chybě došlo asi na základě toho, že podmínky jsou v původním, starém řešení DMEE zpracovávány řádek po řádku, tak jak jsou definovány. S tím bylo počítáno i ve všech návrzích. Realita je však taková, že v DMEEX může uživatel uzavírat jednotlivé logické výrazy do závorek. Zpracování zde tedy neprobíhá „řádek po řádku“, ale na základě postupného vyhodnocování a zjednodušování jednotlivých logických výrazů.

Z toho důvodu byla výsledná funkcionalita po konzultaci s vývojáři platebních formátů, upravena do jiné podoby. Interpretace podmínky je postupně zaznamenávána formou textového řetězce a ve formátované formě předložena uživateli na výstup do příslušného pole, viz obr. 4.23.

Podobrazovka pro segment, segment group a composite
Oproti návrhu bylo z finálního programu odstraněno pole „Key field relevant for level“, protože pozbylo potřeby po implementaci tabulky zobrazující přeskočené řádky na základě shody klíče, kterou jsem popisoval výše.

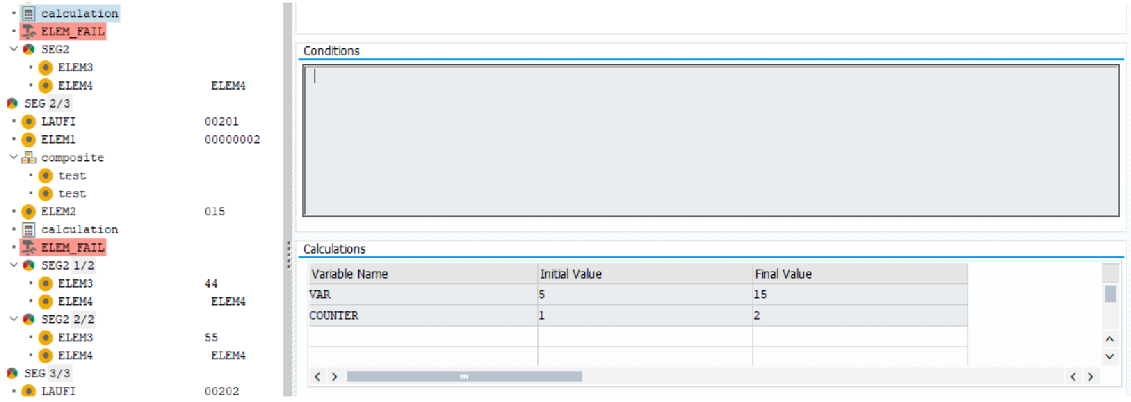
Podmínky jsou zde zastoupeny ve stejné formě jako u podobrazovky `0240` pro element apod.

Podobrazovka pro kalkulační uzel
O podobě záznamu kalkulačního uzlu se rozhodnuto až v posledních fázích vývoje nástroje po jedné ukázce týmu vývojářů. To je důvod, proč se o kalkulačním uzlu nikde ve finalizaci návrhu nezmiňuji. Podobrazovka pro kalkulační uzel zobrazuje, mimo již řečených polí, tabulku obsahující výpočty, které byly v rámci zpracování uzlu zaznamenány. Pro každou



Obrázek 4.23: Podobrazovka pro element. Mimo interpretované podmínky stojí za povšimnutí, jak nástroj zobrazuje aplikaci konverze na vstupní hodnotu. V tomto případě DMEEX Engine doplnil nuly před hodnotu až do maximální délky uzlu 8.

proměnnou je vyčleněn jeden řádek GUI tabulky obsahující její iniciální (před vstupem do této kalkulace) a hodnotu finální, viz obr. 4.24.



Obrázek 4.24: Zobrazení hodnot proměnných na kalkulačním uzlu.

