Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Meta Matjaž

Tehnike dodatkov standardnih SAP programov

diplomsko delo

interdisciplinarni univerzitetni
študijski program prve stopnje
računalništvo in matematika

Mentor: doc. dr. Luka Šajn

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidatka naj razišče, kako lahko dodamo kodo po meri v standardne SAP ERP programe. Pri raziskavi naj se osredotoči na tehnike dodatkov. V diplomskem delu naj predstavi vsako tehniko, ter jo uporabi na praktičnem primeru. Razišče naj tudi, katere tehnike so bile uporabljene pri razvoju aplikacij podjetja Akademika d.o.o.

Zahvaljujem se doc. dr. Luki Šajnu za mentorstvo in prijazno pomoč pri izdelavi diplomske naloge.

Zahvaljujem se Dejanu Vidergarju in Leonu Ovnu, da sta mi predstavila svet SAPa, ter sta mi vedno znova pripravljena priskočiti na pomoč, ko se v tem svetu znajdem v slepi ulici. Zahvalila bi se tudi vsem sodelavkam in sodelavcem, saj me vsak dan znova nasmejijo.

Zahvaljujem se tudi sestri Staši za vso podporo in spodbudo v zadnjih treh desetletjih. Hvala tudi zaročencu Nejcu za vso ljubezen, podporo, potrpljenje in smeh.

Na koncu pa bi se rada zahvalila še mati. Brez njene ljubezni, vzgoje in neomajne podpore na moji izobraževalni poti danes ne bi bila kjer sem.

Hvala.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ABAP	Advanced Business Application Programming	programski jezik za SAP
ERP	Enterprise resource planning	celovita programska rešitev
BAdI	Business Add-Ins	poslovni dodatki
BTE	Business Transaction Events	dogodek poslovne transakcije
FM	Function Module	funkcijski modul
SSCR	SAP Software Change Registration	registracija sprememb SAPove programske opreme

Seznam uporabljenih izrazov

angleško	**slovensko**
enhancement techniques	tehnike dodatkov
user exit	uporabniški izhod
customer exit	izhod stranke
enhancement framework	ogrodje dodatkov
BAdI handle	BAdI oprimek
BAdI adapter class	BAdI prilagoditveni razred
SAP Reference IMG	SAPova vodnik za implementacijo
fallback class	rezervni razred
SAP business object	SAPov poslovni objekt
INCLUDE program	vsebnik
Modification Assistent	Asistent za modifikacijo
SAP Support Portal	SAPov podporni portal
plug-in	vtičnik
trigger	prožilec

Povzetek

Naslov: Tehnike dodatkov standardnih SAP programov

Avtor: Meta Matjaž

V diplomskem delu sem preučila vse tehnike dodatkov, s katerimi lahko dodajamo kodo po meri v standardne SAP ERP aplikacije. Za vsako posamezno tehniko dodatkov sem naredila tudi praktični primer uporabe in predstavila prednosti ter slabosti. Naredila sem tudi primerjavo tehnik dodatkov Klasični BAdI in Novi BAdI ter ugotovila, da je slednja hitrejša. Pri preučevanju tehnik dodatkov sem ugotovila, da SAP vedno priporoča uporabe najnovejših tehnik, vendar velikokrat nimamo izbire, katero tehniko bomo uporabili. Če želimo dodati kodo po meri na določen del v programu, ne moremo prosto izbirati tehnike, vendar lahko uporabimo samo tiste tehnike, ki so tam na voljo. Če imamo na voljo več tehnik, se odločimo za uporabo najnovejše, ki nam omogoča uporabo primernih podatkov.

Preučila sem tudi uporabo tehnik dodatkov v aplikacijah podjetja Akademika d.o.o. Ugotovila sem, da smo se izogibali uporabi tehnik dodatkov, ki jih je mogoče implementirati samo enkrat, ostale tehnike dodatkov pa smo uporabili.

Ključne besede: SAP, programski jezik ABAP, tehnike dodatkov SAP programov, uporabniški izhod, izhod stranke, funkcije BTE, BADIji, ogrodje dodatkov.

Abstract

Title: Enhancement techniques of standard SAP programs

Author: Meta Matjaž

In my thesis, I examined enhancement techniques that can be used to add custom code to standard SAP ERP application. For each enhancement technique, I made a practical use case and presented the technique's pros and cons. I also made a comparison of the techniques Classical BAdI and New BadI and proved that the latter technique is faster. When examining enhancement techniques, I found that SAP always recommends the use of the latest technique. However often we do not have a choice of which technique to use. If we want to add custom code to a specific part of the program, we cannot freely choose the technique, we can only use the ones available in that part of the code. If we have multiple techniques available, we should use the latest, that offers the use of relevant data.

I also examined the use of enhancement techniques in the applications of Akademika d.o.o. and found that we avoid the use of enhancement techniques that can be implemented only once. All other techniques are used.

Keywords: SAP, ABAP programming language, enhancements techniques, user exit, customer exit, BTE functions, BAdIs, enhancement framework.

Poglavje 1

Uvod

SAP ERP je celovita programska rešitev nemškega podjetja SAP, ki je zelo razširjena v poslovnem sektorju pri nas in tudi drugod po svetu. SAP ERP strankam ponudi nešteto aplikacij za različne module kot so: finančno računovodstvo in kontroling, prodaja in distribucija, načrtovanje proizvodnje, upravljanje materialov, upravljanje s človeškimi viri in druge.

Kljub temu, da SAP ponuja veliko različnih funkcionalnosti, pa v poslovnem svetu hitro ugotovimo, da ima vsako podjetje tudi svoje lastne želje in zahteve, kako bi radi dane aplikacije prilagodili oziroma jim nekaj dodali. Ker je SAP predvidel, da bodo programerji njihove aplikacije želeli spreminjati, jim je ponudil tehnike dodatkov, s katerimi lahko to storijo. Različne tehnike obstajajo, ker se je programski jezik ABAP, v katerem so SAPove aplikacije spisane, v zadnjih štirih desetletjih zelo spremenil. Na začetku je bil jezik proceduralen, šele kasneje so v programskem jeziku omogočili tudi objektno programiranje. Z razvojem jezika so se dodajale tudi nove tehnike dodatkov. Ker je SAP želel ohraniti združljivost s starejšimi verzijami, se prejšnje tehnike dodatkov niso nikoli odstranile[7]. Nove tehnike so bile razvite tudi zato, da bi odpravile pomanjkljivosti starejših[7].

V diplomskem delu sem najprej naredila pregled področja, kjer sem raziskala, kako so se druga podjetja, ki razvijajo celovite programske rešitve, spopadla z razširjanjem standardnih aplikacij. Na dveh primerih si bomo

ogledali, kakšne tehnike dodatkov so ponudili razvijalcem.

Nato sem v diplomskem delu predstavila, katere tehnike dodatkov standardnih SAP programov trenutno obstajajo in kako jih lahko uporabimo. Opisala sem prednosti in slabosti posameznih tehnik ter jih uporabila na praktičnem primeru. Praktične primere sem naredila na dveh standardnih SAP aplikacijah: upravljanje z matičnimi podatki dobavitelja in upravljanje s prodajnimi nalogi.

Raziskala sem tudi, katere vrste dodatkov uporablja podjetje Akademika d.o.o., kjer sem zaposlena. V podjetju so bile razvite aplikacije za potrjevanje računov, nabave, prodajnih nalogov, potnih nalogov in še veliko drugih. Večina aplikacij se opira na SAP dokumente, ki se kreirajo v standardnih aplikacijah. Zato so bile v podjetju uporabljene različne tehnike dodatkov, da se pri dodajanju SAP dokumenta ustvari in zažene delovni tok Akademike za potrjevanje. Tehnike dodatkov so bile uporabljene tudi pri vzdrževanju konsistentnosti podatkov med standardno SAP aplikacijo in dokumenti Akademike.

Poglavje 2

Pregled področja

V tem diplomskem delu bom predstavila, kako lahko vplivamo na standardno SAP kodo, kako lahko kodi dodajamo svojo poslovno logiko oziramo kako lahko programsko prilagajamo standardne SAP aplikacije. Ker obstaja veliko različnih ERPjev, si pred tem poglejmo, kako so prilagoditve standardne kode rešili na primeru dveh ERPjev: Oracle NetSuite ERP in Microsoft Dynamics NAV. Prvi sistem je v oblaku, drugi pa se namesti direktno pri stranki, zato si bomo lahko ogledali razlike pri dodajanju standardne kode na dveh sistemih, ki imata različno vrsto namestitve.

2.1 Oracle NetSuite ERP

NetSuite je celovita programska rešitev (ERP) v oblaku podjetja Oracle[9]. Standardne NetSuite kode ni mogoče spreminjati. Razvijalci lahko prilagodijo standardno poslovno logiko NetSuita z uporabo SuiteScripta[9]. SuiteScript je računalniško okolje, ki omogoča prilagoditev in avtomatizacijo poslovnih procesov znotraj NetSuite[9]. S SuiteScript razvijalci kreirajo skripte. Te skripte se lahko kličejo ob predefiniranih dogodkih, z njimi pa spreminjajo poslovne zapise ali podatke, ki jih je vnesel uporabnik[2]. Predefinirani dogodki so na primer: sprememba polja, shranjevanje obrazca, pred branjem, pred pisanjem, ob spletnih zahtevah[2].

Druga podjetja lahko razvijajo lastne aplikacije s pomočjo SuiteApps. Znotraj lastnih aplikacij lahko dodajo vtičnike (angl. plug-ins), ki spominjajo na SAPove BAdIje (poglavje 7), saj so tudi definirani na podlagi vmesnika. Vmesnik ima privzeto implementacijo, ki jo nato drugi razvijalci redefinirajo in tako nadomestijo s svojo lastno logiko. Obstajajo tudi jedrni vtičniki, ki jih razvije NetSuite. Koda, ki je definirana znotraj jedrnega vtičnika se izvede znotraj standardne NetSuite kode, druga podjetja pa jo lahko nadomestijo z implementacijo vtičnika [5].

2.2 Microsoft Dynamics NAV

V tem podpoglavju si bomo pogledali, kako so omogočili dodajanje kode po meri v ERP sistemu Microsoft Dynamics NAV. Ta sistem ni v oblaku, za razliko od Oracle NetSuite ERP. Zaradi tega pride tudi do razlike pri spreminjenju standardne kode. Pri Microsoft Dynamics NAV sistemu se standardna koda lahko spreminja[8]. Z uporabo integriranega razvojnega okolja C/SIDE in uporabo jezika C/AL se lahko ureja standardne objekte, ki jih je naredil Microsoft[8]. Tu lahko potegnemo vzporednice s SAP ERP sistemom. Tudi ta sistem ni v oblaku in omogoča modificiranje standardne kode. Vendar pri SAPu to ni priporočljivo. Za razliko od Microsoft Dynamics NAV, pri SAPu potrebujemo za vsak objekt, ki ga želimo modificirati, ključ za dostop, ki ga dobimo samo, če smo za to posebej avtorizirani. Kot pri SAPu, pa tudi pri Microsoft Dynamics NAV lahko pride do komplikacij pri nadgradnji sistema. Če ima produkt veliko modifikacij, potem ga je težko vzdrževati in težko nadgraditi na novo verzijo[8].

Drugi način dodajanja kode po meri v standardno kodo pa je preko dogodkov. Vsi objekti v Microsoft Dynamics NAV imajo prožilce (angl. trigger)[8]. To so predefinirane funkcije znotraj objektov, ki se samodejno prožijo, ko se zgodi določena akcija na objektu[8]. Dogodki pa se izvedejo pred in po prožilcih. Z uporabo dogodkov lahko dodamo kodo po meri v standardne aplikacije, kljub temu pa se standardna koda ne spremeni[8]. Tako lahko

ločimo kodo po meri od standardne kode[8]. Zato je produkt lažje vzdrževati in nadgraditi, saj nadgradnja ne vpliva na kodo po meri [8].

2.3 Sklepi pregleda področja

Opazimo lahko, da standardne kode sistema v oblaku ne moremo spreminjati, standardno kodo sistema, ki je nameščen pri stranki, pa lahko spreminjamo. Zato lahko sklepamo, da je to res v večini primerov, vendar bi morali pregledati več ERP sistemov, da bi to izjavo lahko potrdili. Kljub temu pa opazimo, da sta oba sistema ponudila razvijalcem tehnike, s katerimi lahko prilagajajo funkcionalnosti standardnih aplikacij. Sklepamo lahko, da so tudi razvijalci drugih ERP sistemov opazili, da včasih stranke želijo prilagoditi funkcionalnosti standardnih aplikacij.

Poglavje 3

Tehnike dodatkov

Tehnike dodatkov so različne tehnike, ki omogočajo nadomestitev standardne SAP kode ali pa samo dodajanje lastne kode na različna mesta v standardnih aplikacijah. S temi tehnikami lahko prilagodimo standardne SAP ERP aplikacije, da ustrezajo željam, zahtevam in potrebam stranke. Za razliko od modifikacije standardne kode, se pri tehnikah dodatkov standardna koda ne spreminja direktno, koda po meri pa je napisana v imenskem prostoru stranke[7].

ERP (angl. Enterprise resource planning) je kratica, ki pomeni celovito programsko rešitev. Celovite programske rešitve se od drugih sistemov razlikujejo po tem, da več različnih poslovnih oddelkov uporablja skupno bazo podatkov[7]. Prednost ERP sistemov je v tem, da če en oddelek posodobi podatke, so ti takoj dostopni tudi vsem drugim oddelkom[7]. SAP ERP vsebuje več različnih modulov: Finančno računovodstvo (FI), Kontroling (CO), Človeški viri (HR), Prodaja in distribucija (SD), Upravljanje z materiali (MM), Vzdrževanje obratov (PM), Načrtovanje proizvodnje (PP) in druge[7]. Če na primer v modulu Upravljanje z materiali označimo, da je določenega materiala zmanjkalo, bo to informacijo takoj videl tudi prodajni oddelek, saj vsi SAP ERP moduli uporabljajo skupno bazo podatkov[7].

Programski jezik za SAP se imenuje ABAP in se je prvič pojavil leta 1983[4]. Od takrat naprej se je jezik neprestano razvijal. Na začetku je bil

samo proceduralen, šele kasneje so v programski jezik dodali tudi objektno programiranje. Ker se je jezik razvijal, so se dodajale vedno nove in izboljšane tehnike dodatkov. Prva tehnika dodatkov je bila Uporabniški izhod in je zato zelo preprosta. V kodi je predstavljena kot klic podrutine, kjer se lahko doda koda po meri. Naslednja tehnika dodatkov, ki so jo dodali, je Izhod stranke. Pri uporabi te tehnike kodo po meri dodamo v vsebnike, ki so znotraj funkcijskih modulov. Nato so dodali tehniko dodatkov, ki se proži ob dogodkih poslovnih transakcij. Tehniko imenujemo BTE funkcije, saj kodo po meri dodamo v svoje funkcijske module, ki jih nato registriramo za dogodke poslovnih transakcij. Tehnika je bila dodana leta 1998[6][11]. Leta 2002[6][7] so dodali tehniko dodatkov Klasični BAdI. Definicija klasičnih BAdIjev vsebuje vmesnik z metodami. Svojo kodo po meri dodamo tako, da naredimo implementacijo vmesnika in metode redefiniramo. Zadnjo tehniko dodatkov so dodali leta 2005[6][7], imenuje se Ogrodje dodatkov. To v resnici ni ena sama tehnika, ampak tri različne tehnike, ki so jih zapakirali v ogrodje. Ena izmed teh tehnik je tudi tehnika dodatkov Nov BAdI. Ta tehnika dodatkov je v resnici izboljšava klasičnega BAdIja, ki uporablja tudi druge elemente ogrodja dodatkov.

