UNIVERZITET U BEOGRADU
ELEKTROTEHNIČKI FAKULTET

Mlađan A. Jovanović

RAZVOJ KONTEKSTNO-OSETLJIVIH KORISNIČKIH INTERFEJSA

doktorska disertacija

Beograd, 2012

UNIVERSITY OF BELGRADE
FACULTY OF ELECTRICAL ENGINEERING

Mlađan A. Jovanović

DEVELOPMENT OF CONTEXT-SENSITIVE USER INTERFACES

Doctoral Dissertation

Belgrade, 2012

Mentor:	redovni profesor dr Zoran Jovanović Univerzitet u Beogradu, Elektrotehnički fakultet
Članovi komisije:	docent dr Igor Tartalja Univerzitet u Beogradu, Elektrotehnički fakultet
	redovni profesor dr Dušan Starčević Univerzitet u Beogradu, Fakultet Organizacionih Nauka

Datum odbrane: _____

Zahvalnice

Zahvaljujem se mentoru, prof. dr Zoranu Jovanoviću, na pomoći i razumevanju koje mi je pružio tokom izrade ovog rada. Zahvaljujem se komentoru, prof. dr Dušanu Starčeviću, na pomoći, savetima i podršci koju mi je pružao ne samo tokom izrade ovog rada, nego i u toku čitave dosadašnje saradnje. Takođe bih želeo da se zahvalim svojim kolegama i prijateljima koji su imali razumevanja za moj rad. I na kraju, želeo bih da se zahvalim svojoj majci čija mi je neizmerna ljubav bila najveći oslonac i podrška u radu.

Beograd, septembar 2012.

RAZVOJ KONTEKSTNO-OSETLJIVIH KORISNIČKIH INTERFEJSA

Rezime

Dobro dizajniran, intuitivan i privlačan za korišćenje korisnički interfejs predstavlja ključni faktor uspeha računarskih proizvoda i sistema. Radi unapređenja razvoja i upotrebljivosti korisničkih interfejsa potrebno je uzeti u obzir karakteristike korisnika. Ovo zahteva interdisciplinaran pristup i korišćenje znanja iz različitih oblasti kao što su računarske, saznajne i biološke nauke. Pored toga, potrebno je uzeti u obzir karakteristike medija i fizičkog okruženja u kojem se odvija interakcija čoveka i računara. Razvoj korisničkog interfejsa treba da uvaži i karakteristike hardverskih uređaja koji se koriste u komunikaciji sa korisnikom, dostupne softverske resurse, kao i karakteristike programskih sistema koji treba da koriste korisnički interfejs. U skladu sa tim, uvodi se pojam kontekstno-osetljivog interfejsa koji se definiše kao korisnički interfejs koji je prilagodljiv kontekstu interakcije sa konkretnim korisnikom. Kontekst interakcije čine tri klase entiteta: korisnik računarskog sistema (čovek); hardverska i softverska platforma pomoću kojih korisnici interaguju sa sistemom i fizičko okruženje u kojem se odigrava interakcija sa sistemom.

Posmatrajući evoluciju razvoja softvera uočavamo povećanje nivoa apstrakcije na kojem se softver opisuje. Dostignuti nivo razvoja omogućava platformski nezavisnu specifikaciju softvera koja se postepeno ili automatizovano prevodi u izvršne aplikacije za različite softverske i hardverske platforme. Arhitektura upravljana modelima, koja se koristi za razvoj složenih programskih rešenja, hijerarhijski organizuje koncepte i modele u više nivoa apstrakcije. Ovo je posebno bitno imajući u vidu da je razvoj kontekstno-osetljivih korisničkih interfejsa složen proces koji uključuje modelovanje velikog broja elemenata na različitim nivoima apstrakcije.

U ovoj tezi smo istraživali problem unapređenja razvoja kontekstno-osetljivih korisničkih interfejsa. Predloženo je rešenje koje omogućava automatizaciju razvoja korisničkog interfejsa prilagođenog kontekstu interakcije čoveka i računara. Rešenje se ogleda u proširenju jezika za modelovanje, standardnog procesa razvoja softverskih sistema (*Unified* proces) i razvojnih alata elementima specifičnim za interakciju čoveka i računara. U skladu sa prethodnim, razvijen je model kontekstno-osetljive interakcije čoveka i računara i predloženi su modeli korisničkih interfejsa na različitim nivoima apstrakcije. Zbog standardizacije, široke prihvaćenosti, i dostupnosti razvojnih alata, odlučili smo se za proširenje UML (Unified Modeling Language) jezika za modelovanje i ATL (Atlas Transformation Language) jezika za transformacije modela. Primena predloženog pristupa je demonstrirana na primerima dve studije slučaja iz različitih domena.