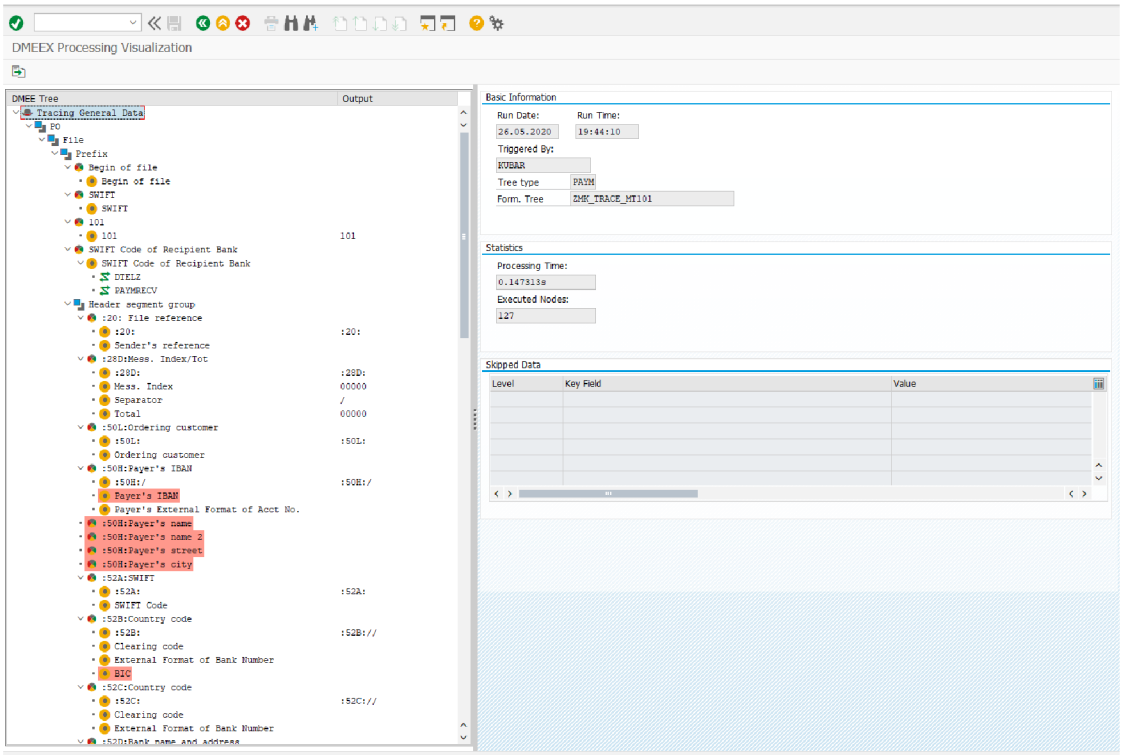
Vyhledávání

Funkcionalitou, která výrazně usnadňuje orientaci ve stromě je vyhledávání. Strom se prohledává do hloubky, uživatel má možnost specifikovat tři parametry vyhledávání: *ID uzlu*, *vstupní hodnotu* a *výstupní hodnotu*. Prohledávání začíná vždy od kořenového uzlu, tedy od atributu `firstnode` třídy `CL_DMEE_TRACE_TREE`, uzly se shodou s parametry vyhledávání se ukládají do interní (paměťové) tabulky. Ze záznamů v této interní tabulce poté nástroj pro vizualizaci označí první nalezený uzel v GUI. Další záznamy se využívají pro funkcionalitu „vyhledej další“.

Importování a exportování záznamu
Vizualizační nástroj umožňuje stáhnout aktuální záznam v podobě XML souboru. Využívá k tomu již zmíněné serializace a deserializace. Pro opětovné nahrání záznamu slouží program `ZKR_TRACE_RPRT_UPLOAD`.

V úvodu kapitoly jsem zmínil možnost využití nástroje na platformě SAP S/4HANA Cloud. Pro kompletní využití nástroje by bylo nutné implementovat samostatně GUI pro SAP Fiori. Díky možnosti importování a exportování by však nástroj pro ladění mohl být využíván pro podporu zákazníků ze strany společnosti SAP.