V prihodnjih poglavjih bom posamezne tehnike dodatkov predstavila bolj podrobno, prikazala praktične primere uporabe, ter predstavila prednosti in slabosti vsake tehnike. Čeprav nekatere tehnike dodatkov omogočajo tudi spreminjanje ekranov in menijev preko grafičnega vmesnika, se bom v tej diplomski nalogi osredotočila na programske rešitve pri uporabi tehnik dodatkov.

Tehnično računalniško okolje, ki ga uporablja SAP ERP, se imenuje SAP NetWeaver. Vsi praktični primeri v tej diplomi so bili narejeni na SAP NetWeaver 7.0 Enhancement Package 1, razen primera eksplicitnih dodatkov, ki je bil narejen na SAP NetWeaver 7.5.

Poleg tehnik dodatkov poznamo tudi modifikacijo standardne SAP kode. Pri modifikaciji SAP kode lahko s pomočjo orodja Asistent za modifikacijo direktno spreminjamo standardno SAP kodo tako, da vstavljamo lastno kodo,

nadomestimo del standardne kode s kodo po meri, ali pa izbrišemo del standardne kode. Čeprav SAP omogoča modifikacjo standardne kode, je kljub temu ne priporoča. Pri nadgradnji standardne kode, se morajo modifikacije ponovno implementirati, če se nadgradi standardni SAP program[7]. Zaradi tega se bom v tem diplomskem delu osredotočila na tehnike dodatkov in ne na modifikacije. Modifikacije bodo omenjene samo pri tehniki dodatkov Uporabniški izhod (poglavje 4), saj so modifikacije del te tehnike.

Poglavje 4

Uporabniški izhod

Uporabniški izhod je prva tehnika dodatkov, ki jo je SAP ponudil razvijalcem. Uporablja se predvsem v modulu za prodajo in distribucijo. Ta tehnika dodatkov je v kodi predstavljena s prazno podrutino, ki je poimenovana USEREXIT_<IME>. Podrutine so znotraj lastnega vsebnika. Z vsebniki (angl. INCLUDE program) lahko razdelimo SAP programe na več delov [3]. Vsebnik je objekt, ki ga lahko samostojno prenašamo med sistemi, vendar pa ga ne moremo izvesti samostojno [3]. Izvedemo ga lahko, če izvedemo program, kjer se vsebnik nahaja. Uporabniški izhod se uporabi tako, da dodamo kodo po meri znotraj prazne podrutine.[6] [7]

Ta tehnika je v resnici mešanica tehnike dodatkov in modifikacije. Lahko rečemo, da gre za tehniko dodatkov, ker se v programu kliče posebna podrutina, kjer je bilo predvideno, da bomo dodali kodo po meri. Ker pa je koda znotraj vsebnika, ki ni v imenskem prostoru stranke, ne moremo urejati kode, saj nimamo avtorizacije za spreminjanje standardne SAP kode. Zato moramo SAP kodo modificirati. SAP kodo lahko modificiramo tako, da uporabimo orodje Asistent za modifikacijo. Z asistenom za modifikacijo lahko vstavljamo kodo po meri v standardno kodo. Dele standardne kode lahko tudi nadomestimo ali celo izbrišemo. Vsak objekt v SAPu, ki ga želimo modificirati, moramo najprej registrirati na SAPovem podpornem portalu (angl. SAP Support Portal) v aplikaciji za registracijo sprememb

SAPove programske opreme (aplikacija SSCR). Objekt moramo registrirati, da pridobimo ključ za dostop do objekta, saj drugače tam ne moremo izvesti modifikacije. Tudi za registracijo objektov na aplikaciji SSCR moramo imeti posebno avtorizacijo[7].

Modifikacije SAP kode niso priporočljive, saj se z nadgradnjo standardne SAP kode ponavadi izgubijo. Pri tej tehniki dodatkov so se temu izognili tako, da SAP pri nadgradnji nikoli ne briše ali spreminja vsebnikov, ki vsebujejo uporabniške izhode. V primeru potrebe po novem uporabniškem izhodu, bodo raje dodali nov vsebnik[6].

4.1 Primer uporabe

Ker se uporabniški izhodi uporabljajo predvsem v modulu za prodajo in distribucijo, bom primer uporabe prikazala v standardni SAP aplikaciji za ustvarjanje, spreminjanje in prikaz prodajnih nalogov (transakcije VA01, VA02, VA03). Recimo, da smo prejeli zahtevo stranke, da naj bo polje Razlog za nalog, ki se nahaja znotraj te aplikacije, obvezno. Zahtevo začnemo reševati tako, da najprej poiščemo primeren uporabniški izhod. Našla sem podrutino USEREXIT_FIELD_MODIFICATION znotraj vsebnika MV45AFZZ, ki je namenjena temu, da v njej spremenimo atribute polj na ekranu.

Pred modifikacijo vsebnika sem morala najprej pridobiti ključ za dostop. Ključ sem pridobila tako, da sem ta objekt registrirala preko aplikacije SSCR. Nato sem odprla vsebnik, šla v spremembo programa, in vnesla ključ. Ker sem po vnosu ključa imela avtorizacijo za modifikacijo vsebnika, so se na vrhu pojavili gumbi Vstavljanje, Nadomestitev, Brisanje in Razveljavitev, s katerimi lahko modificiramo kodo. Gumbe lahko vidimo na sliki 4.1. Uporabila sem gumb za vstavljanje kode in v podrutini se je pojavil komentar INSERT, znotraj katerega sem lahko dodala lastno kodo. Kodo, ki sem jo dodala, lahko vidite na sliki 4.1. Na koncu sem vsebnik samo še aktivirala in sprememba je bila vidna v aplikaciji za urejanje/pregled prodajnih nalogov, kjer je bilo polje Razlog za nalog po novem obvezno.

Slika 4.1: Implementacija uporabniškega izhoda

4.2 Prednosti

Tehnika ima resnično več slabosti kot prednosti. Ko je bila prvič dodana v standardne SAP programe, je bila njena glavna prednost v tem, da so razvijalci lahko dodajali lastno kodo standardnim SAP programom, saj drugih možnosti niso imeli na voljo. Ker je SAP kasneje dodal novejše tehnike, se zdaj uporaba te tehnike odsvetuje.

4.3 Slabosti

Pri tej tehniki moramo uporabiti modifikacijo standardne SAP kode, kar ni priporočljivo. Vsak objekt, ki ga želimo spremeniti, moramo registrirati, za kar rabimo tudi posebno avtorizacijo. Poleg tega pri modifikaciji ne moremo prosto spreminjati kode, ampak jo lahko samo vstavljamo, nadomestimo ali brišemo s posebnimi ukazi s pomočjo orodja Asistent za modifikacijo.

Poleg tega lahko dodajamo lastno kodo samo na mesta v SAP programu, kjer se kliče uporabniški izhod. Z uporabo te tehnike ne moremo dodati svoje

kode nikjer drugje. Tudi, če želimo nehati uporabljati kodo, ki smo jo dodali v uporabniški izhod, ne moremo kode samo deaktivirati, kot je to mogoče storiti pri nekaterih drugih tehnikah. V tem primeru moramo kodo po meri ročno izbrisati iz uporabniškega izhoda ali pa jo zakomentirati[7].

Pri uporabi te tehnike moramo biti previdni, saj lahko v podrutini dostopamo tudi do globalnih spremenljivk programa in tako spreminjamo tudi podatke, ki jih ne bi smeli spremeniti[7].

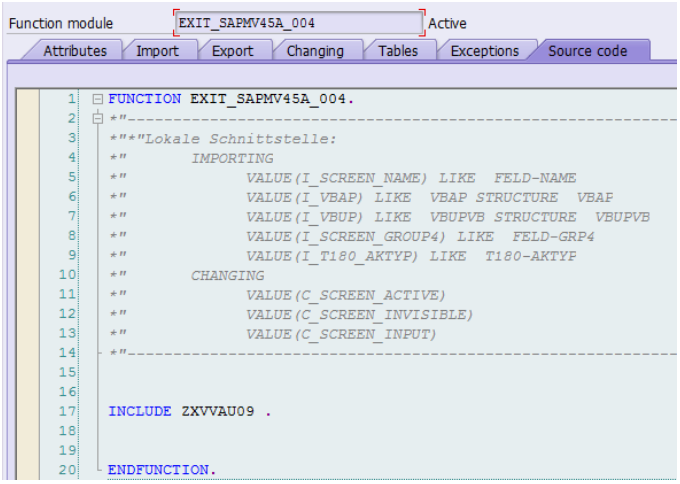
Poglavje 5

Izhod stranke

Uporabniškim izhodom je sledila tehnika dodatkov, ki se imenuje Izhod stranke. Izhod stranke je funkcijski modul, ki vsebuje samo klic vsebnika. Funkcijski modul je globalna funkcija, ki se jo lahko kliče tudi zunaj programov ali razredov [3]. Funkcijski modul za izhod stranke je vedno poimenovan po sledečem vzorcu EXIT_<IME_PROGRAMA>_<NNN>, kjer je NNN tromestna številka izhoda. V vsakem programu je lahko več izhodov stranke, zato so oštevilčeni s tromestno številko. Vsebnik je poimenovan tako, da ustreza imenskemu prostoru stranke, zato ga lahko spreminjamo. Imenski prostor stranke se v SAPu začne s črko Z ali Y[6] [7].

Na sliki 5.1 vidimo primer izhoda stranke, ki pripada standardnemu SAP programu SAPMV45A. To je program za obdelavo prodajnih nalogov. Vidimo tudi, da je to četrti izhod znotraj tega programa in vsebuje klic vsebnika ZXVVAU09.

Uporabniški izhodi se znotraj programov ne kličejo kot ostali funkcijski moduli v SAPu. Navadni funkcijski moduli se kličejo z ukazom CALL FUNCTION zraven katerega je zapisano ime funkcijskega modula. Izhod stranke pa se kliče z ukazom CALL CUSTUMER-FUNCTION, poleg pa je navedena tromestna številka izhoda. Primer klica uporabniškega izhoda s slike 5.1 lahko vidimo na sliki 5.2 [6] [7].



Slika 5.1: Primer izhoda stranke

```
CALL CUSTOMER-FUNCTION '004'
     EXPORTING
          I_SCREEN_NAME      = SCREEN-NAME
          I_VBAP             = VBAP
          I_VBUP             = XVBUP
          I_SCREEN_GROUP4    = SCREEN-GROUP4
          I_T180_AKTYP       = T180-AKTYP
     CHANGING
          C_SCREEN_ACTIVE    = SCREEN-ACTIVE
          C_SCREEN_INVISIBLE = SCREEN-INVISIBLE
          C_SCREEN_INPUT     = SCREEN-INPUT.
```

Slika 5.2: Klic izhoda stranke

5.1 Primer uporabe

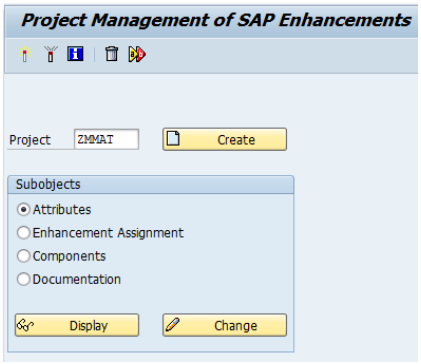
Recimo, da smo dobili zahtevo stranke, da znotraj standardne SAP transakcije za kreiranje, spremembo in prikaz prodajnih nalogov onemogočimo spremembo polj Blokada dobave in Blokada fakture. Ti dve polji želijo spreminjati samo programsko v ozadju in ne želijo, da bi jih uporabniki lahko spreminjali ročno.

Da lahko onemogočimo spremembo teh dveh polj, moramo najprej poiskati ustrezni izhod v pogramu. Po programu sem poiskala niz CUSTOMER-FUNCTION in ugotovila, da se na ustreznem mestu in z ustreznimi parametri kliče izhod stranke EXIT_SAPMV45A_004, ki ga lahko vidite na

sliki 5.1. Izhode stranke lahko alternativno poiščemo tudi s pomočjo transakcije SMOD, kjer lahko iščemo izhode stranke, ter pregledujemo njihove komponente.

Če želimo izhod stranke uporabiti, ga moramo dodeliti na projekt preko transakcije CMOD in ga označiti kot aktivnega. Transakcija CMOD je transakcija za projektno vodenje SAP dodatkov. S pomočjo te transakcije lahko logično povežemo več izhodov stranke na en projekt. Preko projekta jih lahko nato tudi aktiviramo ali deaktiviramo. V primeru, da dodatka ne želimo več uporabljati, nam ni potrebno brisati ali komentirati kode znotraj vsebnika, temveč produkt preprosto deaktiviramo [6].

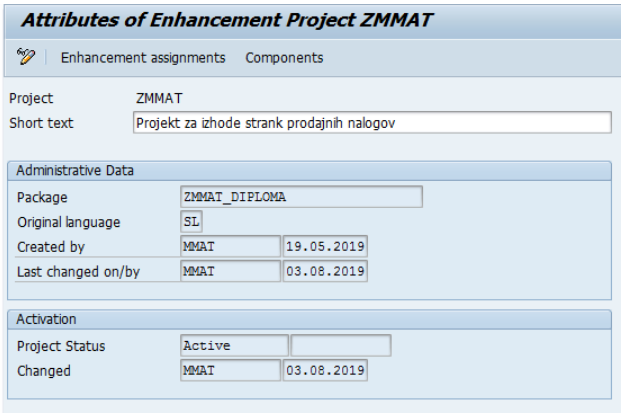
Če projekta za dodatke še nimamo, ga moramo najprej ustvariti. Projekt ustvarimo tako, da vpišemo tehnično ime projekta in kliknemo na gumb za kreiranje - Create (slika 5.3). V primeru, da projekt že obstaja, samo vnesemo tehnično ime in kliknemo na gumb za spremembo - Change.



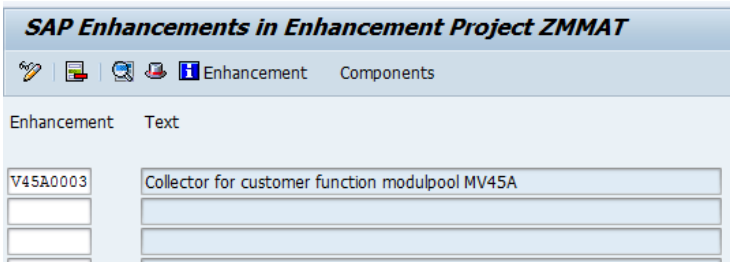
Slika 5.3: Kreiranje projekta v transakciji CMOD

Odpre se nam okno z osnovnimi podatki, ki ga vidimo na sliki 5.4. Pri kreiranju projekta moramo tu vnesti še kratko besedilo projekta in podatke shraniti.

Ko imamo projekt, mu dodelimo izhod stranke, ki ga želimo uporabiti. To storimo preko gumba za dodeljevanje dodatkov - Enhancement assignments. Odpre se nam okno, kjer lahko vnesemo skupino dodatkov, ki jo želimo uporabiti. V našem primeru je to V45A0003 (Slika 5.5).