Ključne reči: interakcija čoveka i računara, konteksno-osetljivi korisnički interfejs, inženjerstvo vođeno modelima, arhitektura vođena modelima, model čoveka, model uređaja, model okruženja, višenačinska interakcija, model korisničkog interfejsa, transformacije modela.

Naučna oblast : sofversko inženjerstvo.

Uža naučna oblast : korisnički interfejsi.

UDK : _____

DEVELOPMENT OF CONTEXT-SENSITIVE USER INTERFACES

Abstract

Well-designed, intuitive and catchy-to-use user interface represents key issue of success of computer products and systems. In order to improve development and usability of user interfaces it is needed to take into account user's charasteristics. This entails interdisciplinary approach and use of knowledge from different fields such as computing, cognitive and biological sciences. In addition, it is needed to consider features of the physical environment and the medium in which interaction between human and computer takes place. Development of user interface must include characteristics of hardware devices employed in interaction with the user, availabale software resources, as well as characteristics of software systems using the interface. According to stated, concept of context-sensitive user interface is introduced, defined as a user interface adaptable to context of interaction with concrete user. Context of interaction is decomposed into three classes of entitites: user of a computer system (human); hardware and software platforms by which users interact with the system and physical environment in which interaction with system happens.

Looking at the evolution of software development, we can notice that the abstraction level on which software is described is increasing all the time. The latest trend is to specify software using platform-independent models, which are then gradually and (semi-) automatically transformed into executable applications for different platforms and target devices. Model-driven architecture, used for development of complex software solutions, hierarchically organizes concepts and models into multiple abstraction levels. This is especially important regarding development of context-sensitive user interfaces which appears to be a complex process involved with modeling of a large number of elements on different abstraction levels.

In this thesis, we have been exploring problem concerned with the improvement of development of context-sensitive user interfaces. Solution enabling automation of development of user interface adaptable to context of interaction between human and computer is proposed. Solution includes extensions of modelling language, standard software development process (Unified process) and development tools with the elements specific for human-computer interaction. Based on previous, model of context-sensitive human-computer interaction has been developed and user interface models on different abstraction levels have been proposed. For reasons of standardization, wide acceptance and availability of development tools, we have decided to extend UML (Unified Modeling Language) modeling language and ATL (Atlas Transformation Language) language for model transformations. Application of the proposed approach is demonstrated with examples of two case studies from different domains.

Keywords: human-computer interaction, context-sensitive user interface, model-driven engineering, model-driven architecture, human model, device model, environment model, multimodal interaction, user interface model, model transformations.

Scientific field: software engineering.

Scientific subfield: user interfaces.

UDK : _____

Sadržaj

1. Uvod

Za korisnika računarskog sistema korisnički interfejs predstavlja čitav sistem. Korisnik delove sistema i njegovu funkcionalnost vidi isključivo preko odgovarajućih elemenata tog interfejsa, pa zbog toga uspeh nekog softverskog proizvoda zavisi od pogodnosti i prihvatanja korisničkog interfejsa od strane korisnika. Ako korisnik ne može lako i intuitivno koristiti programski sistem, vrlo je verovatno da sistem neće postati šire prihvaćen, i zbog toga je razvoj korisničkih interfejsa sve bitniji deo razvoja nekog programskog sistema. Pored toga, nove unapređene mogućnosti računara, kao i pojava novih računarskih uređaja i aplikacija, donele su i potrebu za unapređenjem interakcije između čoveka i računara.

Izrada kvalitetnih korisničkih interfejsa sa stanovišta funkcija i upotrebljivosti je složen zadatak. Projektant korisničkih interfejsa mora uzeti u obzir više faktora iz oblasti kao što su računarske nauke, medicina, psihologija ili sociologija. Mnoga od ovih znanja su predstavljena u različitim oblicima i na različitim mestima, često previše specijalizovana i u formatu koji ih čini teško upotrebljivim u praktičnom razvoju programskih sistema. Sa druge strane, aktuelne metodologije koje projektanti koriste za razvoj softverskih sistema nisu prilagođene potrebama detaljnijeg definisanja interakcije između čoveka i računara. Iako je do sada bilo više pokušaja da se, u obliku priručnika ili baza znanja, opišu faktori od interesa za razvoj korisničkih interfejsa, kao i da se ova znanja integrišu sa softverskim inženjerstvom, ovi rezultati su imali ograničen uticaj.