Finální produkt
Finální vzhled nástroje pro ladění lze vidět na obr. 4.25.



Obrázek 4.25: Finální vzhled nástroje pro ladění.

4.10 Testování nástroje pro ladění

Testování nástroje pro ladění kombinovalo několik různých přístupů. Vzhledem k použité metodice Scrum musel být nástroj, byť třeba zcela neúplný, na konci každého sprintu ve funkčním stavu. Jako příklad uvedu první sprint, kdy byla implementována funkční podpora pro označování uzlů v DMEEX a proběhla první základní integrace do DMEEX Enginu. Výstupem tehdy byl pouze XML soubor s identifikátory uzlů, které byly zpracovány. Pozůstatkem tohoto období je např. metoda `generate_xml_representation( )`, která je definovaná v `CL_DMEE_TRACE_RECORD` a zděděná všemi potomky. Nyní se už nevyužívá.

Aby bylo možné tento funkční inkrement produktu na konci každého sprintu garantovat, snažil jsem se udržovat jednotkové testy všech metod, které souvisely s tímto vývojem, každou novou funkcionalitu manuálně testovat a kontrolovat automatické regresní testy.

4.10.1 Jednotkové testování

Zcela základním přístupem bylo jednotkové testování (unit testy). Sám pro sebe jsem si kladl za cíl mít jednotkovými testy pokryté, pokud možno, všechny metody tříd sloužících pro reprezentaci záznamu zpracování stromu. Pokrytí jednotlivých tříd je v tabulce 4.3.

Třída	**Pokrytí jednotkovými testy**
CL_DMEE_TRACE_TREE	62,5%
CL_DMEE_TRACE_NODE	96,55%
CL_DMEE_TRACE_RECORD	100%
CL_DMEE_TRACE_RECORD_ROOT	100%
CL_DMEE_TRACE_RECORD_ELEM	100%
CL_DMEE_TRACE_RECORD_SEGM	100%
CL_DMEE_TRACE_RECORD_SEGG	100%
CL_DMEE_TRACE_RECORD_CALC	100%

Tabulka 4.3: Pokrytí tříd nástroje pro ladění jednotkovými testy. U třídy `CL_DMEE_TRACE_TREE` je pokrytí nižší z důvodu častějšího používání konstrukcí obtížné k pokrytí např. SQL.

4.10.2 Manuální testování

Manuální testování probíhalo jednak z moji strany, za druhé každou funkcionalitu ještě manuálně testoval nějaký druhý člověk. Tuto pasáž budu popisovat čistě ze svého pohledu.

Pro každou dílčí část funkcionality jsem si definoval jednoduchý testovací případ, který jsem potom ověřoval. Při vytváření těchto dílčích případů jsem dbal na to, aby v nich byla do maximální možné míry zachycena očekávání zákazníků. Pro úplnost uvádím některé z případů.

- Testování správnosti záznamu na základě analýzy dočasného XML souboru.
- Testování správnosti záznamů všech typů uzlů.
- Testování správného chování nástroje při iteracích při zpracování stromu (počítadla iterací, vazby mezi uzly).
- Testování správného zaznamenání statických dat stromu.
- Testování správného zaznamenání dynamických dat - pro všechny typy uzlů a typy mapování.
- Testování funkcionality vyhledávání na základě všech kombinací možných vstupů.

Chyby nalezené v rámci manuálního testování, ať už mnou nebo někým jiným, byly na základě závažnosti buď řešeny okamžitě nebo v některém z následujících sprintů.

4.10.3 Regresní testování

Vzhledem k zásahům do zdrojového kódu DMEEX Enginu bylo na místě pravidelně regresně testovat, aby se minimalizovala rizika možného zanesení chyby, která by způsobila neočekávané chování při zpracování stromu. Automatické periodicky spustitelné testy pro DMEEX Engine již existovaly z minulosti, proto jsem je pouze důkladněji kontroloval.