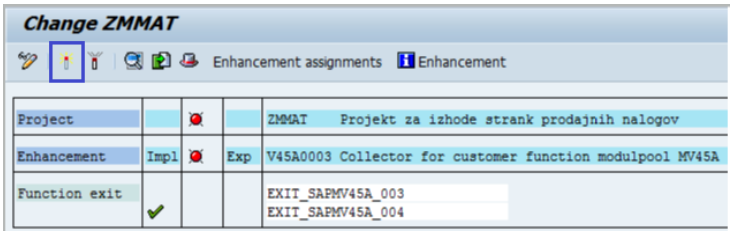
Slika 5.4: Osnovni podatki projekta



Slika 5.5: Dodeljevanje skupine dodatkov na projekt

S pomočjo gumba za komponente (Components) si lahko ogledamo in aktiviramo izhode strank. Na sliki 5.6 vidimo, da znotraj V45A0003 obstajata dva izhoda strank. Izhod stranke uporabimo tako, da dvakrat kliknemo na ime funkcijskega modula EXIT_SAPMV45A_004, kar nam odpre funkcijski modul, ki je viden na sliki 5.1. Nato kreiramo vsebnik, ki je znotraj funkcijskega modula tako, da dvakrat kliknemo na ime programa.

Na tem mestu pridemo do najpomembnejšega dela uporabe dodatka. Dodati moramo kodo po meri znotraj na novo kreiranega vsebnika. Za prikaz delovanja izhoda stranke sem na tem mestu dodala kodo, ki jo lahko vidite spodaj. Pogojni stavek poskrbi, da se bo koda izvedla samo v primeru kreiranja ali spremembe prodajnega naloga, ter samo v primeru polj blokade. Znotraj pogojnega stavka je stavek, ki spremeni vidljivost teh dveh polj tako,



Slika 5.6: Izhodi strank za komponente V45A0003

da sta na voljo samo za branje.

```
IF i_t180_aktyp CA 'VH' AND
 ( i_screen_name EQ 'VBAK-LIFSK' OR
   i_screen_name EQ 'VBAK-FAKSK' ).
    c_screen_input = 0.
ENDIF.
```

Ko je koda spisana in aktivirana, je potrebno še aktivirati uporabo izhoda stranke znotraj projekta v transakciji CMOD. Dokler je komponenta deaktivirana, se koda po meri ne bo uporabila. Na sliki 5.6 vidimo, da sta izhoda stranke in s tem tudi projekt še deaktivirana, saj vidimo rdečo ikono v vrstici projekta in implementacije dodatka. Ko kliknemo na gumb za aktivacijo, ki je na sliki označen z modro barvo, se projekt in funkcijski izhod aktivirata, rdeča ikona pa se spremeni v zeleno.

Po aktivaciji se koda znotraj izhoda stranke uporabi. Pri kreaciji ali spreminjanju prodajnega naloga preko transakcije VA01 ali VA02 ne moremo več spreminjati polj za blokado.

5.2 Prednosti

Izhodi stranke so nadgradnja uporabniških izhodov. Ker se pri izhodih stranke ne kliče podrutine, ki je del programa, ampak globalni funkcijski modul, tu ne moremo več dostopati do globalnih spremenljivk programa.

Dostopamo lahko le do tistih podatkov, ki so podani preko parametrov funkcijskega modula, kar je z vidika varnosti veliko bolje[6].

Druga zelo velika prednost pred uporabniškimi izhodi pa je projektno vodenje SAP dodatkov preko transakcije CMOD. S pomočjo projekta lahko skupaj povežemo in vodimo več različnih izhodov stranke in tako z njimi lažje upravljamo. Kodo znotraj vsebnika spreminjamo in aktiviramo, sprememba pa ne bo vidna, dokler CMOD projekt ni aktiviran [6].

5.3 Slabosti

Kljub prednostim projektnega vodenje izhodov strank pa obstajajo tudi slabosti. Vsako skupino dodatkov lahko dodelimo samo na en projekt. Če je skupina že uporabljena na projektu, potem moramo nujno uporabiti ta projekt, tudi če nova koda po meri logično ne spada znotraj obstoječega projekta ali pa če projekta nismo kreirali sami.

Razlog za to omejitev je v tem, da morajo vsi razvijalci uporabiti isti vsebnik znotraj izhoda stranke. Razvijalec, ki prvi kreira vsebnik, tudi določi paket, v katerem se le ta kreira. Do težav pride, če želi drug razvijalec ta vsebnik spremeniti, vendar nima avtorizacije za urejanje paketa, kjer je bil kreiran. Zaradi uporabe istega vsebnika lahko do težav pride tudi pri prenašanju popravkov na druge sisteme, kot na primer testni ali produkcijski sistem. Ko spreminjamo prenosni objekt (vsebnik) moramo v SAPu določiti tudi prenosni paket, s katerim se objekt uvozi drugam. Tudi tu prenosni paket določi prvi razvijalec, zato drugi razvijalci mogoče nimajo nadzora nad tem, kdaj se bodo spremembe prenesle naprej.

Kot pri uporabniškem izhodu je pomanjkljivost tudi ta, da se izhod kliče samo na določenih mestih v programu - kavljih [7]. Lastne kode zato ne moremo dodajati na poljubnem mestu, kar nam lahko prepreči, da bi določeno zahtevo rešili s to tehniko dodatkov.

Čeprav je z vidika varnosti bolje, da ne moremo več dostopati do vseh podatkov v programu, pa nam tudi pomanjkanje podatkov lahko prepreči

uporabo te tehnike dodatkov.

Zadnja slabost, ki je vredna omembe, je uporaba proceduralnega programiranja.

Poglavje 6

BTE funkcije

BTE funkcije so funkcijski moduli, ki se kličejo ob posebnih dogodkih v SAP programu - dogodkih poslovnih transakcij. To je tehnika dodatkov, ki se uporablja samo v dveh SAP modulih:

- finančno računovodstvo in kontroling,
- prodaja in distribucija [6].

BTE funkcije se kličejo z ukazom OPEN_FI_PERFORM_<NNNNNNNN>-_<T> ali OUTBOUND_CALL_<NNNNNNNN>_<T>. Osemmestna številka N predstavlja številko dogodka, T pa predstavlja tip oziroma vrsto vmesnika za BTE funkcijo. Poznamo namreč dve različni vrsti vmesnikov:

- **Objavi in naroči vmesnik**

  Vmesnik je predstavljen s črko E. Ko implementiramo BTE funkcijo s tem vmesnikom, bomo v funkcijski modul samo prejeli podatke iz programa, ne bomo pa mogli nanje vplivati. Ker take BTE funkcije ne spreminjajo podatkov v programu, jih lahko za isti dogodek registriramo neomejeno število. To lahko storimo, saj ni pomembno v kakšnem vrstnem redu se funkcije kličejo. Te funkcije se uporabljajo za izvoz poslovnih podatkov[6].

- **Procesni vmesnik** BTE funkcije s to vrsto vmesnika vplivajo na podatke v programu. Zato lahko za en dogodek registriramo največ en BTE funkcijski modul[6].

6.1 Primer uporabe

Tudi tu bom primer uporabe pokazala na programu SAPMF02K, ki skrbi za matične podatke dobaviteljev. Recimo, da smo od stranke prejeli zahtevo, da želijo pri blokadi knjiženja za vse šifre podjetja ali pri blokadi nabave za vse nabavne organizacije ustvariti zabeležko, kjer lahko obrazložijo razloge za blokado. Blokade knjiženja ali nabave se nastavijo preko standardne SAP transakcije XK05. V tej transakciji moramo zato poskrbeti, da se bo pri shranjevanju pojavilo okno, s katerim bodo lahko kreirali zabeležko.

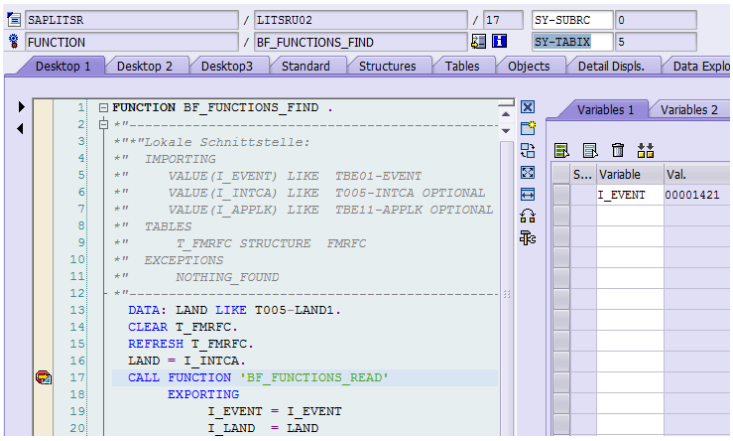
Če želimo zahtevo rešiti z uporabo BTE funkcij, moramo narediti naslednje korake:

- poiskati ustrezni dogodek poslovne transakcije,
- ustvariti funkcijski modul z BTE vmesnikom,
- registrirati nov funkcijski modul za izbran dogodek.

6.1.1 Iskanje dogodka poslovne transakcije

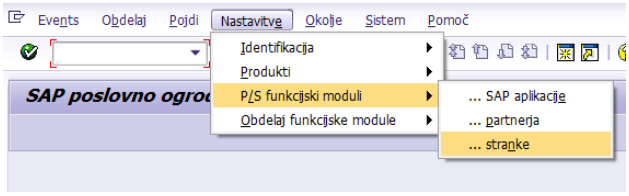
Ustrezen dogodek poslovne transakcije lahko poiščemo na več načinov. Prvi način je, da postavimo prekinitveno točko v funkcijski modul BF_FUNCTIONS_FIND, če iščemo objavi in naroči dogodke. V nasprotnem primeru pa postavimo prekinitveno točko PC_FUNCTION_FIND, ki se kliče ob procesnih dogodkih[10].

Ker v našem primeru ni potrebno spreminjati podatkov poslovne transakcije, postavimo prekinitveno točko samo v BF_FUNCTIONS_FIND. Po kliku na gumb za shranjevanje v transakciji XK05, se nam odpre okno za razhroščevanje, ki je prikazano na sliki 6.1. Pri pregledu vrednosti vhodnega parametra I_EVENT ugotovimo, da je dogodek, ki ga iščemo 00001421.



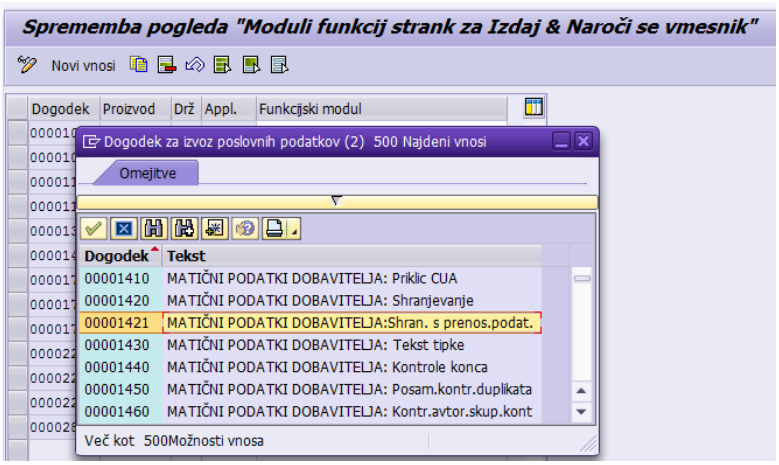
Slika 6.1: Iskanje BTE dogodka s BF_FUNCTIONS_FIND

Ustrezen dogodek bi lahko poiskali tudi preko transakcije FIBF. Ker iščemo objavi in naroči dogodke, sledimo poti Nastavitve → P/F funkcijski moduli → stranke, kot je prikazano na sliki 6.2.



Slika 6.2: FIBF transakcija

Pri uporabi pomoči za iskanje (F4) na polju Dogodek, se nam odpre seznam vseh objavi in naroči dogodkov (slika 6.3). Glede na tekst dogodkov, hitro ugotovimo, da so za matične podatke dobavitelja na voljo dogodki od 00001410 do 00001460. Tu vidimo, da sta za shranjevanje odgovorena dogodek 00001421 in 00001420. Mi bomo izbrali dogodek 00001421. saj se pri tem dogodku prenesejo še trenutni in stari podatki dobavitelja. Ker želimo omogočiti shranjevanje zabeležke le, kadar se postavi blokada, moramo vedeti tudi, kakšne so bile nastavitve blokade, ko je uporabnik transakcijo zagnal.



Slika 6.3: Dogodki poslovne transakcije za matične podatke dobavitelja

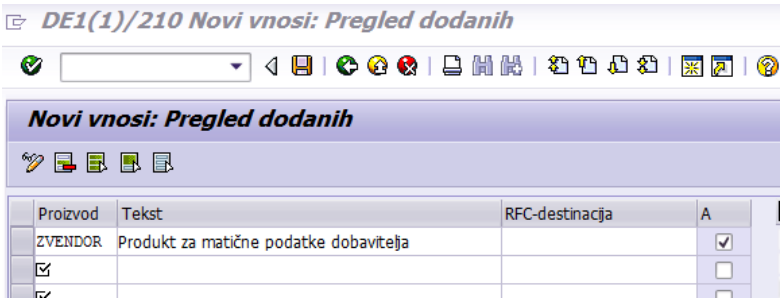
Dogodke lahko najdemo tudi tako, da iščemo niz OPEN_FI_PERFORM in OUTBOUND_CALL po kodi programa.

6.1.2 Kreacija funkcijskega modula z BTE vmesnikom

Ko najdemo ustrezen dogodek, moramo ustvariti funkcijski modul, v katerega bomo dodali svojo logiko za pošiljanje obvestila. Pri kreaciji funkcijskega modula moramo paziti, da ima popolnoma enak vmesnik, kot ga ima BTE funkcija za izbran dogodek. Tu nam je SAP olajšal delo tako, da je za vsak objavi in naroči dogodek na voljo funkcijski modul SAMPLE_INTERFACE_<NNNNNNNN>. Za procesne dogodke pa so na voljo funkcijski moduli SAMPLE_PROCESS_<NNNNNNNN>. To so prazni funkcijski moduli, ki imajo natančno take parametre, kot so zahtevani pri klicu dogodka. Zato modula ne kreiramo ročno, ampak samo kopiramo SAMPLE_INTERFACE_00001421. Na tak način sem kreirala funkcijski modul ZFM_MM_INTERFACE_00001421. V funkcijski modul sem dodala kodo, ki odpre okno za kreiranje zabeležke, če je uporabnik nastavil blokado knjiženja ali blokado nabave.

6.1.3 Registracija novega funkcijskega modula

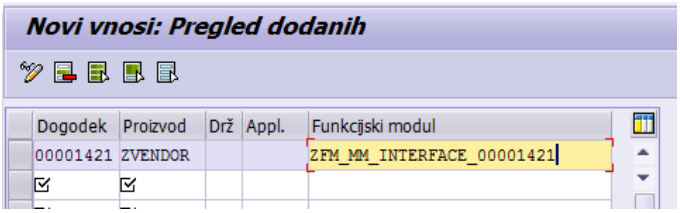
Po kreaciji funkcijskega modula se le ta še ne bo uporabil, dokler ga ne dodamo na BTE produkt in ga registriramo za dogodek. Produkti so objekti, na katere lahko dodamo funkcijske module, ki smiselno spadajo skupaj. Ker v našem primeru še nimamo produkta, ga je najprej potrebno kreirati. To storimo preko transakcije FIBF, transakcije za upravljanje z dogodki poslovnih transakcij. Slediti moramo poti Nastavitve → Produkti → stranke. Produkt kreiramo tako, da navedemo tehnično ime produkta in kratek tekst, kot lahko vidimo na sliki 6.4.