U ovom radu istražujemo problem unapređenja razvoja korisničkih interfejsa korišćenjem modela kontekstno-osetljive interakcije između čoveka i računara. Model kontekstno-osetljive interakcije je rezultat primene interdisciplinarnog pristupa i objedinjavanja znanja iz oblasti kao što su računarske, saznajne i biološke nauke. Prethodnih godina pojavili su se industrijski široko prihvaćeni standardi, kao i inicijative poput arhitekture upravljane modelima, a koji mogu da pruže dobar okvir za formalan opis i primenu znanja iz oblasti interakcije između čoveka i računara u razvoju softverskih sistema. Korišćenjem modela kontekstno-osetljive interakcije i pomenutih industrijskih standarda, moguće je na različite načine unaprediti razvoj korisničkih interfejsa. Prvo, model može da služi kao baza znanja o konceptima i relacijama između koncepata koji opisuju interakciju između čoveka i računara. Pored toga, korišćenjem modela konteksta, moguće je semantički proširiti jezik za modelovanje i razvojno okruženje uvođenjem primitiva za opisivanje relevantnih faktora interakcije između čoveka i računara na različitim nivoima apstrakcije. Uvedena proširenja je moguće integrisati u postojeća razvojna okruženja sa kojima je upoznat veći broj projektanata i programera.

Uvodna izlaganja su organizovana u pet celina. Najpre će biti reči o rešavanom problemu i motivaciji za naša istraživanja. Nakon toga, preciznije definišemo cilj rada, polazne hipoteze i korišćenje metode. Zatim se iznosi kraći prostorno vremenski prikaz evolucije razvoja korisničkih interfejsa. Potom se prezentuju neke od korišćenih konvencija i termina u tekstu. Na kraju uvoda opisana je organizacija čitavog rada.

1.1. *Opis problema i motivacija*

U dosadašnjim pristupima razvoju korisničkih interfejsa obično se nije vodilo mnogo računa o specifičnostima čovekovih sposobnosti, što ograničava mogući kapacitet komunikacionih kanala između čoveka i računara. Dakle, radi unapređenja razvoja korisničkih interfejsa poželjno je uzeti u obzir karakteristike korisnika, što zahteva interdisciplinaran pristup i korišćenje znanja iz različitih oblasti kao što su računarske, saznajne i biološke nauke. Da bi novi korisnički interfejsi približili komunikaciju između čoveka i računara svakodnevnoj međuljudskoj komunikaciji, potrebno je uzeti u obzir niz faktora kao što su čovekova senzorska fiziologija i anatomija, percepcija, saznajni mehanizmi i socijalna interakcija. Takođe, potrebno je uzeti u obzir karakteristike medija i fizičkog okruženja korisnika. S druge strane, razvoj korisničkog interfejsa treba da uvaži i karakteristike hardverskih uređaja koji se koriste u komunikaciji sa korisnikom, dostupne softverske resurse, kao i karakteristike programskih sistema koji treba da koriste korisnički interfejs.

Dostupni softverski alati koji mogu da se koriste za razvoj korisničkih interfejsa ne pružaju odgovarajuću podršku novim zahtevima za unapređenje interakcije između čoveka i računara. Postojeći specijalizovani alati za razvoj korisničkih interfejsa problem interakcije između čoveka i računara rešavaju parcijalno i korišćenjem nestandardnih metodologija, što značajno umanjuje njihovu praktičnu upotrebljivost, a usložnjava integraciju korisničkog interfejsa sa ostatkom softverskog sistema. S druge strane, aktuelne metodologije za razvoj softverskih sistema nisu prilagođene potrebama detaljnijeg definisanja interakcije između čoveka i računara.

Dobro dizajniran, intuitivan i privlačan za korišćenje korisnički interfejs predstavlja ključni faktor uspeha računarskih proizvoda i sistema. Ovaj faktor još više dolazi do izražaja u nastupajućim paradigmama interakcije čoveka i računara kao što su ambijentalna inteligencija (*eng. Ambient Intelligence*) i sveprisutno računarstvo (*eng. Ubiquitous Computing or Pervasive Computing*). Pored toga, u mnogim oblastima postaće nezamislivo zahtevati od korisnika navikavanje na veliki broj različitih sistema. Zbog toga se ovi sistemi moraju projektovati tako da budu upotrebljivi, intutivni za korišćenje, adaptibilni i osetljivi na kontekst interakcije. Dakle, postoji potreba za novim pristupima automatske konstrukcije korisničkih interfejsa koji su ergonomični, jednostavni za učenje i optimalno upotrebljivi.