Tyto regresní testy jsou velmi důkladné, spouští se nad všemi DMEEX stromy, které společnost SAP dodává pro platby a zahrnují široké spektrum funkcionality DMEEX. Výsledky regresního testování tedy považuji za velmi směrodatné. V průběhu vývoje resp. integrace jsem nezpůsobil žádnou chybu, která by vedla k selhání některého z automatických regresních testů.

4.11 Zpětná vazba od cílových skupin

Vzhledem k použité metodice vývoje probíhaly s cílovou skupinou vývojářů pravidelné diskuze v době implementace nástroje. Tyto diskuze měly za následek dílčí změny, které jsem již popisoval výše. Konkrétně šlo o:

- Změnu formy zobrazení podmínek.
- Grafické znázornění chybně vyhodnocené podmínky.
- Implementaci podpory kalkulačního uzlu.
- Možnost exportovat a importovat záznam.

Zpětná vazba na nástroj ze strany vývojářů je pozitivní. Používání v praxi ukázalo významný přínos při analýze zpracování definičních stromů DMEEX. Rozhodl jsem se tato tvrzení potvrdit měřením jednoduchých scénářů, které vychází z častých problémů popisovaných v podkapitole 4.2, viz tabulka 4.4. Měření jsem prováděl s uživatelem, který se nevyzná ve zdrojových kódech DMEEX, ale má dobrou uživatelskou znalost této transakce. Vzhledem k malému vzorku je vhodné, brát naměřené údaje s rezervou.

Problém	**Čas nalezení příčiny s nástrojem**	**Čas nalezení příčiny bez nástroje**
Chybějící uzel ve výstupním souboru	do 2 minut	desítky minut
Chybějící blok ve výstupním souboru	do 1 minuty	do 10 minut
Problém s konverzí	do 2 minut	desísky minut

Tabulka 4.4: Měření časové náročnosti nalezení problému s a bez nástroje pro ladění.

Kapitola 5

Závěr

Práce popsala a porovnala systémy SAP R/3 a SAP S/4HANA, jejich vrstvy a základní komponenty. Dále detailně rozebrala transakce DMEE a DMEEX v rámci integrace a podpory platebních formátů v systémech SAP. S použitím příkladů byla demonstrována funkcionality těchto transakcí a představeny problémy se kterými se uživatelé často setkávají při odlaďování platebních formátů. Proběhlo zpracování detailní analýzy současné situace a představeny nástroje, které jsou v současné době používány za podobným účelem. Z této analýzy vzešlo potvrzení o užitečnosti podobného nástroje.

Na základě této analýzy jsem vytvořil předběžný návrh řešení, který jsem představil třem typickým skupinám uživatelů transakcí DMEE a DMEEX. S každou z cílových skupin jsem zorganizoval meeting za účelem zjištění více požadavků koncových uživatelů. Práce s cílovými skupinami se osvědčila, protože všechny přinesly do původního návrhu řadu zlepšení, které nakonec byly téměř všechny realizovány. Vytvořil jsem finální návrh na základě jejich požadavků, který byl koncepčně i funkčně významně odlišný od původního návrhu. Vzhledem k iterativnímu vývoji pomocí metodiky Scrum mohla být vybraná cílová skupina účastna celého procesu vývoje a dílčí inkrementy produktu připomínkovat.

Nástroj jsem se rozhodl implementovat pomocí programovacího jazyka ABAP s uživatelským rozhraním kompatibilním se SAP GUI. Při vývoji nástroje pro ladění pro DMEEX na platformě S/4HANA byl dodržen původní předpoklad minimalizace zásahů do původního kódu DMEEX Engine a díky zvoleným technologiím bude v budoucnu snadné nástroj přenést na starší platformu SAP R/3, kde se stále těší oblibě původní transakce DMEE. Použitá metodika vývoje přinesla několik nových funkcí, které byly dodatečně navrženy až při vývoji nástroje např.: implementace podpory kalkulačního uzlu nebo možnost exportu a importu záznamu, díky kterému má nástroj využití i na platformě S/4HANA Cloud.