Slika 6.4: Kreacija novega produkta

Zelo pomemben je zadnji stolpec A na seznamu. Ta namreč pove, če je produkt aktiven. S tem potrditvenim poljem lahko onemogočimo izvajanje vseh modulov produkta. Zato moramo res paziti, da na produkt dodamo le tiste module, ki so smiselno povezani. Če v prihodnosti funkcionalnosti s teh modulov ne želimo več uporabljati, nam funkcijskih modulov ni potrebno izbrisati, ampak moramo samo deaktivirati produkt.

Ker želimo modul registrirati za objavi in naroči dogodek, moramo v transakciji FIBF slediti poti Nastavitve → P/S funkcijski moduli → stranke (slika 6.2). Navesti moramo dogodek, produkt in funkcijski modul, kot vidimo na sliki 6.5. Po shranitvi podatkov je naš dodatek za shranjevanje matičnih podatkov dobavitelja aktiven.



Slika 6.5: Registracija funkcijskega modula za dogodek poslovne transakcije

6.2 Prednosti

Tehnika dodatkov z BTE funkcijami je boljša od uporabniškega izhoda, saj lahko enostavno onemogočimo izvajanje kode po meri. SAP je to podprl z uporabo produktov, ki se jih da deaktivirati. Ni nam treba ročno brisati kode, ki je ne želimo več uporabljati.

Ta tehnika vsaj do neke mere odpravi tudi zelo veliko pomanjkljivost uporabniškega izhoda in izhoda stranke. Pri tej tehniki je lahko naša koda ločena od kode drugih razvijalcev, saj lahko kreiramo svoj funkcijski modul. Pri uporabniškem izhodu morajo vsi izvajalci uporabiti isto podrutino, pri izhodu stranke pa isti INCLUDE program. Priznati pa moramo, da je ta pomanjkljivost odpravljena samo pri objavi in naroči dogodkih, kjer lahko registriramo neomejeno število funkcijskih modulov. Pri procesnih dogodkih smo omejeni samo na en funkcijski modul, zato se še vedno lahko zgodi, da bomo primorani spreminjati funkcijski modul drugih razvijalcev.

6.3 Slabosti

Tehnika dodatkov se še vedno zanaša na proceduralno programiranje, saj uporablja funkcijske module - globalne funkcije.

Kot pri vseh tehnikah dodatkov do zdaj, lahko tudi tu dodajamo kodo po meri le na določenih delih v programu. Omejeni smo na tiste dele programa, kjer se prožijo dogodki poslovnih transakcij. Dostop pa imamo samo do podatkov iz programa, ki so posredovani preko vmesnika BTE funkcije. Če

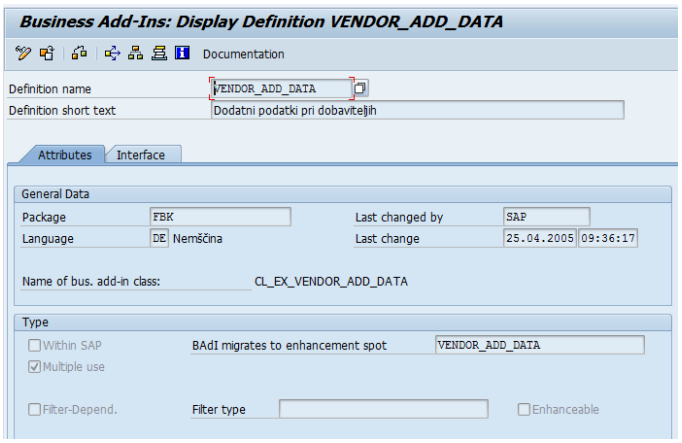
SAP razvijalci niso predvideli, da bomo v določeni točki programa morali dodati lastno logiko, ali pa če vmesnik ne vsebuje dovolj podatkov, potem se moramo zanesti na bolj napredne tehnike dodatkov, ki so opisane v naslednjih poglavjih.

Poglavje 7

BAdI

Z verzijo SAP ERP 4.7[7] je bila dodana nova tehnika dodatkov, ki se imenuje BAdI. Ime je sestavljeno iz angleške besedne zveze Business Add-Ins[7], kar bi po slovensko prevedli kot poslovni dodatki.

BAdI je prva tehnika dodatkov, ki se zanaša na objektno programiranje in ne več na proceduralno. Definicija BAdIja je narejena z vmesnikom, z implementacijo vmesnika pa lahko razširimo funkcionalnost kode [6]. Na sliki 7.1 lahko vidimo definicijo BAdIja VENDOR_ADD_DATA.



Slika 7.1: BAdI definicija

BAdI lahko implementiramo večkrat, če je obkljukan atribut za večkratno

uporabo (Multi use) ali če je implementacija BAdIja odvisna od filtra (atribut Filter Dependence). Pri večkratni uporabi se moramo zavedati, da je lahko aktivnih več implementacij, implementacije pa bodo poklicane po naključnem vrstnem redu[6].

Če je BAdI odvisen od filtra, potem lahko za vsako vrednost filtra naredimo svojo implementacijo BAdIja. Na primer, če je narejen filter za šifro podjetja, potem lahko za vsako šifro podjetja naredimo največ eno implementacijo. V primeru, da atributa za filter in večkratno uporabo nista obkljukana, je dovoljena največ ena implementacija BAdIja. [6]

Pomemben je tudi atribut, ki označuje, da je BAdI lahko implementiran le za interno SAP uporabo (Within SAP). Teh BAdIjev ne moremo implementirati.

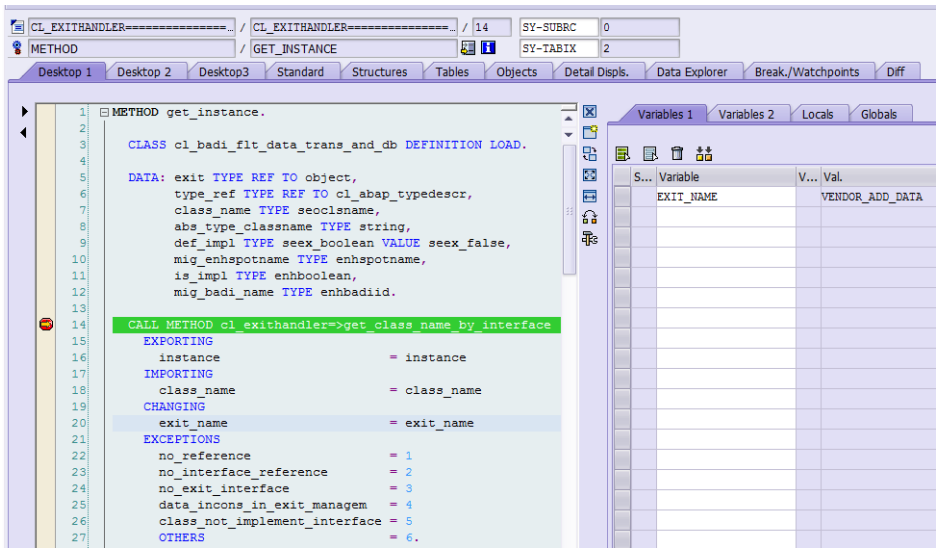
Poleg vmesnika lahko definicija BAdIja vsebuje tudi funkcijske kode in podekrane. S funkcijskimi kodami lahko dodamo dodatne možnosti na menije v programu, s podekrani pa dodatne podekrane na že obstoječe ekrane v programu. Čeprav lahko s pomočjo BAdIja razširimo tudi menije in ekrane, pa se bom v tem poglavju osredotočila na razširitev standardne kode s pomočjo vmesnika.

V kodi se aktivne implementacije BAdIjev poiščejo s pomočjo statične metode GET_INSTANCE v razredu CL_EXITHANDLER. Klic metode nam vrne instanco prilagoditvenega razreda (angl. adapter class [6]). Ko se pokliče metoda BAdIja z uporabo prilagoditvenega razreda, se bo poklicala ta metoda v vseh aktivnih implementacijah. Zavedati pa se moramo, da se bodo implementacije poklicale po naključnem vrstnem redu.

V metodi get_instance se za iskanje implementacij uporablja klic na bazo, kar upočasni aplikacijo, še posebej, če aplikacija uporablja veliko BAdIjev. Ta problem je SAP rešil z uporabo novih BAdIjev (poglavje 8.3), ki so del ogrodja dodatkov in si bili dodani v SAP ERP 6.0. Nove BAdIje imenujemo tudi jedrni BAdIji. BAdIje, ki jih opisujem v tem poglavju in ne uporabljajo ogrodja dodatkov, pa SAP imenuje klasični BAdIji. Nove BAdIje se ne išče več na nivoju podatkovne baze, ampak na nivoju jedra, kar pospeši iskanje.[7]

7.1 Iskanje BAdIjev

Klasične BAdIje lahko poiščemo na več različnih načinov. Najlažji način je, da damo prekinitveno točko na prvi izvedljiv stavek znotraj metode get_instance globalnega razreda CL_EXITHANDLER.

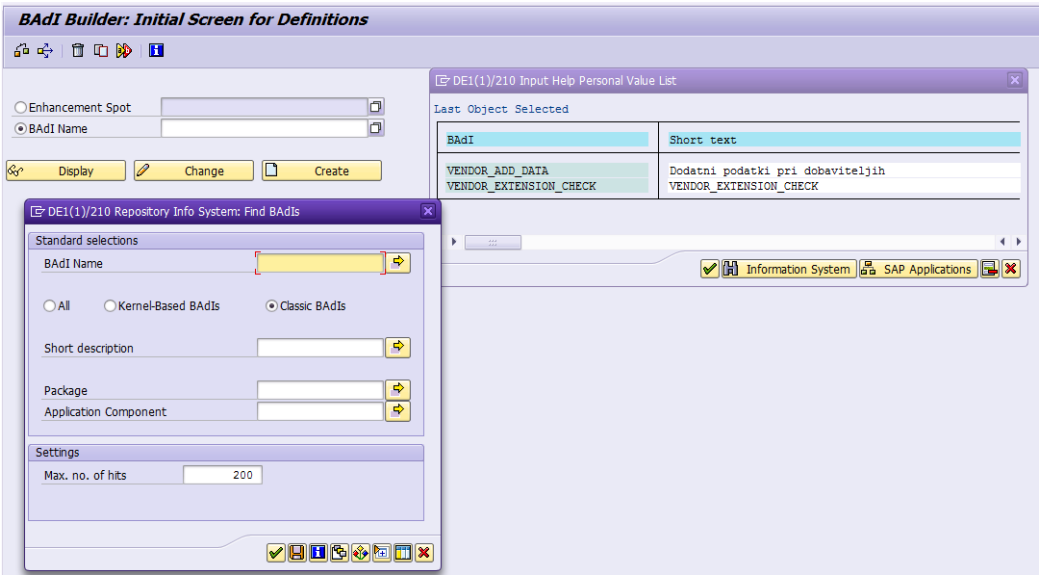


Slika 7.2: Iskanje BAdIjev z metodo get_instance

Ko zaženemo aplikacijo, se nam bo razhroščevalnik ustavil pri iskanju implementacije klasičnih BAdijev. Znotraj metode lahko preverimo vrednost spremenljivke EXIT_NAME, ki bo vsebovala ime BAdIja (slika 7.2).

BAdIje lahko poiščemo tudi preko transakcije SE18. To je transakcija, kjer lahko ustvarimo, spreminjamo in prikazujemo BAdI definicije. Klasičnih BAdI definicij ni mogoče ustvariti od SAP ERP 6.0 naprej, saj so bili takrat dodani novi BAdIji[7], zato lahko preko te transakcije ustvarimo le te. Kljub temu pa lahko še vedno iščemo in prikazujemo definicije klasičnih BAdIjev.

To storimo tako, da označimo izbirno tipko BAdI Name, postavimo kurzor na vnosno polje in pritisnemo F4 - gumb za pomoč pri iskanju (slika 7.3). Pojavi se nam okno, kjer so prikazani zadnji izbrani objekti. Če iskanega objekta ni med zadnjimi, potem izberemo gumb Information System. Poja-



Slika 7.3: Iskanje klasičnih BAdIjev preko transakcije SE18

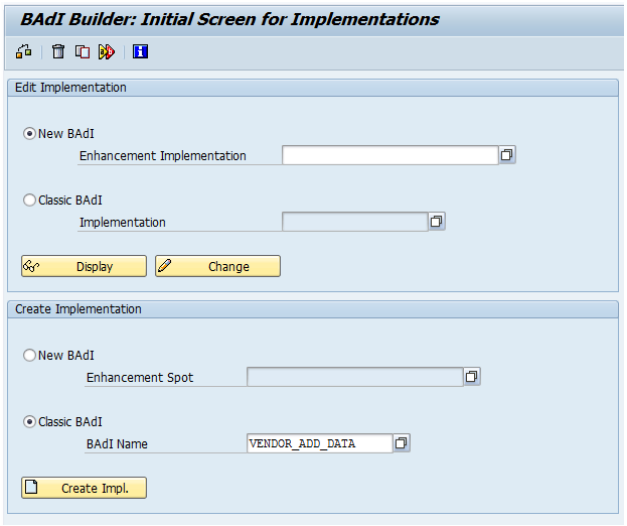
vilo se nam bo okno, ki ga vidimo na sliki levo spodaj. Če tu označimo izbirno tipko Classic BAdIs, bomo lahko iskali klasične BAdIje po imenu, opisu, paketu ali pa komponenti aplikacije. Ko bomo pritisnili na gumb za potrditev, se nam bodo prikazale vse definicije klasičnih BAdIjev, ki ustrezajo iskalnim kriterijem.

7.2 Primer uporabe

Primer uporabe bom prikazala na BAdIju VENDOR_ADD_DATA, katerega definicijo smo videli na sliki 7.1. S pomočjo tega BAdIja bom znotraj transakcije za matične podatke dobaviteljev dodala kontrolo, ki bo zahtevala vnos regije pri slovenskih dobaviteljih.

Za implementacijo te funkcionalnosti moramo najprej narediti implementacijo BAdIja preko transakcije SE19. To je transakcija, kjer lahko spreminjamo in ustvarjamo implementacije za klasične in nove BAdIje.

V transakciji označimo možnost Classic BAdI v okviru za kreiranje imple-



Slika 7.4: Iskanje klasičnih BAdIjev preko transakcije SE18

mentacije in v vnosno polje vpišemo ime BAdIja (slika 7.4). Nato kliknemo na gumb za kreacijo implementacije in vpišemo ime implementacije. Pri kreaciji implementacije moramo navesti še kratki tekst, s katerim opišemo implementacijo, nato pa lahko implementacijo aktiviramo.