1.2. *Cilj rada i polazne hipoteze*

U radu će biti izložen pristup razvoju korisničkih interfejsa upravljan modelima koji će omogućiti automatsku konstrukciju korisničkog interfejsa prilagođenog kontekstu interakcije. Cilj rada je proširivanje i unapređenje standardne metodologije razvoja informacionih sistema elementima specifičnim za interakciju čoveka i računara. Ovo unapređenje se ogleda u uvođenju koncepata interakcije čoveka i računara u ranoj fazi razvoja sistema, tj. posmatrano u kontekstu arhitekture vođene modelima, u definisanju metamodela kontekstno-osetljive interakcije. U skladu sa tim, uvodi se pojam kontekstno-osetljivog (*eng. context-sensitive*) korisničkog interfejsa koji se definiše kao

korisnički interfejs koji ispoljava svest o kontekstu i na osnovu toga reaguje na promene u istom. Kontekst interakcije čine tri klase entiteta [Crowley02]:

- Korisnik sistema koji je opisan senzorskim, percepcijskim, kognitivnim i mehaničkim mehanizmima.
- Hardverska i softverska platforma, tj. računarski uređaji i uređaji za interakciju pomoću kojih korisnici interaguju sa sistemom.
- Fizičko okruženje u kojem se odigrava interakcija sa sistemom.

Korisnik predstavlja stereotip korisnika interaktivnog sistema. Postoji veliki broj kriterijuma za opisivanje i klasifikaciju korisnika, od kojih je najširu prihvaćenost našao Procesorski model čoveka (*eng. Human Processor Model*) [Card83] koji opisuje čoveka-korisnika sistema pomoću skupa vrednosti koje bliže određuju percepcijske, kognitivne i akcione kapacitete čoveka. Konkretno, specifičnosti percepcijskih, kognitivnih i akcionih kapaciteta konkretnog korisnika odrediće optimalne načine komunikacije za interakciju sa sistemom.

Plarforma se modeluje pomoću resursa, koji dalje određuju način na koji se informacija izračunava, emituje, prikazuje i obrađuje od strane korisnika. Primeri resursa uključuju kapacitet memorije, mrežni protok, ulazne i izlazne uređaje za interakciju. Resursi određuju ulazne i izlazne načine komunikacije koji će biti angažovani kod korisnika i za svaki od njih količinu dostupnih informacija. Tako je, na primer, veličina ekrana odlučujući faktor kod dizajna web strane. U opštem slučaju, platforma nije ograničena na jedan personalni računar, već obuhvata sve računarske i resurse za interakciju dostupne u datom vremenu i okruženju radi izvršenja skupa povezanih zadataka.

Okruženje predstavlja skup objekata, osoba i događaja koji su periferni u odnosu na tekuću aktivnost, ali mogu imati uticaj na ponašanje korisnika i/ili sistema, u datom trenutku ili u budućnosti [Coutaz02]. U praksi, granice okruženja postavljaju analitičari domena čiji je uloga da izdvoje entitete relevantne u konkretnom slučaju. Kriterijumi za klasifikaciju se svode na posmatranje navika korisnika [Dey01a, Beyer98, Cockton95], kao i na razmatranje tehničkih ograničenja. Na primer, buka okruženja se mora uzimati u obzir u slučaju sonifikacije. Slično tome, uslovi osvetljenja predstavljaju faktor od uticaja u slučaju korišćenja sistema za vizuelnog sistema za praćenje.

Polazna hipoteza je da je pomoću predloženog modela kontekstno-osetljive interakcije moguće kreirati arhitekturu upravljanu modelima za rešavanje problema interakcije između čoveka i računara, i na taj način unaprediti razvoj korisničkih interfejsa. Posmatrajući evoluciju softvera uočavamo povećanje nivoa apstrakcije na kojem se softver opisuje. Dostignuti nivo razvoja omogućava platformski nezavisnu specifikaciju softvera koja se postepeno ili automatizovano prevodi u izvršne aplikacije za različite softverske i hardverske platforme. Arhitektura upravljana modelima, koja se koristi za razvoj složenih programskih rešenja, hijerarhijski organizuje koncepte i modele u više nivoa, što je posebno bitno imajući u vidu da je razvoj korisničkih interfejsa složen proces koji uključuje modelovanje velikog broja elemenata na različitim nivoima apstrakcije. Pošto je arhitektura upravljana modelima zasnovana na standardnim i široko prihvaćenim tehnologijama, moguće je ostvariti veću praktičnu upotrebljivost rešenja jer se mogu koristiti postojeći alati za projektovanje, a sa kojima je upoznat veći

broj analitičara, projektanata i programera. U modelski zasnovanom pristupu model predstavlja osnovni rezultat razvojnog procesa, za razliku od ranijih pristupa gde je to mesto imao programski sistem. Osnovna prednost modelski zasnovanog pristupa je mogućnost opisa sistema korišćenjem koncepata koji su manje ograničeni sa mogućnostima implementacione platforme, a bliži su domenu problema koji se rešava. Ovo čini model lakšim za definisanje, razumevanje i održavanje, a u nekim slučajevima čak i domenski ekspert može samostalno da kreira model. Takođe, na ovaj način model postaje manje osetljiv na odabranu implementacionu platformu, pa je zato i termin platformski-nezavisan model često tesno povezan sa modelski zasnovanim razvojem računarskih sistema.