Z prvních uživatelských zkušeností jasně vyplývá, že nástroj pro ladění výrazně urychluje proces hledání specifických chyb v rámci zpracování stromu, to se potvrdilo po interním používání tohoto nástroje ve společnosti SAP. Nástroj se podařilo navrhnout a implementovat tak, že splňuje původní očekávání co se týká funkcionality, nenarušuje původní kód DMEEX Engine, protože veškerá logika nástroje pro ladění je obvykle zapouzdřena do jediného volání metody v rámci jednoho kroku zpracování a je možné ho snadno přenést na jiné platformy.

Z finálního návrhu bylo implementováno vše, s drobnými úpravami provedenými po konzultacích s cílovými skupinami. Prostor pro vylepšení vidím např. ve vyhledávání uzlů, kde by se na základě vyhledávacích kritérií mohly vyhledat všechny uzly splňující tato kritéria a zobrazit v přehledném seznamu. Dále se lze zamyslet nad formou zobrazení zaznamenaného

stromu, protože pro stromy o tisíci uzlech a mnoha iteracích může být přehlednost záznamu zhoršena.

Literatura

[1] *ABAP* [online]. [cit. 2020-03-30]. Dostupné z: https://en.wikipedia.org/wiki/ABAP.

[2] *Compression Types Exist in SAP HANA* [online]. [cit. 2020-04-22]. Dostupné z: https://www.stechies.com/which-compression-types-exist/.

[3] *SAP R/2* [online]. [cit. 2020-03-30]. Dostupné z: https://en.wikipedia.org/wiki/SAP_R/2.

[4] *SAP Versions Release and History of Evolution* [online]. 2015 [cit. 2020-03-30]. Dostupné z: https://www.stechies.com/about-sap-erp-solution-different-versions/.

[5] Bardhan, D., Baumgartl, A., Choi, N.-S., Dudgeon, M., Lahiri, A. et al. *SAP S/4HANA An Introduction*. 3. vyd. Rheinwerk Publishing, Inc., 2019. ISBN 987-1-4932-1775-5.

[6] Maasen, A., Schoenen, M., Frick, D. a Gadatsch, A. *SAP R/3 kompletní průvodce*. 1. vyd. Computer Press, a.s., 2007. ISBN 987-80-251-1750-7.

[7] Reddy, A. K. *What are SAP ERP Architecture Models?* [online]. 2017 [cit. 2020-03-30]. Dostupné z: https://sapnuts.com/courses/core-abap/sap-intro/sap-architecture.html.

[8] SE, S. *Accessing the Database in the R/3 System* [online]. [cit. 2020-03-30]. Dostupné z: https://help.sap.com/doc/saphelp_scm41/4.1/en-US/fc/eb3976358411d1829f0000e829fbfe/content.htm?no_cache=true.

[9] SE, S. *Columnar Data Storage* [online]. [cit. 2020-04-22]. Dostupné z: https://help.sap.com/viewer/52715f71adba4aaeb480d946c742d1f6/1.0.12/en-US/8c1fb4ff2f9640ee90e2dccea49c1739.html.

[10] SE, S. *Dynpro Flow and Dynpro Sequences* [online]. [cit. 2020-03-30]. Dostupné z: https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-US/abenabap_dynpros_processing.htm.

[11] SE, S. *Overview of SAP NetWeaver AS ABAP* [online]. [cit. 2020-03-30]. Dostupné z: https://help.sap.com/doc/saphelp_nw70ehp3/7.03.19/en-US/fc/eb2e97358411d1829f0000e829fbfe/content.htm?no_cache=true.

[12] SE, S. *Payment Medium Workbench* [online]. [cit. 2020-05-27]. Dostupné z: https://help.sap.com/viewer/5424e112323f4f8fa760fe742cde1a1c/6.17.17/en-US/7e3ac6535e601e4be10000000a174cb4.html.