Čeprav je BAdI VENDOR_ADD_DATA klasični BAdI, je bil migriran na mesto dodatka VENDOR_ADD_DATA, kar lahko vidimo na sliki 7.1. To pomeni, da se ta BAdI lahko uporablja na klasični in jedrni način. Kar pomeni, da ga lahko instanciramo z GET_INSTANCE metodo razreda CL_EXIT-HANDLER ali pa z ukazom GET BADI, s katerim se instancirajo novi BAdIji. Tudi metode BAdIja kličemo na klasični ali novi način: CALL METHOD ali CALL BADI. Kateri ukaz uporabimo je odvisno od tega, kako smo BAdI instancirali. Ker novi BAdIji uporabljajo nekatere komponente ogrodja za razširitve (mesto dodatka, definicija dodatka), moramo pri kreaciji tega migriranega BAdIja kreirati tudi implementacijo dodatka. Več o ogrodju dodatkov in njegovih komponentah, ki so bile v tem odstavku omenjene (mesto dodatka, definicija dodatka, implementacija dodatka, jedrni BAdI) si lahko preberete v poglavju 8.

Pri kreaciji implementacije se samodejno ustvari tudi razred, ki vsebuje vmesnik BAdIja. Z implementacijo metod v novonastalem razredu, bomo razširili standardno kodo. V našem primeru bomo uporabili metodo CHECK_ALL_DATA. Z uporabo te metode dodamo omejitev, da je pri slovenskem dobavitelju regija obvezen podatek.

```
METHOD if_ex_vendor_add_data~check_all_data.
  CHECK i_lfa1-land1 EQ 'SI' AND
        i_lfa1-regio IS INITIAL.

  e_msgid = 'ZMM_VENDOR'.
  e_msgno = '002'.
  e_dynnr = '0111'.
ENDMETHOD.
```

Ko metodo spremenimo, aktiviramo razred in implementacijo BAdIja, bi se morala implementacija BAdIja že uporabiti. Pri večini BAdIjev se tu naše delo konča. Vendar pa moramo BAdIje, ki se uporabljajo pri kreaciji oziroma spreminjanju matičnih podatkih dobavitelja, označiti kot aktivne še v standardnih SAP nastavitvah (angl. SAP reference IMG) preko transakcije SPRO. Ko BAdI označimo kot aktiven, aplikacija za dobavitelje zahteva vnos regije za slovenske dobavitelje.

7.2.1 Prednosti

Ta tehnika je ena boljših tehnik, saj je edina, poleg novega BAdIja, ki uporablja objektno programiranje. Če je BAdI označen za večkratno uporabo, ga je mogoče večkrat implementirati. Tako lahko to tehniko uporabimo tudi, kadar je isti BAdI pred nami uporabil že kdo drug. BAdI je tudi edina tehnika, ki omogoča uporabo filtra. Zato lahko implementiramo BAdI samo za določene vrednosti filtra.

7.2.2 Slabosti

Slabost te tehnike je tako kot pri vseh ostalih, z izjemo implicitnih dodatkov, da se lahko uporablja samo na tistih mestih, kjer je bilo to vnaprej predvideno. Znotraj metod, ki jih implementiramo, lahko dostopamo samo do tistih podatkov, ki so bili navedeni znotraj definicije vmesnika. Če tehnike ni tam, kjer jo potrebujemo ali pa nam metoda ponuja premalo podatkov, potem moramo žal uporabiti kakšno drugo tehniko dodatkov. Pri BAdIjih, ki niso označeni za večkratno uporabo, se nam lahko zgodi, da ga je implementiral že drug razvijalec pred nami in ga zato mi ne moremo več uporabiti.

Poglavje 8

Ogrodje dodatkov

Ogrodje dodatkov je bilo dodano v verziji SAP NetWeaver 7.0 [6]. Ogrodje podpira eksplicitne dodatke, implicitne dodatke in nove BAdIje, ki bodo opisani v naslednjih podpoglavjih. SAP predlaga uporabo najnovejših tehnik dodatkov, zato priporoča uporabo elementov iz Ogrodja dodatkov[7].

8.1 Eksplicitni dodatki

Eksplicitni dodatki so tiste dodatki, kjer lahko dodamo kodo po meri le v vnaprej predvidena mesta v programu. Poznamo dve vrsti eksplicitnih dodatkov: eksplicitne točke in eksplicitni odseki. Eksplicitne točke so točke v programu, kjer lahko dodamo lastno kodo.

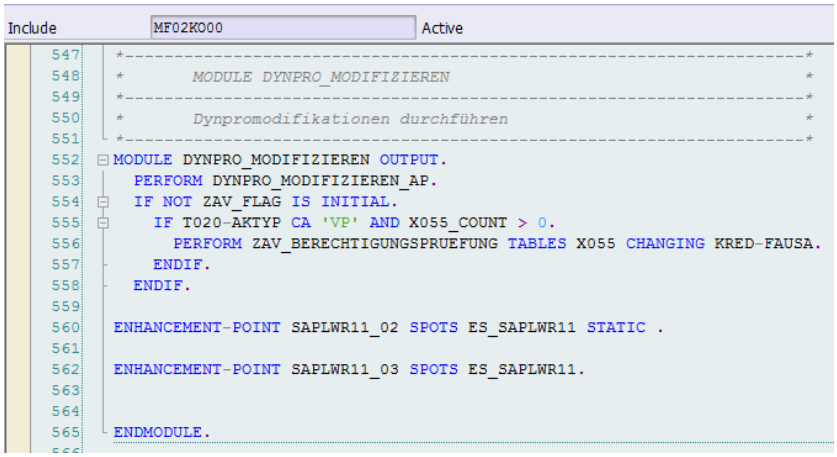
Točke so v kodi označene na spodnji način[3].

```
ENHANCEMENT-POINT <id> SPOTS <mesto> [STATIC] [INCLUDE BOUND].
```

Eksplicitni odseki se od točk razlikujejo v tem, da nadomestimo odsek standardne kode s svojo kodo. Torej ne samo da kodo dodamo, ampak tudi nadomestimo obstoječo kodo. Ukaz za eksplicitni odsek je podoben ukazu za točko. Začne se z ENHANCEMENT-SECTION, konča z END-ENHANCEMENT-SECTION znotraj pa vsebuje kodo, ki jo lahko nadomestimo[3].

```
ENHANCEMENT-SECTION <id> SPOTS <mesto> [STATIC] [INCLUDE BOUND].
* koda za nadomestitev
END-ENHANCEMENT-SECTION.
```

Eksplicitni dodatki so lahko statični ali dinamični, za kar je odgovorna opcijska ključna beseda STATIC. Statične se uporablja za najavo spremenljivk, v dinamične pa dodamo izvršljivo kodo[7]. Primer statične in dinamične eksplicitne točke vidimo na sliki 8.1.



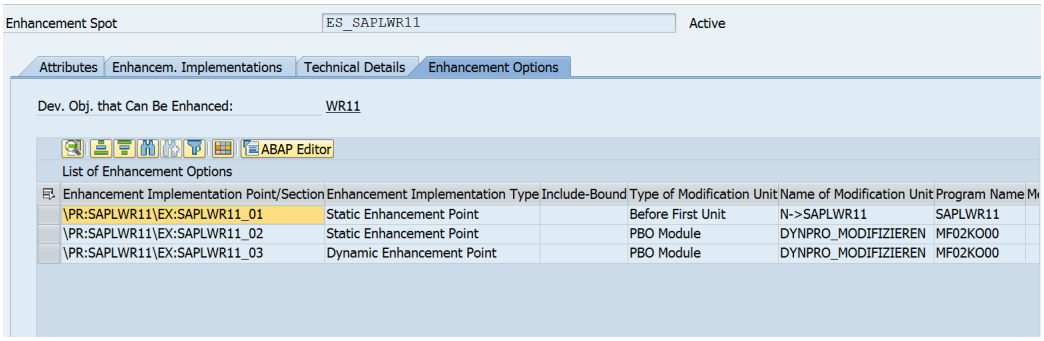
Slika 8.1: Statična in dinamična eksplicitna točka

Id dodatka mora biti unikatno ime znotraj trenutne enote za prevajanje. V programskem jeziku ABAP poznamo naslednje enote za prevajanje: izvršljivi programi, skupni fond modulov, funkcijske skupine, skupni fond razredov, skupni fond vmesnikov, skupni fond podrutin in skupni fond tipov. Vsebnik ni samostojna enota za prevajanje, zato obstaja ključna besedna zveza INCLUDE BOUND. To besedno zvezo uporabimo, kadar želimo, da je id dodatka unikaten znotraj vsebnika. Vsak id dodatka pripada vsaj enemu mestu dodatka[3].

Mesto dodatka vsebuje pomembne informacije o dodatkih, s katerimi upravlja [1], kar lahko vidimo na sliki 8.2. Vidimo lahko, kje v kodi se pripadajoči dodatki nahajajo. Vidimo tudi ali gre za statični/dinamični eksplicitni dodatek ali pa nov BAdI. Če gre za eksplicitni dodatek, bo tudi pisalo

ali gre za točko ali odsek, ter če je bila uporabljena besedna zveza INCLUDE BOUND.

Znotraj mesta dodatka lahko na zavihku Implementacije dodatkov vidimo vse obstoječe implementacije.



Slika 8.2: Mesto dodatka - podatki o pripadajočih dodatkih

Mesta dodatka so lahko enostavna ali sestavljena. Enostavno mesta lahko vsebujejo samo eno vrsto dodatkov: eksplicitne dodatke ali pa nove BAdIje. Sestavljena mesta dodatkov lahko vsebujejo enostavna in/ali druga sestavljena mesta. Sestavljena mesta uporabljamo, da skupaj logično povežemo različna mesta dodatkov in tako tvorimo tudi drevesno strukturo. V tej drevesni strukturi so listi enostavna mesta, vozlišča pa sestavljena mesta [1].

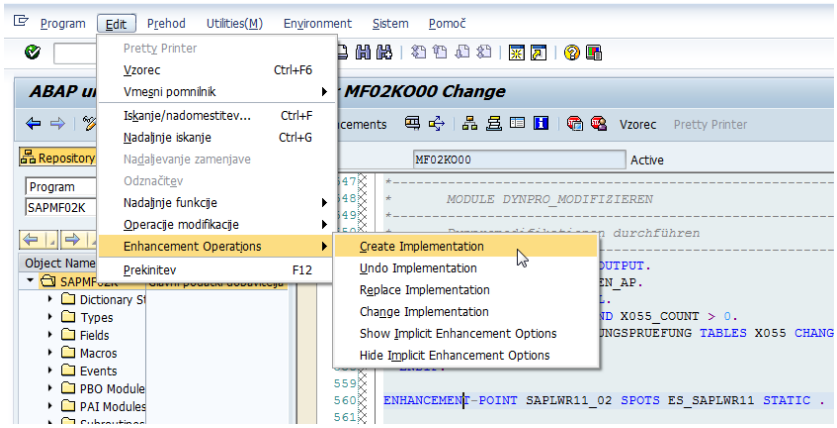
8.1.1 Primer uporabe

Recimo, da želi stranka znotraj aplikacije za urejanje matičnih podatkov dobaviteljev (transakcije XK01, XK02, XK03) navesti dodatne podatke nabave. To zahtevo lahko rešimo tako, da preko standardnih SAP nastavitev (transakcija SPRO) dodamo nov gumb Dodatni podatki, ki je viden znotraj standardne aplikacije za dobavitelje. Po kliku na gumb se odpre novo okno, kamor se lahko vnesejo dodatni nabavni podatki. Vendar stranka želi, da je gumb viden samo na ekranu Podatki nabave. Željo stranke lahko uresničimo z eksplicitnim dodatkom dinamične točke, ki smo jo videli na sliki 8.1. Z implementacijo te točke bomo na drugih ekranih gumb skrili.

Zahtevo začnemo reševati tako, da v ABAP urejevalniku odpremo del kode, kjer se eksplicitni dodatek nahaja. V našem primeru je to vsebnik MF02KO00. Nato kliknemo na gumb, s katerim v urejevalniku kode omogočimo upravljanje z dodatki (slika 8.3).

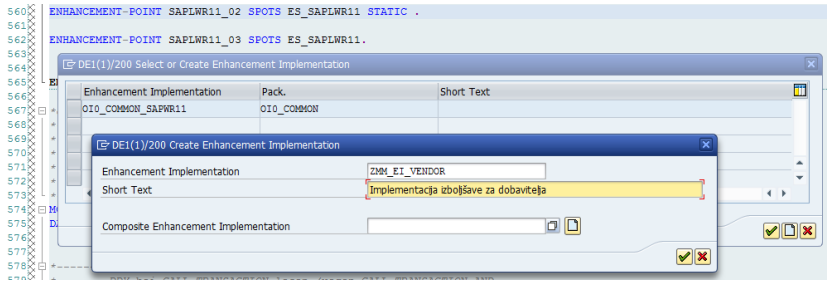
Slika 8.3: Gumb za dodatke

Kurzor postavimo na eksplicitni dodatek v kodi ter sledimo poti v meniju: Edit → Enhancement Operations → Create Implementation (slika 8.4).



Slika 8.4: Kreacija implementacije za eksplicitno točko

Prikaže se nam pojavno okno, kjer lahko izberemo obstoječo implementacijo dodatka ali pa kreiramo novo. Ker še nimamo obstoječe implementacije, izberemo gumb za kreacijo. Vpisati moramo ime in opis dodatka. Kot mesta dodatkov so tudi implementacije lahko enostavne ali sestavljene, ter tako tvorijo drevesno strukturo[1]. Če želimo, da je implementacija del sestavljene implementacije, potem moramo tu vpisati še njeno ime. Pri kreaciji implementacije sem vpisala podatke, ki jih vidite na sliki 8.5.



Slika 8.5: Vnos osnovnih podatkov implementacije

Ko je implementacija kreirana, se pojavi na seznamu implementacij in jo lahko izberemo. Po potrditvi se nam samodejno kreira del kode, ki ga vidimo spodaj.

```
ENHANCEMENT <id> <ime_implementacije>.

ENDENHANCEMENT.
```

Ta del kode se doda takoj za točko dodatka, ki jo želimo razširiti. Id dodatka je predstavljen z zaporedno številko in je unikaten za implementacijo dodatka. Pri točki dodatka je del med ključnima besedama ENHANCEMENT in ENDENHANCEMENT prazen in tu lahko zdaj dodamo kodo po meri, ter tako izboljšamo standardni SAP program.

Gumb je na začetku označen kot neaktiven in ni prikazan na nobenem ekranu. Ker lahko znotraj eksplicitnih dodatkov dostopamo do vseh spremenljivk, ki so vidne na tistem delu kode, lahko spreminjamo globalne spremenljivke programa, ki skrbijo za prikazovanje oziroma skrivanje gumbov ter dodatnih ekranov. Zato sem znotraj implementacije napisala kodo, ki gumb in dodatni ekran omogoči samo na ekranu Dodatni podatki nabave, drugače pa ju skrije. Ko sem kodo dodala, je bilo dodatek potrebno samo še aktivirati, nato pa so bile spremembe vidne v aplikaciji za dobavitelje.

8.1.2 Prednosti

Z eksplicitnimi dodatki lahko kodo po meri dodajamo direktno v standardno kodo, vendar nam za to ni potrebno uporabiti modifikacije, torej ne potrebujemo ključa za dostop. Ker dodajamo kodo direktno, ne pa v podrutino, funkcijski modul ali znotraj metod, lahko dostopamo do vseh globalnih in lokalnih spremenljivk na tistem delu programa.

Ker so eksplicitni dodatki del ogrodja dodatkov, uporabljajo mesta dodatkov za definicijo in implementacijo. Tako lahko implementacije eksplicitnih dodatkov skupaj logično povežemo v drevesno strukturo, kar jih naredi bolj pregledne.