Polazeći od osnovne hipoteze, u tezi smo razmatrali dva osnovna problema:

- razvoj modela kontekstno-osetljive interakcije,
- primenu modela kontekstno-osetljive interakcije u razvoju korisničkih interfejsa.

Model kontekstno-osetljive interakcije predstavlja jedinstven i formalan opis osnovnih koncepata i relacija između faktora od interesa za razvoj korisničkog interfejsa. Izrada modela kontekstno-osetljive interakcije između čoveka i računara je rezultat primene interdisciplinarnog pristupa i objedinjavanja znanja iz različitih naučnih naučnih oblasti. Model može da se posmatra i kao ontologija znanja potrebnih za razvoj novih korisničkih interfejsa, jer treba da sadrži pregled najznačajnijih rezultata i saznanja iz predmetnih oblasti. Na osnovu predloženog modela, biće opisani različiti aspekti razvoja korisničkih interfejsa. Korišćenjem modela, moguće je semantički proširiti razvojno okruženje uvođenjem primitiva specifičnih za modelovanje interakcije između čoveka i računara na različitim nivoima apstrakcije. Na taj način moguće je potpunije i preciznije opisati poželjna svojstva interakcije između čoveka i računara za dati programski sistem. Ovde će prvenstveno biti razmatrani mehanizmi integracije predloženih koncepata u standardne jezike i metodologije razvoja softverskih sistema, kao i prednosti koje takav pristup može doneti projektantima u različitim fazama razvoja. Korišćenje formalnih modela može da umanji rizik razvoja softvera pomažući nam da bolje razumemo složeni problem i njegova potencijalna rešenja pre nego što nego što pristupimo punoj implementaciji sistema. Ovo je posebno bitno za razvoj korisničkih interfejsa, gde je potrebno uzeti u obzir veliki broj faktora iz različitih oblasti. Formalan opis sistema predstavlja i osnovu za automatizaciju dela razvojnog procesa, pa se u radu istražuju i mogućnosti transformacija modela korisničkih interfejsa, kao i mogućnosti razvoja alata koji mogu da podrže ove transformacije.

U izradi teze korišćene su sledeće metode istraživanja:

- Prikupljanje i sređivanje podataka o dostupnim rešenjima na osnovu dostupne literature, pretraživanjem Interneta, razmenom informacija i neposrednim kontaktima sa relevantnim subjektima u svetu,
- Analiza postojećih i predlog sopstvenog rešenja sintezom elemenata postojećih rešenja,
- Formalna specifikacija rezultata istraživanja. Kao alat za istraživanje i opisivanje rezultata istraživanja biće korišćena objektno-orijentisana metodologija projektovanja i jezik UML, i
- Verifikacija polazne hipoteze izradom nekoliko odabranih primera.

1.3. *Osvrt na istorijat razvoja korisničkih interfejsa*

Problem koji rešavamo u ovom radu je predmet istraživanja u dužem vremenskom periodu. Da bismo bolje sagledali trenutno stanje i pravce razvoja, u ovom delu dajemo kratak pregled istorije korisničkih interfejsa. Ovde ćemo se prvenstveno zadržati na opisu početaka razvoja korisničkih interfejsa, kao i na osnovnim faktorima koji su upravljali ovim razvojem, dok je detaljniji pregled postojećih rešenja dat u poglavlju 2.

Termin "korisnički interfejs" nije bio relevantan u prvim računarskim sistemima jer su prvi korisnici računara bili su inženjeri, a korišćenje računara je zahtevalo potpuno poznavanje hardvera. Inženjeri su interakciju sa računarom obavljali neposrednim radom na hardveru, na primer, podešavanjem prekidača, a osnovni oblik komunikacije je bio razmena brojeva u binarnom, oktalnom ili heksadecimalnom brojevnom sistemu. Pri tome su inženjeri direktno radili sa registrima i memorijskim lokacijama, a težište u radu sa računarima je bilo poboljšanje performansi izvršavanja instrukcija.