[13] SE, S. *Statements in the Screen Flow Logic* [online]. [cit. 2020-03-30]. Dostupné z: https://help.sap.com/doc/abapdocu_752_index_htm/7.52/en-US/abenabap_dynpros_dynpro_statements.htm.

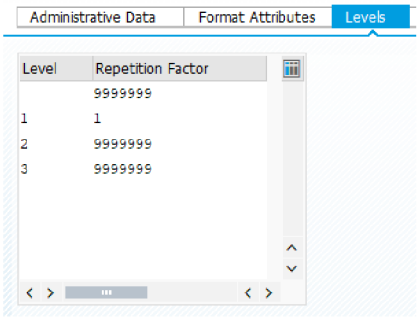
[14] SE, S. *Supported Databases* [online]. [cit. 2020-04-21]. Dostupné z: https://help.sap.com/viewer/ccc9cdbdc6cd4eceaf1e5485b1bf8f4b/7.5.9/en-US/840cb892edfbc34290c1685132006662.html.

[15] SE, S. *Work Processes* [online]. [cit. 2020-03-30]. Dostupné z: https://help.sap.com/saphelp_me60/helpdata/en/fc/eb2e7d358411d1829f0000e829fbfe/content.htm?no_cache=true.

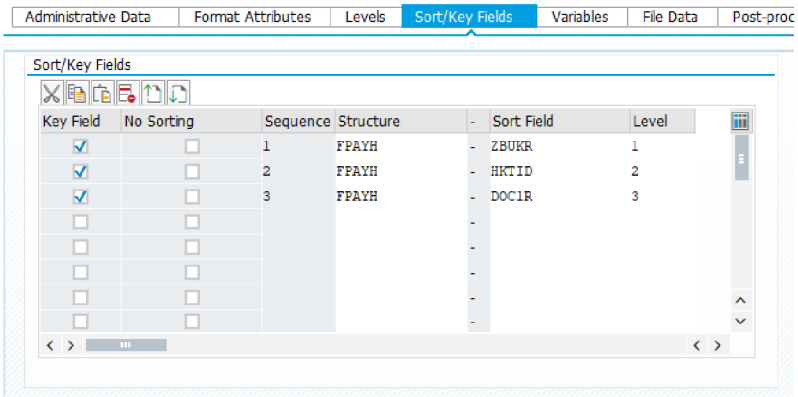
Příloha A

Přílohy

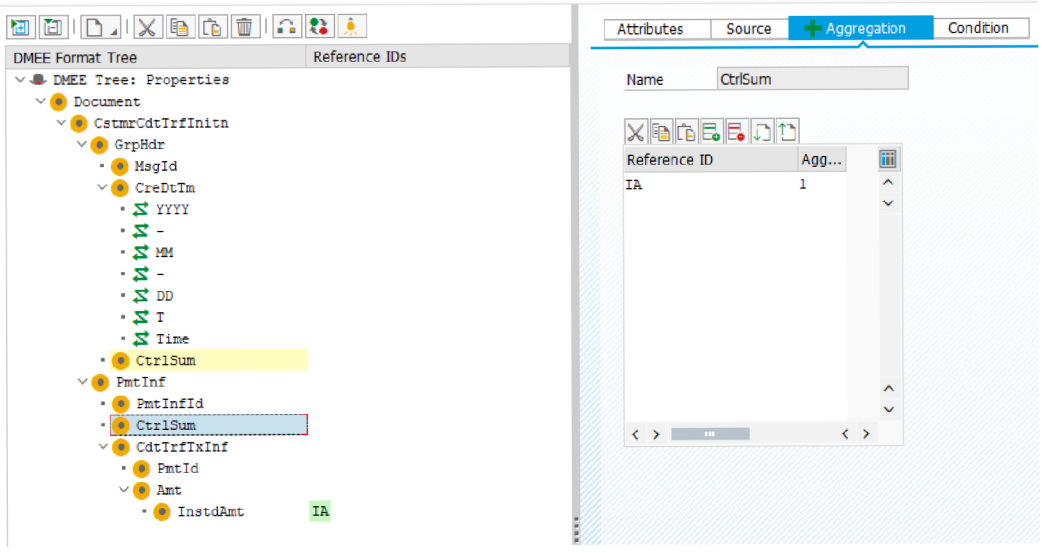
A.1 Demonstrace definice stromu v DMEEX



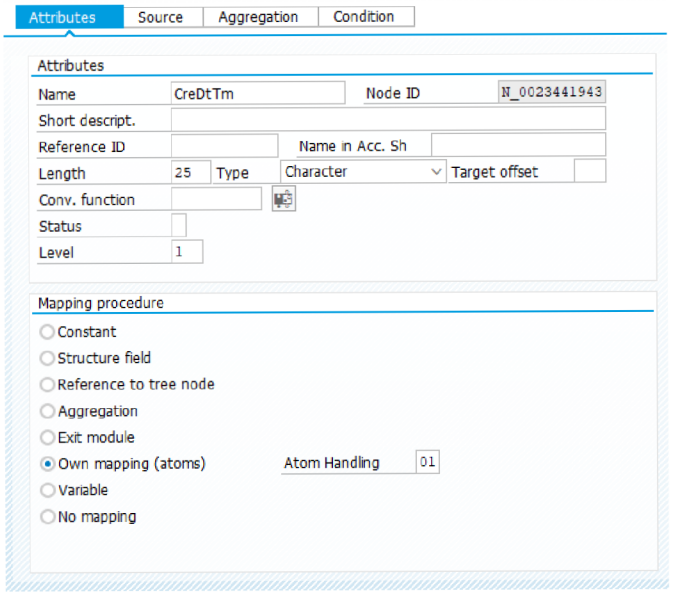