8.1.3 Slabosti

Ker lahko spreminjamo globalne in lokalne spremenljivke, je pri implementaciji dodatka potrebna pazljivost. Tudi tu smo odvisni od razvijalcev standardnega SAP programa, da so predvideli potrebo za dodatek, ter v kodo dodali točko ali odsek dodatka.

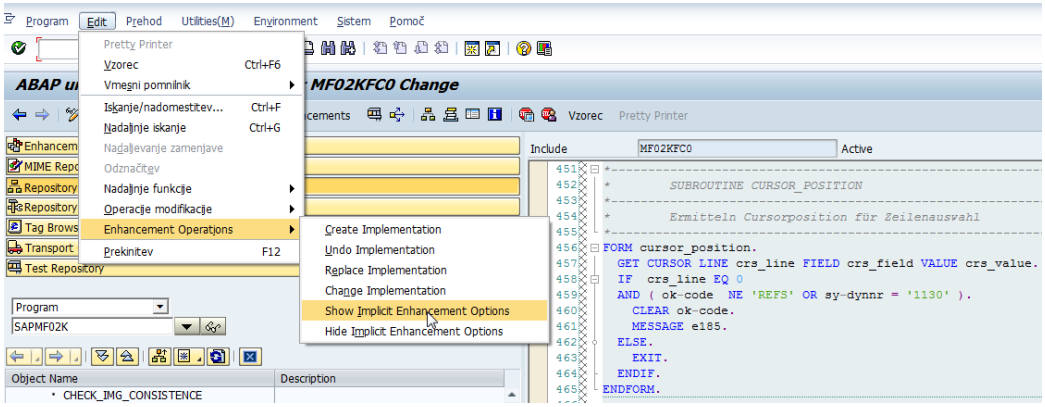
8.2 Implicitni dodatki

Naslednja vrsta dodatkov, ki si jih bomo ogledali, so implicitni dodatki. Če so eksplicitni dodatki postavljeni na točno določenih mestih v programu, pa smo pri uporabi implicitnih dodatkov veliko bolj svobodni. Nismo več odvisni od tega, da so razvijalci predvideli, kje bomo želeli narediti dodatek, ampak lahko naredimo dodatek sami. Čeprav lahko implicitne dodatke uporabimo na veliko mestih v kodi, jih ne moremo uporabiti kjerkoli. Spodaj so opisana mesta v kodi, kjer jih lahko uporabimo:

- na začetku ali koncu procedur (funkcijskega modula, metod, podrutin),
- za ključno besedo ENHANCEMENT in pred ključno besedo ENDENHANCEMENT,

- na koncu programov, funkcijskih skupin, skupnih fondov podrutin, skupni fond modulov, INCLUDE programov,
- v lokalnem razredu na koncu odsekov vidljivosti pri deklaraciji spremenljivk in metod,
- v lokalnih metodah na koncu seznama uradnih parametrov iste vrste,
- pri definiciji struktur z uporabo ključnih besed BEGIN OF in END OF, pred besedno zvezo END OF[3].

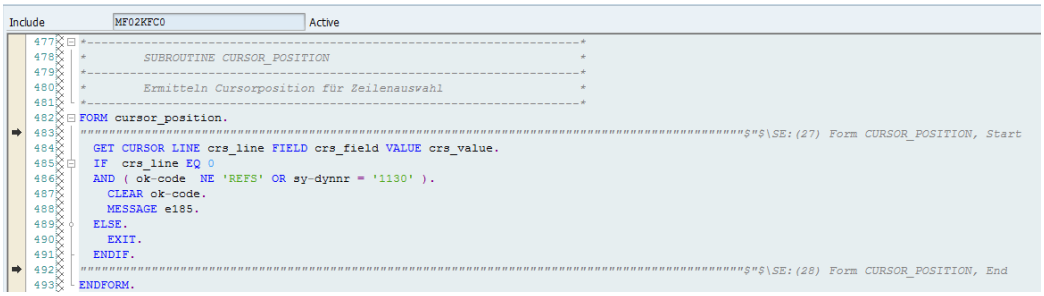
Če želimo videti, kje v kodi lahko uporabimo implicitni dodatek, najprej kliknemo na gumb za dodatke (slika 8.3), potem pa sledimo poti v meniju: Edit → Enhancement Operations → Show Implicit Enhancement Options (slika 8.6).



Slika 8.6: Prikaz možnosti za implicitne dodatke

Prikazale se bodo vse možnosti za implicitne dodatke, ki jih lahko uporabimo. Mesta, kjer je implicitni dodatek mogoč, bodo označena s komentarji, kot jih lahko vidimo na sliki 8.7. Opazimo, da so se komentarji pojavili na začetku in na koncu podrutine.

Pri uporabi implicitnih dodatkov moramo paziti, kakšno kodo pišemo. Kjerkoli dodajamo kodo, mora biti ta sintaktično pravilna. Ne moremo na



Slika 8.7: Prikaz možnih implicitnih dodatkov v podrutini

primer dodati podrutine v lokalnem razredu na koncu odseka vidljivosti pri deklaraciji spremenljivk in metod [7].

Ker niso vnaprej predvidene, ne poznamo eksplicitnih odsekov, kjer bi se koda nadomestila. Če želimo nadomestiti kodo z uporabo implicitnega dodatka, je najboljša ideja, da uporabimo implicitni dodatek na začetku procedure (funkcijskega modula, podrutine, metode), znotraj dodatka napišemo lastno kodo, na koncu katere dodamo ukaz za izhod iz procedure (ukaz RETURN). S tem bomo dejansko nadomestili kodo, ki je znotraj procedure, saj so bo izvedla samo naša lastna koda, standardna koda pa se zaradi ukaza za izhod ne bo nikoli izvedla.

Pri implicitnih dodatkih se tudi ne uporablja mest dodatkov, kot pri eksplicitnih [6]. Implementacija dodatka tako ne pripada točno določenemu mestu dodatka. Znotraj implementacije je zato samo zapisana lokacija implicitnega dodatka, brez mesta dodatka.

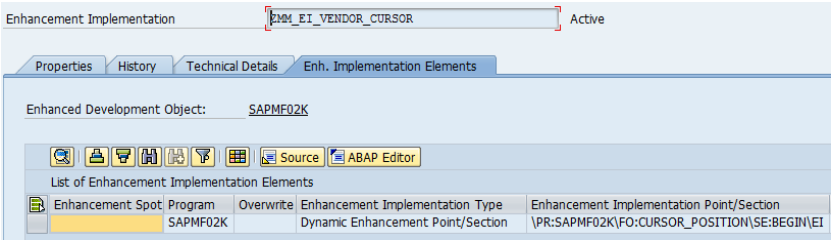
8.2.1 Primer uporabe

Primer uporabe bom prikazala na implicitnem dodatku, ki smo ga videli na sliki 8.7 in se nahaja na začetku podrutine CURSOR_POSITION. Implicitne dodatke se uporablja zelo podobno kot eksplicitne. Najprej kliknemo na gumb za dodatke (slika 8.3), nato moramo prikazati implicitne dodatke (slika 8.7) in postaviti kurzor na dodatek na začetku podrutine. Izberemo pot na meniju Edit → Enhancement Operations → Create Implementation

(slika 8.4).

Tu se nam pojavi okno, kjer moramo definirati, ali želimo kreirati statični ali dinamični dodatek. Povedati moramo, ali bomo dodatek uporabili za deklaracijo spremenljivk, tipov, itd., ali pa za izvršljivo kodo. V našem primeru bomo ustvarili dinamično implementacijo.

Kot pri eksplicitnih dodatkih, se tudi tu pojavi okno, kjer moramo ustvariti oz. izbrati primerno implementacijo dodatka (slika 8.5). Ustvarila sem implementacijo ZMM_EI_VENDOR_CURSOR (slika 8.8). Na sliki vidimo, da je mesto dodatka prazno, saj se pri implicitnih dodatkih ne uporablja. Vidimo tudi, kje se dodatek nahaja in, da je implementacija dinamična.



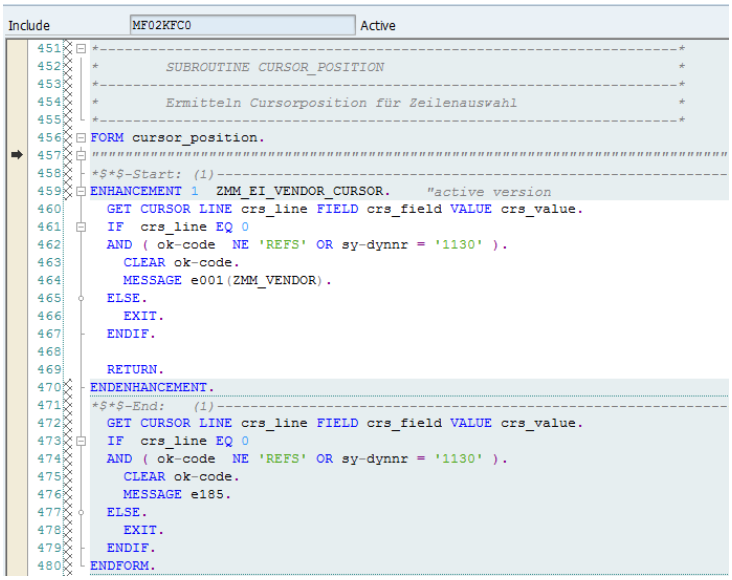
Slika 8.8: Implementacija implicitnega dodatka

Ko izberemo implementacijo, se nam samodejno kreira blok kode, ki se začne z ukazom ENHANCEMENT in konča z ENDENHANCEMENT. Znotraj sem dodala kodo po meri, ki jo vidite na sliki 8.9. Z uporabo tega dodatka sem nadomestila standardno kodo, saj sem na koncu uporabila ukaz za vrnitev iz podrutine (RETURN). Koda po meri bo naredila enako kot standardna koda, le da se bo ob napaki prikazalo naše lastno opozorilo, ki je drugačno od standardnega.

Na koncu je potrebno dodatek samo še aktivirati, kar storimo z gumbom za aktivacijo dodatkov.

8.2.2 Prednosti

Od vseh tehnik dodatkov, ki jih imamo na voljo, smo pri tej tehniki najbolj svobodni. To je edina tehnika dodatkov, kjer lahko dodamo kodo po meri



Slika 8.9: Nadomestitev standardne kode z imp. dodatkom

na mesta v programu, ki niso bila posebej predvidena za to s strani SAP razvijalca, ampak so tam privzeto.

Tudi tu lahko dodajamo kodo direktno v standardno SAP kodo in pri tem ne uporabimo modifikacije. Različne implementacije implicitnih dodatkov lahko logično urejamo v drevesno strukturo z uporabo enostavnih in sestavljenih mest implementacij dodatkov.

8.2.3 Slabosti

Tudi tu lahko dostopamo do vseh globalnih in lokalnih spremenljivk programa, funkcijskega modula ali razreda, zato moramo biti pri uporabi te tehnike pazljivi.

Velika slabost te tehnike pa je, da lahko implicitni dodatek implementiramo samo enkrat. Večkratna implementacija pri tej tehniki ni mogoča. Če je nekdo implementiral dodatek pred nami, ga ne bomo mogli uporabiti.

8.3 Novi BAdIji

Novi BAdIji so zelo podobni klasičnim BAdIjem, ki so opisani v poglavju 7. Od klasičnih se ločijo v dveh pomembnih točkah:

1. Nastali so, saj so želeli izboljšati klasične BAdIje. Klasični BAdIji za iskanje implementacij uporabljajo klic na bazo, kar upočasni aplikacijo. Zato so pri novih BAdIjih iskanje premaknili v jedro, kar ni tako časovno zahtevno. Nove BAdIje zato imenujemo tudi jedrni BAdIji[7].

2. Ker so del ogrodja dodatkov, vsaka definicija/implementacija BAdIja pripada mestu dodatka. Vsako mesto dodatka lahko vsebuje več definicij/implementacij BadIjev, ki so tako logično povezane skupaj. Tudi tu ločimo enostavna mesta dodatkov in sestavljena mesta, ki skupaj tvorijo drevesno strukturo.

Implementacije novih BAdIjev se med izvajanjem poiščejo z ukazom GET BADI. S tem ukazom pridobimo BAdI oprimek (angl. handle)[6], s katerim lahko potem izvajamo metode, ki so v BAdIju definirane.

```
GET BADI badi [FILTERS f1 = x1 f2 = x2 ...]
              [CONTEXT con].
```

Če BAdI uporablja filter, je pri klicu potrebno navesti še parametre filtra[3].

Pri definiciji BAdIja lahko označimo, kako se oprimek instancira. Lahko se odločimo, da se instanca vedno znova kreira. Druga možnost je, da dovolimo ponovno uporabo že obstoječe instance. Zadnja možnost, ki jo imamo na voljo, pa je uporaba konteksta. Če pri izvedbi ukaza GET BADI uporabimo isti kontekst, se bo instanca oprimka ponovno uporabila, drugače pa se bo kreirala nova instanca[7].

Metode vmesnika, ki ga uporablja BAdI, se pokličejo z ukazom CALL BADI, kjer navedemo še ustrezne parametre. Če je BAdI označen za večkratno uporabo, potem se bo pri klicu metode z BAdI oprimkom poklicala metoda v vseh implementacijah tega BAdIja.

```
CALL BADI badi->meth    [EXPORTING  p1 = a1 p2 = a2 ...]
                      { {[IMPORTING  p1 = a1 p2 = a2 ...]
                        [CHANGING   p1 = a1 p2 = a2 ...]}
                      |  [RECEIVING   r = a  ] }
                        [EXCEPTIONS [exc1 = n1 exc2 = n2 ...]
                                    [OTHERS = n_others]].
```

8.3.1 Primerjava klasičnih in novih BAdIjev

Ker je implementacija novega BAdIja podobna implementaciji migriranega BAdIja, ki smo si jo ogledali v poglavju 7.2, sem se odločila, da bom v tem poglavju namesto primera uporabe raje naredila primerjavo med klasičnimi in novimi BAdIji. Glavna razlika med klasičnimi in novimi BAdIji je način iskanja implementacij. Klasični BAdIji iščejo implementacije s klicem na bazo, medtem ko novi BAdI to nalogo opravijo v jedru. Ker naj bi bili novi BAdIji zaradi tega hitrejši, sem se odločila raziskati, ali so res hitrejši in grafično prikazati koliko hitrejši so od klasičnih BAdIjev.

Napisala sem program, v katerem sem primerjala hitrosti inicializacije prilagoditvenega razreda in inicializacije BAdI oprimka. Da bi bila primerjava čimbolj pravična, sem želela primerjati klasični in novi BAdI, ki sta si čimbolj podobna. Zato sem želela, da uporabljata isti vmesnik. Ker od uvedbe novih BAdIjev s SAP ERP verzijo 6.0 ni več mogoče kreirati definicije klasičnega BAdIja, sem morala uporabiti klasični BAdI, ki že obstaja. Odločila sem se za uporabo BAdIja BUPA_BANK_CHECK, ki uporablja vmesnik IF_EX-_BUPA_BANK_CHECK.

Ko sem izbrala klasični BAdI, sem morala definirati še nov BAdI. Želela sem uporabiti isti vmesnik, kot ga uporablja klasični BAdI, vendar morajo vse definicije novih BAdIjev imeti vmesnik, znotraj katerega je vmesnik IF_BADI_INTERFACE. Zato sem naredila nov vmesnik, ki vsebuje oba vmesnika (IF_EX_BUPA_BANK_CHECK, IF_BADI_INTERFACE), ter ga uporabila za definicijo novega BAdIja.