Dakle, u početku je hardver bio centralni deo korisničkog interfejsa, a tipični korisnici su bili inženjeri i programeri. Unapređenje korisničkog interfejsa išlo je u tri pravca [Grudin90]. Prvo, razvijan je pouzdaniji hardver sa više mogućnosti, tako da su korisnici mogli da sa manje komandi i većom pouzdanošću koriste računare. Drugo, ergonomska prezentacija i manipulacija hardverskih komponenti je unapređivana, na primer, pogodnijem rasporedom komponenti i dodavanjem oznaka na prekidače. Treće, korisnici računara, prvenstveno programeri, su postepeno bili oslobađani potrebe da poznaju sve detalje funkcionisanja hardvera, na primer, uvođenjem koncepta virtuelne memorije. Čak i 70-tih godina dvadesetog veka, računari su imali sličan hardverski korisnički interfejs, dok su napredniji računari često još više smanjivali udaljenost između korisnika i hardvera [Gentner90]. Većina korisnika je radila sa računarima prema precizno definisanim uputstvima. Fizička ograničenja, kao što je veličina memorije, bila su izuzetno velika. Pored toga, računarska oprema i radno vreme računara bili su veoma skupi. Na primer, prvi ekrani sa katodnom cevi (*eng. cathode ray tube - CRT*) koštali su preko 10.000 dolara.

Tokom 60-tih i 70-tih godina dvadesetog veka, programeri su ostali glavni korisnici računara. Prvi editori teksta bili su linijski orijentisani editori namenjeni programerima. Upotreba računara u cilju proizvoljne obrade teksta bila je preskupa. U osnovi, unapređenje korisničkog interfejsa značilo je unapređenje efikasnosti programera. Kritična unapređenja, iako još nisu bila okarakterisana kao unapređenja korisničkog interfejsa, bili su viši programski jezici, "debageri" i operativni sistemi. Međutim, navedena unapređenja obično nisu uzimala u obzir ljudske faktore poput čitljivosti koda ili unapređenja pamćenja termina. U slučaju programskih jezika, na primer, osnovni fokus je bio stavljen na boljem strukturiranju koda i podizanju nivoa apstrakcije, a ne na unapređenju ergonomskih karakteristika izrade koda [Wexelblat78]. Tek kasnije su se pojavile studije o programiranju kao obliku rešavanja problema, o strategijama otkrivanja grešaka (*eng. debugging*), o efektivnoj programskoj dokumentaciji i o vizuelizaciji programa. Čak i početkom 80-tih godina dvadesetog veka, većina

istraživanja je i dalje bila fokusirana na programiranje kao osnovni oblik interakcije sa računarom i zasnovana na pretpostavci da je osnovni korisnik računara programer.

Od niza unapređenja, za programere su najznačajnije novine predstavljale mogućnosti rada sa više procesa (*eng. multitasking*), virtuelna memorija i interaktivni terminali. Interaktivni terminali su doneli najznačajnije promene, jer su ne samo povećali broj korisnika već su doveli do pojave nove klase korisnika sistema koji nisu bili programeri. Termin "korisnički interfejs" je uskoro postao često korišćen, a nova oblast istraživanja interakcije između čoveka i računara počela je da se formira. Iako je rad na unapređenju programerskih okruženja i dalje nastavljen, težište rada u oblasti interakcije između čoveka i računara je prešlo sa *pomoći programerima kao korisnicima, na pomoć programerima da razviju bolje interfejse za korisnike koji nisu programeri*.

Vizuelni displeji i interaktivne mogućnosti terminala otvorile su novu oblast razvoja i istraživanja. Uskoro se pojavio velik broj istraživanja o percepcijskim problemima interakcije između čoveka i računara poput čitljivosti ispisa ili istraživanja motorike u radu sa mehaničkim ulaznim uređajima. Postojeća istraživanja ergonomije i drugih ljudskih faktora, počela su da se primenjuju i proširuju na oblast interakcije između čoveka i računara. Kao tipičan primer možemo navesti Fittsov zakon koji opisuje zavisnost između vremena potrebnog za označavanje nekog objekta do njegove udaljenosti i veličine [Fitts54]. Ovaj zakon je definisan 1954. godine, ali je vrlo uspešno prenesen u oblast interakcije između čoveka i računara [MacKenzie92, Moguffin05].

Fleksibilnost i laka proširljivost računarskih sistema uskoro je dovela do pojave interaktivnih elemenata korisničkih interfejsa poput funkcijskih tastera, komandnih jezika i menija. Ovi novi elementi doveli su do proširenja istraživanja na teme iz saznajne psihologije poput učenja motoričkih radnji, formiranja koncepata, semantičke memorije i akcija.

Rad na percepcijskim i osnovnim saznajnim procesima zajedničkim za većinu korisnika ostao je dominantan fokus istraživanja u oblasti interakcije između čoveka i računara. Neki od osnovnih istraživačkih tema o interakciji između čoveka i računara, koje su aktuelne i danas, definisao je 1986. godine Shneiderman na CHI'86 konferenciji [Shneiderman86]. Oblast interakcije između čoveka i računara je okupila i velik broj istraživača psihologije, medicine, saznajnih nauka, ali i određen broj umetnika i dizajnera.