Obrázek A.1: Definice maximálního opakování úrovní v definičním stromě.



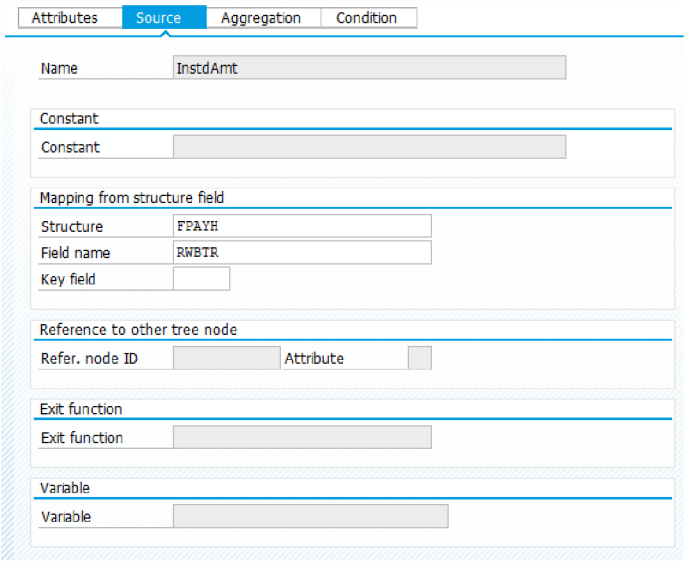
Obrázek A.2: Definice klíčových polí pro každou z úrovní.



Obrázek A.3: Nastavení agregace na uzlu `CtrlSum` za použití „Reference ID“ `IA` z uzlu `InstdAmt`.



Obrázek A.4: Element mapující datum a čas, který využívá atomy pro formátování svého výstupu. Volba „Atom handling“ nastavená na 01 znamená, že hodnoty z atomů budou v nadřazeném elementu spojeny.



Obrázek A.5: Nastavení zdrojové struktury a jejího pole na elementu, který mapuje částku.

A.2 Obsah přiloženého paměťového média

- složka `testy` - jednotkové testy
- složka `třídy` - zdrojové kódy tříd
- složka `vizualizator` - zdrojový kód programu pro vizualizaci
- složka `vizualizator_upload` - zdrojový kód programu pro nahrání exportovaného záznamu