Pri definiciji novega BAdIja sem označila, da se mora oprimek vedno

znova instancirati. Oprimek se ne sme ponovno uporabiti ali biti odvisen od konteksta, saj potem primerjava ne bi bila poštena.

Ker je klasični BAdI označen za večkratno uporabo, sem tako označila tudi nov BAdI. Ker sta zdaj oba lahko večkrat implementirana, se mi je pri izvajanju poskusa zdelo pomembno tudi, da imata enako število aktivnih implementacij. Nisem želela, da bi različno število aktivnih implementacij vplivalo na rezultate. Ob času izvajanja poskusa sta zato oba imela eno aktivno implementacijo.

V programu, ki sem ga napisala, izberemo naslednje parametre izvajanja:

1. Število ponovitev: število ponovitev nam pove kolikokrat izvedemo meritev za določeno število instanc. Rezultat je nato povprečje teh meritev.

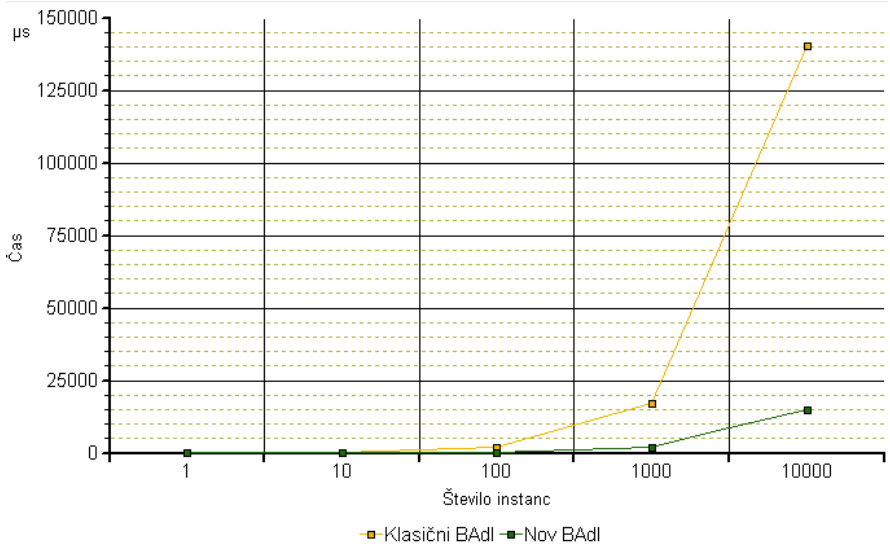
2. Največje število instanc: število instanc, za katero še izvedemo meritev oziroma pri kateri nehamo izvajati meritve.

3. Red velikosti: meritve se začnejo izvajati za eno instanco. Nato se v vsakem koraku število instanc pomnoži z redom velikosti. Ko je število instanc enako ali večje od največjega števila instanc, se program neha izvajati.

Program prikaže rezultate meritev v tabelarični in grafični obliki. Program sem izvedla tako, da je vsako meritev ponovil 10-krat, največje število instanc je bilo 10.000, za red velikosti pa sem izbrala 10. V tabeli 8.1 lahko vidimo meritve, ki jih je program izvedel. Čas v tabeli je zapisan v mikrosekundah. Iz teh meritev lahko vidimo, da je inicializacija novega BAdIja v grobem 10-krat hitrejša. Glede na izmerjene čase lahko sklepamo, da pri malem številu instanc uporabnik ne bo zaznal razlike. Do občutne razlike bi prišlo, če bi v programu instancirali res veliko število instanc. Isto primerjavo si lahko ogledate še v grafični obliki na sliki 8.10.

Št. instanc	Klasični BAdI	Nov BAdI
1	162	5
10	374	30
100	2.001	212
1.000	17.041	1.812
10.000	140.295	14.791

Tabela 8.1: Primerjava hitrosti inicializacije



Slika 8.10: Grafična primerjava hitrosti inicializacije

8.3.2 Prednosti in slabosti

Prednosti in slabosti so enake, kot so pri klasičnih BAdIjih. Opisala sem jih v poglavju 7.2.1 in poglavju 7.2.2. Kljub temu pa moram omeniti, da je jedrni BAdI vseeno boljši od klasičnega BAdIja. Je hitrejši, kar sem pokazala v poglavju 8.3.1. Poleg tega pa je del ogrodja dodatkov, torej uporablja mesta dodatkov, s katerimi lahko skupaj povezujemo BAdI definicije ali implementacije v drevesno strukturo. S tem poskrbimo za boljši pregled nad BAdI definicijami in implementacijami.

Poglavje 9

Uporaba tehnik dodatkov v aplikacijah Akademike d.o.o.

V podjetju Akademika d.o.o. imamo veliko aplikacij, ki so bile razvite znotraj podjetja in se opirajo na standardne SAP programe in dokumente. Pri razvoju aplikacij smo morali uporabiti različne tehnike dodatkov, da smo lahko prilagodili standardno SAP kodo, kjer smo to potrebovali zaradi funkcionalnosti aplikacij. V tem poglavju se bom osredotočila na šest aplikacij in predstavila katere tehnike dodatkov smo uporabili pri posamezni aplikaciji in razloge, zakaj smo se odločili za točno določene tehnike dodatkov. Aplikacije, ki si jih bomo pogledali so: Potrjevanje računov, Poslovni partnerji, Dokumenti v nabavi, Prodajni dokumenti, Osnovna sredstva, Službene poti in odsotnosti.

9.1 Analiza uporabljenih tehnik dodatkov

V tem podpoglavju bomo analizirali, katere tehnike dodatkov so bile uporabljene pri posameznih aplikacijah in razloge za uporabo. Pogledali si bomo tudi, zakaj smo se nekaterim tehnikam dodatkov izognili pri vseh aplikacijah.

Na sliki 9.1 je prikazana uporaba dodatkov. Prva stvar, ki jo lahko opazimo je, da je bila pri vseh aplikacijah uporabljena vsaj ena tehnika dodatkov.

To ni presenetljivo, saj so tehnike dodatkov zelo uporabno orodje, s katerimi lahko vplivamo na kodo, na katero ne moremo vplivati z modifikacijami, saj za to ponavadi nimamo avtorizacije.

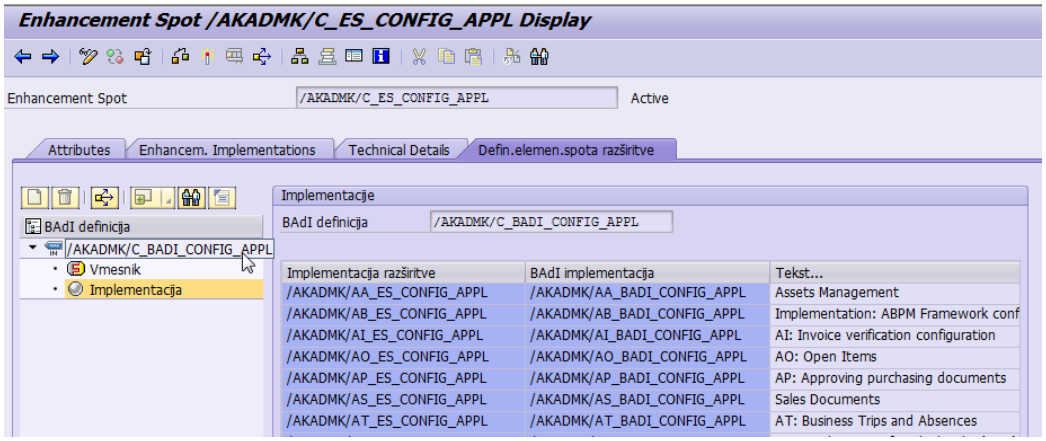
Aplikacija\Tehnika dodatkov	Up. izhod	Izhod stranke	BTE funkcije	Klasični BAdI	Eksplicitne d.	Implicitne d.	Kernel BAdI
Potrjevanje računov			✓	✓	✓		✓*
Poslovni partnerji							✓*
Dokumenti v nabavi				✓	✓		✓
Prodajni dokumenti					✓		✓*
Osnovna sredstva							✓*
Službene poti in odsotnosti							✓*

Slika 9.1: Uporabljene tehnike dodatkov

Pri nobeni aplikaciji nismo uporabili naslednjih tehnik: uporabniški izhod, izhod stranke in implicitni dodatki. Razlog za to je zelo preprost. V aplikacijah ne smemo uporabiti tehnik dodatkov, ki se jih lahko uporabi samo enkrat, saj moramo te tehnike pustiti na razpolago stranki. Uporabimo lahko samo tehnike, ki so namenjene za večkratno uporabo. Enkratne dodatke zato uporabimo samo, ko programiramo direktno na razvojnem SAP sistemu pri stranki. Na primer, če razvijamo aplikacijo po meri za stranko ali dodajamo funkcionalnosti njihovim obstoječim aplikacijam.

Na sliki lahko opazimo, da je pri vseh aplikacijah uporabljena tehnika dodatkov novi BAdI, pri večini pa je zraven zapisana zvezdica. Prisotnost novih BAdIjev sem označila z zvezdico tam, kjer jih v aplikacijah sicer uporabljamo, vendar uporabljamo samo BAdIje, ki smo jih definirali sami v domeni podjetja Akademika d.o.o. Nov BAdI, ki se uporablja v vseh aplikacijah, je prikazan na sliki 9.2 in se imenuje /AKADMK/C_BADI_CONFIG_APPL. BAdI se kliče v programu za konfiguracijo Akademika aplikacij in projektov. Z implementacijo BAdIja lahko vplivamo na registracijo aplikacij, posodobitev projektov in tako dalje. Čeprav v večini aplikacijah nismo uporabili novih BAdIjev, ki so bili definirani znotraj standardnih SAP domen, pa to ne pomeni, da smo se zavestno izogibali uporabe novih BAdIjev. Vedno

uporabimo tisto tehniko dodatkov, ki je na voljo na tistem delu programa, kjer jo potrebujemo. Kot sem že omenila v prejšnjem odstavku, pri tem ne uporabimo enkratnih tehnik dodatkov. Če bi bila na primernem delu v standardnem SAP programu na voljo tehnika dodatkov novi BAdI, bi jo seveda uporabili, vendar se pri razvoju teh aplikacij to ni zgodilo. Če imamo na nekem delu programa na voljo več različnih tehnik dodatkov, se odločimo za novejšo tehniko. SAP vedno priporoča uporabo najnovejših tehnik dodatkov [7].



Slika 9.2: Implementacije BAdIja /AKADMK/C_BADI_CONFIG_APPL

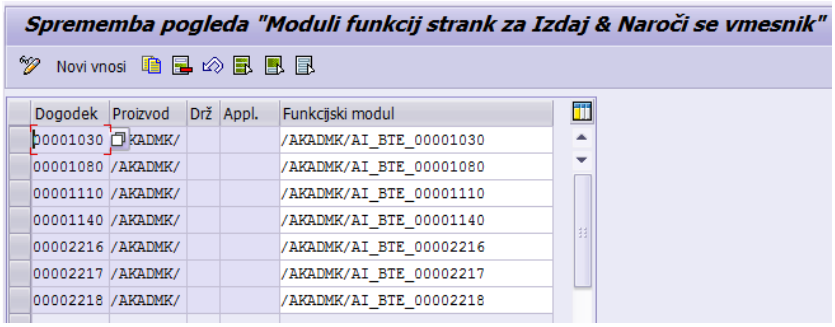
9.1.1 Potrjevanje računov

S to aplikacijo potrjujemo prejete vhodne račune. Postopek se začne, ko v podjetju prejmejo sliko računa. Proces potrjevanja računa ponavadi teče po naslednjem vzorcu. Najprej računovodstvo vnese osnovne podatke o računu na naš dokument. Ko so podatki vnešeni, se račun predhodno vnese (parkira), kar pomeni, da se osnovni podatki računa prenesejo v standardno SAP transakcijo in shranijo. Nato gre dokument čez proces potrjevanja, kjer se naredi tudi stroškovna razporeditev. Ko je račun potrjen, se uporabi akcija za knjiženje. Ob kliku na akcijo za knjiženje se stroškovna razporeditev z našega dokumenta prenese v standardno SAP transakcijo za knjiženje, kjer

lahko računovodstvo račun poknjiži. Postopek se ponavadi konča, ko je račun poknjižen.

Ker je postopek potrjevanja računov tesno povezan s standardnimi SAP transakcijami, je bilo treba pri razvoju uporabiti kar nekaj tehnik dodatkov. Uporabljene so bile BTE funkcije, klasični BAdIji in eksplicitni dodatki.

Ker aplikacije za potrjevanje računov uporablja finančni modul, smo tu lahko uporabili BTE funkcije. Večina BTE funkcij, ki je bilo uporabljenih, ima objavi in naroči vmesnik. Ta vmesnik omogoča, da lahko za isti dogodek poslovne transakcije registriramo več kot eno funkcijo. Na sliki 9.3 vidimo, da so bili za aplikacijo registrirani funkcijski moduli za naslednje dogodke poslovnih transakcij: 1030, 1080, 1110, 1140, 2216, 2217, 2218. Ti funkcijski moduli skrbijo za sinhronizacijo podatkov med standardnimi SAP transakcijami za parkiranje/knjiženje računov in našim dokumentom.



Slika 9.3: Registrirane BTE funkcije pri Potrjevanju računov

Čeprav sem napisala, da ne uporabljamo tehnik dodatkov, ki jih je mogoče implementirati samo enkrat, pa smo pri aplikaciji za potrjevanje računov naredili izjemo. Registrirali smo funkcijski modul za dogodek poslovne transakcije 1420 s procesnim vmesnikom. S funkcijskim modulom skrbimo, da se v standardni transakciji prikaže stransko okno s sliko računa. Prav tako z modulom poskrbimo, da omogočimo/onemogočimo določene opcije obdelave oziroma gumbe v standardnih SAP transakcijah. Kljub temu, pa smo želeli, da stranke ne izgubijo lastne funkcionalnosti, če so imele registriran

svoj funkcijski modul za isti dogodek. Po inštalaciji naše aplikacije so bo njihov modul še vseeno izvedel. V našem funkcijskem modulu namreč najprej preverimo, če ima stranka registriran svoj modul. Če ga ima, najprej izvedemo njihov modul, šele nato nadaljujemo z izvajanjem našega.

BTE funkcije smo lahko uporabili pri finančnih računih. Za logistične račune pa smo uporabili klasični BAdI INVOICE_UPDATE. V razredu implementacije smo redefinirali metodi CHANGE_AT_SAVE in CHANGE_BEFORE_UPDATE. Z metodami smo poskrbeli za sinhronizacijo podatkov ter omogočili ali onemogočili gumbe v standardni transakciji.

Uporabljena je bila tudi eksplicitna točka TRANSAKTIONS_INIT_01 za eksplicitno mesto ES_SAPLMR1M. Znotraj implementacije se pripravi stransko okno, da se lahko v standardni SAP transakciji prikaže slika računa.

9.1.2 Poslovni partnerji

Aplikacija skrbi za upravljanje z dobavitelji. Preko aplikacije lahko ustvarjamo, spreminjamo dobavitelje, jih razširjamo za določeno šifro podjetja ali nabavno organizacijo. Lahko jih tudi označimo za blokiranje/brisanje ali pa jim to oznako odstranimo. Postopek se začne z vnosom osnovnih podatkov dobavitelja in izbiro akcije, ki jo na dobavitelju želimo izvesti - kreiranje, sprememba, brisanje, itd. Ko so podatki vnešeni, gre dokument v potrjevanje. Če je zahteva za kreacijo oz. spremembo dobavitelja odobrena, se na koncu podatki dobavitelja prenesejo v standardno SAP aplikacijo za upravljanje z matičnimi podatki dobavitelja, dokument pa se označi kot končan.