Nakod formiranja oblasti interakcije između čoveka i računara, broj istraživača i radova je značajno porastao. Pored istraživanja percepcijskih i saznajnih mehanizama pojedinačnih korisnika, počela su da se primenjuju i istraživanja iz sociologije u cilju podrške grupnim aktivnostima korisnika. Predmet interakcije između čoveka i računara postaje i sastavni deo nastavnog procesa na velikom broju univerziteta.

Bez obzira na činjenicu da mnoga istraživanja iz oblasti interakcije između čoveka i računara nisu odmah našla praktičnu primenu, neka od njih su imala fundamentalan uticaj na razvoj današnjih računarskih sistema. Na primer, grafički interfejs koji danas postoji u Windows i Linux operativnim sistemima, zasnovan je na rešenjima koja su početkom 80-tih godina prošlog stoleća koristili Apple

računari. S druge strane, Apple računari su primenili istraživanja iz Xerox PARC laboratorija, zasnovana na istraživanjima iz 60-tih godina dvadesetog stoleća realizovana na Standfordu pri Institutu za tehnologiju u Masačusetsu [Myers98].

Iako je postignut veliki komercijalni uspeh grafičkih korisničkih interfejsa *(eng. graphical user interfaces - GUI*), paradoksalno je da interakcija između čoveka i računara nije značajno unapređena u periodu od dve decenije. Mada je grafički korisnički interfejs danas *de facto* standard, on je daleko od toga da bude dovoljno efikasan i intuitivan. Komunikacija sa računarom preko ovih interfejsa ne koristi dovoljno čovekove percepcijske i izražajne mogućnosti i značajno je drugačija od uobičajne međuljudske komunikacije. Zbog toga je izvršen veći broj istraživanja i eksperimenata u cilju unapređenja interakcije između čoveka i računara eksploatišući u većoj ili manjoj meri karakteristike čovekovog percepcijskog aparata. Nove paradigme interakcije između čoveka i računara polaze od pretpostavke da komunikacija između njih treba da bude zasnovana na istim principima koje obični ljudi svakodnevno koriste pri međusobnoj komunikaciji ili pri opažanju sveta oko sebe. To je dovelo do pojave više novih klasa korisničkih interfejsa kao što su percepcijski korisnički interfejsi, korisnički interfejsi zasnovani na pažnji, elektro-fiziološki korisnički interfejsi i dodirni korisnički interfejsi [Turk00, Vertegaal03, Allanson02, Ishii08].

Sve češći zahtevi za značajnim izmenama paradigme interakcije, posledica su i pojave novih računarskih uređaja uređaji poput prenosivih računara i mobilnih telefona koji ne mogu da iskoriste rešenja sa stonih računara. Kao što je predviđeno od strane Mark Weisera u Xerox PARC laboratoriji, doba svuda prisutnog računarstva (*eng. ubiquitous computing*) je već stiglo [Weiser94]. Weiser je predvideo da će okruženja za život i rad biti ispunjena računarskih i komunikacionim servisima koji su orijentisani ka čoveku i da će čovek u svakom trenutku i na svakom mestu imati pristup ovim servisima. Danas su ovi servisi dostupni čoveku zahvaljujući uređajima kao što su IPhone mobilni telefoni ili IPad uređaji, koji imaju integrisane tekstualno-govorne mehanizme, ekrane osetljive na dodir i čitav skup servisa poput *location-based* servisa (*Google Maps*) ili servisa socijalnih mreža (*Facebook, Twitter, LinkedIn*). S druge strane, prikazne tehnologije velikih dimenzija, poput displeja veličine zida (*eng. wall-size displays*) ili displeja proširene stvarnosti (*eng. augmented-reality displays*) su već dostupni u obliku projekcionih uređaja ili u obliku velikih plazma panel displeja. Cena hardvera će prema procenama i dalje nastaviti da opada, a pojavljivaće se i nove računarske mogućnosti i primene. Interfejsi na ovim veoma malim ili veoma velikim displejima tipično ne mogu koriste standardni *desktop* model. Pored toga, Windows elementi korisničkog interfejsa poput padajućih menija nisu pogodni ni za velike displeje veličine zida, jer meniji mogu biti previsoki za niske korisnike. Nadalje, ljudi rade sa više uređaja istovremeno, tako da će uređaji morati da komuniciranju i koordiniraju svoje aktivnosti. Kao posledica ovih promena može se očekivati značajno povećanje raznolikosti računarskih uređaja i konteksta u kome će se oni koristiti, što će zahtevati i razvoj odgovarajućih korisničkih interfejsa. Raznolikost uređaja za interakciju postavlja nove izazove pred HCI softversku zajednicu [Eisenstain01]. Tu pre svega spadaju:

- Kreiranje i održavanje verzija jedne aplikacije za različite tipove uređaja.
- Proveravanje konzistentnosti između verzija radi održavanja istovetnog načina interakcije sa aplikacijom na različitim uređajima.