V tej aplikaciji nismo uporabili nobene tehnike dodatkov, razen lastnega novega BAdIja za konfiguracijo aplikacije. Tehnik dodatkov nismo uporabili, saj ta aplikacija ni tako tesno povezana s standardnimi SAP transakcijami, kot je aplikacija za potrjevanje računov. Tu šele na koncu prenesemo podatke v standardne transakcije za upravljanje z matičnimi podatki dobavitelja. Vmesna sinhronizacija podatkov med standardnim dokumentom dobavitelja in našim dokumentom ni podprta. Zato nismo uporabili tehnik dodatkov ob shranjevanju standardne SAP transakcije.

9.1.3 Dokumenti v nabavi

Aplikacija Dokumenti v nabavi podpira potrjevanje različnih tipov nabavnih dokumentov. Z aplikacijo lahko potrjujemo naslednje vrste dokumentov: interno naročilo, naročilo, povpraševanje, pogodbo in terminski plan. Ko se kreira nabavni dokument v standardni SAP transakciji, se pojavi dialogno okno za izbiro scenarija in agentov potrjevanja. Dialogno okno ni del standardnega programa, ampak aplikacije za potrjevanje nabavnih dokumentov. V standardne transakcije smo ga dodali s pomočjo uporabe tehnik dodatkov. Ko so scenarij in agenti izbrani, se dokument lahko shrani. Pri shranjevanju dokumenta se na dokumentu postavi strategija lansiranja. Strategija lansiranja je SAPova funkcionalnost, ki označuje, ali je dokument že potrjen. Dokler dokument ni lansiran (potrjen), ga ne moremo nikjer uporabiti. Na primer, če naročilnica ni lansirana, je ne moremo uporabiti v transakciji za prevzem ali na vhodnih računih. Ko se dokument shrani in strategija lansiranja postavi, se zažene delovni tok aplikacije za potrjevanje nabavnih dokumentov. Ko je dokument potrjevanja odobren, se delovni tok zaključi, nabavni dokument pa se lansira.

Za nabavno naročilo in interno naročilo sta bila uporabljena klasična BAdIja ME_PROCESS_PO in ME_PROCESS_REQ. Čeprav sta oba BAdIja klasična, pa sta bila že migrirana na mesti dodatka, tako da se lahko uporabita tudi na jedrni način. S tema dvema BAdIjema dodamo različne funkcionalnosti v standardni aplikaciji za nabavno naročilo in interno naročilo. Z njima poskrbimo, da se prikaže stransko okno. Pri shranjevanju ali preverjanju podatkov se prikaže tudi dialogno okno, kjer se izbere scenarij potrjevanja in agente, ki bodo dokument potrdili. Z implementacijo BAdIjev preverimo tudi, če nabavni dokument ustreza pogojem za potrjevanje. Če ne ustreza pogojem, potem zanj ne kreiramo dokumenta potrjevanja in ne zaženemo delovnega toka.

V standardnih SAP programih za druge nabavne dokumente ni primernih BAdIjev, ki bi jih lahko uporabili za dodajanje novih funkcionalnosti, kot smo to naredili pri naročilih in internih naročilih. Zato smo za druge nabavne

dokumente morali uporabiti več različnih eksplicitnih dodatkov.

Pri tej aplikaciji so bili uporabljeni tudi jedrni BAdIji MEOUT_BAPI-_CUST, ME_BAPI_PO_CUST in ME_BAPI_PR_CUST. Z implementacijo teh BAdIjev smo poskrbeli za sinhronizacijo podatkov, če se dokumenti za potrjevanje spreminjajo preko naše spletne aplikacije. Prav tako smo omogočili kreiranje določenih nabavnih dokumentov tudi preko naše spletne aplikacije. Z implementacijo teh BAdIjev smo nato poskrbeli, da se kreirajo še ustrezni standardni dokumenti in naši dokumenti potrjevanja znotraj SAPa.

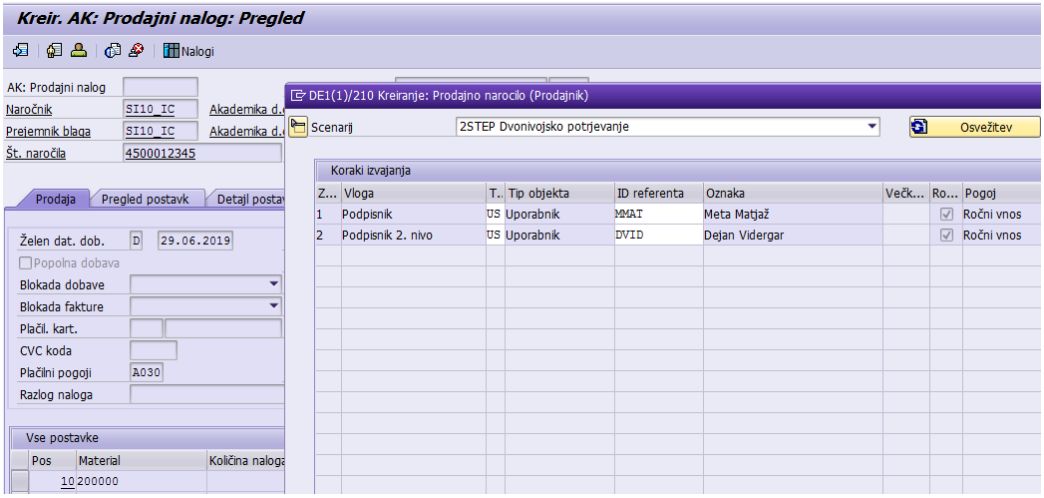
9.1.4 Prodajni dokumenti

Proces za potrjevanje prodajnih dokumentov se začne, ko se v standardni SAP transakciji shrani prodajni dokument. Trenutno aplikacija podpira prodajne naloge, kjer se proces začne, ko v SAP transakciji VA01 ustvarimo prodajni nalog. Če prodajni nalog ustreza pogojem za potrjevanje, se pojavi dialogno okno aplikacije za Prodajne dokumente, kjer lahko izberemo scenarij potrjevanja in določimo agente, ki bodo v procesu potrjevanja sodelovali (slika 9.4). Čeprav je za vsako stranko proces potrjevanja drugačen, ponavadi sledi naslednjemu vzorcu. Po kreiranju prodajnega naloga se na nalogu postavi blokada dobave in/ali blokada fakture. Ko je prodajni nalog potrjen se delovni tok za potrjevanje konča, s prodajnega naloga pa se odstrani blokada.

V aplikaciji za potrjevanje prodajnih nalogov smo uporabili eksplicitne dodatke, saj so bile eksplicitne točke definirane na ustreznih mestih v programu za upravljanje s prodajnimi nalogi SAPMV45A. Implementirali smo več točk eksplicitnega mesta ES_SAPMV45A.

V implementacijah teh točk dodatkov poskrbimo za naslednje stvari:

- prikaz stranskega okna znotraj standardne transakcije,
- preverjanje, ali prodajni nalog ustreza pogojem za potrjevanje,
- zagon delovnega toka za potrjevanje ob shranjevanju prodajnega naloga,



Slika 9.4: Pojavno okno za izbiro scenarija

- postavitev blokade fakture in/ali dobave ob shranjevanju prodajnega naloga,

- sinhronizacija podatkov med prodajnim nalogom in našim dokumentom.

9.1.5 Osnovna sredstva

V podjetju Akademika d.o.o. smo razvili tudi aplikacijo za upravljanje z osnovnimi sredstvi. Potrjevanje se začne z oddajo zahteve za kreacijo osnovnega sredstva, kjer avtor zahteve v dokument potrjevanja vnese podatke o osnovnem sredstvu in dobavitelju. Če dobavitelj v sistemu še ne obstaja, se ga lahko kreira preko aplikacije. V poteku potrjevanja se lahko kreira začasno številko osnovenga sredstva, postopek pa se ponavadi konča z aktivacijo osnovnega sredstva.

Aplikacija podpira kreacijo osnovnega sredstva in dobavitelja. Osnovno sredstvo in dobavitelja kreiramo s prenosom podatkov v standardno SAP transakcijo (AS01, XK01), kjer se objekt kreira s simulacijo akcij uporabnika v ozadju (angl. batch input). Aplikacija podpira tudi prikaz kreiranega

osnovnega sredstva v standardni SAP transakciji. Opisane funkcionalnosti so edine, ki uporabljajo standardne SAP transakcije oziramo programe. Pri razvoju teh funkcionalnosti nismo potrebovali nobene tehnike dodatkov. Zato se v tej aplikaciji uporablja samo novi BAdI, ki je bil razvit za konfiguracijo aplikacij in projektov podjetja Akademika d.o.o. (opisano v poglavju 9.1).

9.1.6 Službene poti in odsotnosti

Aplikacija za potrjevanje službenih poti je ponavadi razdeljena na tri faze. V prvi fazi potnik ali njegov nadrejeni odda zahtevo za potni nalog, kjer opiše kraj in čas potovanja. Pri oddaji zahteve navede tudi, če potrebuje letalsko karto, rezervacijo vozila, hotela ali kaj drugega. V drugi fazi gre zahteva za potovanje čez proces potrjevanja. V primeru potrditve se na koncu kreira SAPova standardna zahteva za potovanje na podlagi informacij, ki so bile zapisane pri oddaji zahteve. Ker ima lahko potnik na službeni poti določene stroške, se le ti zavedejo in potrdijo v tretji fazi - obračun. Tretja faza se konča, ko se obračun poknjiži v standardni SAP transakciji. S knjižbo obračuna se konča tudi proces potrjevanja službene poti. Ista aplikacija podpira tudi potrjevanje drugih odsotnosti, kot na primer dopustov.

Pri razvoju aplikacije ni bila uporabljena nobena tehnika dodatkov, razen jedrnih BAdIjev, ki so bili tudi razviti znotraj Akademike. Zahteva za potovanje in obračun se preko aplikacije kreirata v standardnemu SAPu s pomočjo standardnih SAP funkcijskih modulov, zato tu ne potrebujemo tehnik dodatkov. Če pride v standardni SAP transakciji do spremembe pri zahtevi za potovanje ali pri obračunu, se sproži dogodek na SAPovemu poslovnemu objektu. Aplikacija te dogodke ujame in ustrezno spremeni podatke na dokumentu potrjevanja. Zato tudi tu ni potrebno uporabiti tehnik dodatkov, saj za konsistentnost podatkov poskrbimo z lovljenjem dogodkov poslovnega objekta.

Poglavje 10

Sklepne ugotovitve

Na začetku diplomskega dela sem naredila pregled področja tako, da sem predstavila, kako se lahko dodaja kodo po meri v druge ERP sisteme. Po pregledu sistema, ki je v oblaku in drugega, ki ni, lahko zaključim, da se standardna koda v oblaku ne more spreminjati, drugače pa se lahko. Vendar to ne morem trditi zagotovo, saj bi morali za to pregledati večje število ERP sistemov. Kljub temu pa oba sistema omogočata dodajanje standardne kode ob različnih dogodkih, kot to omogoča tudi SAP ERP. To ni presenetljivo, saj ima vsako podjetje, ki uporablja ERP sisteme, svoje želje za prilagoditev standardnih aplikacij.

V diplomskem delu sem preučila vse obstoječe tehnike dodatkov SAP ERP sistema in predstavila njihove prednosti ter slabosti. Ugotovila sem, da SAP priporoča uporabo novejših tehnik, uporabo starejših pa odsvetuje. Kljub temu sem pri razvoju praktičnih primerov in pregledu aplikacij podjetja Akademika d.o.o. ugotovila, da velikokrat ne moremo izbirati med tehnikami dodatkov, vendar smo primorani uporabiti tisto, ki je na željenem delu standardnega SAP programa na voljo in nam ponuja uporabo podatkov programa, ki jih potrebujemo. Če nam SAP na določenem mestu v programu ni ponudil nobene tehnike dodatkov, potem lahko uporabimo implicitne dodatke, ki so na veliko mestih v kodi dodani privzeto. Pri aplikacijah Akademike d.o.o. se ne sme uporabiti tehnik dodatkov za enkratno uporabo. Ker

med te tehnike spadajo tudi implicitni dodatki, se pri razvoju aplikacij lahko zgodi, da nimamo na voljo nobene primerne tehnike.

Pri pregledu aplikacij Akademika d.o.o. sem ugotovila, da so v aplikacijah najbolj uporabljene tehnike dodatkov eksplicitni dodatki in BAdIji. Pri aplikaciji za potrjevanje računov, so bile uporabljene še BTE funkcije, ostale tehnike pa niso bile uporabljene, saj niso primerne za večkratno uporabo ali pa uporabljajo modifikacijo standardne SAP kode (uporabniški izhod).

Naredila sem tudi primerjavo klasičnih BAdIjev in jedrnih BAdIjev in potrdila trditve, ki sem jih zasledila v teoriji. Dokazala sem, da je uporaba jedrnih BAdIjev hitrejša. Pri mojih poskusih se je izkazalo, da so jedrni BAdIji približno 10-krat hitrejši od klasičnih.

Literatura

[1] Enhancement framework. Dosegljivo: https://help.sap.com/doc/saphelp_nw70/7.0.31/en-US/91/f1e540f8648431e10000000a1550b0/frameset.htm. [Dostopano: 4. 8. 2019].

[2] Suitescript. Dosegljivo: https://www.netsuite.com/portal/developers/resources/suitescript.shtml. [Dostopano: 10. 8. 2019].

[3] Abap - keyword documentation. Dosegljivo: https://help.sap.com/doc/abapdocu_751_index_htm/7.51/en-US/abenabap.htm, 2017. [Dostopano: 30. 7. 2019].

[4] Abap - wikipedia. Dosegljivo: https://en.wikipedia.org/wiki/ABAP, 2019. [Dostopano: 15. 8. 2019].

[5] Plug-ins guide. Dosegljivo: https://docs.oracle.com/cloud/latest/netsuitecs_gs/NSPLG/NSPLG.pdf, 2019. [Dostopano: 10. 8. 2019].

[6] Puneet Asthana and David Haslam. Abap 7.5 certification guide-the sap-endorsed certification series (sap press). 2018.

[7] Kiran Bandari. *Complete ABAP: The Comprehensive Guide to ABAP 7.5 (SAP PRESS)*. SAP PRESS, 2016.

[8] Stefano Demiliani and Duilio Tacconi. *Dynamics 365 Business Central Development Quick Start Guide: Modern Development Techniques for Dynamics 365 Business Central*. Packt Publishing Ltd, 2018.

[9] Courtney Gamangasso. The ultimate list of netsuite pros and cons. Dosegljivo: `https://www.techfino.com/blog/netsuite-pros-and-cons`, 2018. [Dostopano: 10. 8. 2019].

[10] Bhargav Mylavarapu. How to find a bte. Dosegljivo: `https://blogs.sap.com/2015/07/16/how-to-find-a-bte-business-transaction-events/`, 2015. [Dostopano: 3. 8. 2019].

[11] Neil Robertson. Understanding sap versions. Dosegljivo: `https://www.slideshare.net/peteaksaya/understanding-sap-versions`, 2019. [Dostopano: 15. 8. 2019].