- Ugrađivanje sposobnosti dinamičkog prilagođavanja verzija promenama u okruženju kao što su konektivnost mreže, karakteristike korisnika, lokacija korisnika, ambijentalni zvuk ili uslovi osvetljenja.

Kako bi navedeni zahtevi bili ispunjeni, pojam konteksta se definiše kao triplet <*korisnik, okruženje, platforma*>. Pored toga, uvedeni su pojmovi višeciljnosti (*eng. multi-context ili multi-target*) i plastičnosti (*eng. plasticity*) [Collignon08]. Oba pojma su usko povezana sa adaptacijom konteksta upotrebe korisničkog interfejsa koja pokriva raznovrsnost uređaja za interakciju bez velikog povećanja troškova razvoja i održavanja. Pojmovi višeciljnosti i plastičnosti se odnose na adaptaciju raznovrsnih konteksta upotrebe. Dok se pojam višeciljnosti odnosi na tehničke aspekte adaptacije, plastičnost definiše način za vrednovanje upotrebljivosti sistema od strane korisnika u toku adaptacije [Coutaz10]. Višekontekstni korisnički interfejs podržava veći broj konteksta upotrebe, ali ovde ne postoji zahtev u pogledu očuvanja upotrebljivosti adaptiranih interfejsa. Sa druge strane, po analogiji sa osobinom materijala koji se šire i skupljaju pod određenim uslovima bez oštećenja, plastični korisnički interfejs je višekontekstni korisnički interfejs koji zadržava svojstvo upotrebljivosti u toku adaptacije [Vanderdonckt08].

1.4. *Terminologija i konvencije korišćene u tekstu*

U ovom delu uvodnih izlaganja ukratko ćemo izneti neke od konvencija koje smo koristili pri pisanju teksta, kao i definicije nekih od ključnih termina.

Za označavanje referenci koristili smo oznaku sastavljenu od prezimena prvog autora i dvocifrenog zapisa godine izdanja. Ukoliko u istoj godini postoji više radova sa istim prvim autorom, oznaku smo proširili sa dodavanjem malog slova (‘a’, ‘b’, ‘c’, ...). Ukoliko rad nema eksplicitno navedene autore, kao što je slučaj sa tekstovima standarda, onda referenca sadrži skraćeni naziv dokumenta i godinu izdanja. Mnoge stručne reči engleskog porekla, a koje su se pojavile u poslednjih nekoliko godina u računarskoj literaturi, kod nas još uvek nemaju usaglašen prevod pa se prevode različito. Zbog toga smo u tekstu, pored usvojenog prevoda, kod prvog pojavljivanja ovakvih termina, u zagradi ostavili i originalnu englesku reč. Izuzetak su nazivi jezika UML i Unified procesa, koje nismo prevodili.

Imajući u vidu da terminologija koja se koristi u oblasti interakcije između čoveka i računara još uvek nije ujednačena, i da značenje termina često zavisi od konteksta u kojem se on upotrebljava, definisaćemo i pojasniti neke od ključnih termina korišćene u ovom radu. Tehnički termini često se koriste bez preciznog razumevanja onoga šta oni označavaju [Meera96]. Neki od ovih termina će biti detaljnije objašnjeni u kasnijim poglavljima.

- *Medij* u računarskim naukama predstavlja nosilac ili prenosilac podataka. Termin medij ima više značenja, pa bi preciznije bilo razlikovati: prezentacioni medij, koji označava tip fizičkog sredstva pri komunikaciji čoveka i računara, smeštajni medij kojim se označava tip fizičkog sredstva za skladištenje podataka, prenosni medij, kojim se označava tip fizičkog sredstva za prenos podataka, reprezentacioni medij, kojim se označava načina predstavljanja ili reprezentacije medijskih informacija u apstraktnoj formi posredstvom skupa medijskih podataka i veza između tih

podataka i percetivni medij, kojim se označava pojavni oblik nosioca informacija posmatrano sa gledišta korisnika.

- *Interfejs* u opštem smislu je mesto dodira dva ili više entiteta, i on obično podrazumeva blizinu granica tih entiteta.
- *Interfejs između čoveka i računara* (*eng. human-computer interface*) omogućuje ljudima i računarima da međusobno komuniciraju. Ovi sistemi se sastoje od fizičkih ulazno-izlaznih uređaja neophodnih radi fizičkog prenošenja podataka između čoveka i računara kao i od odgovarajućeg jezika komunikacije. Pod jezikom komunikacije podrazumevamo skup simbola, signala i konvencija koje čovek i računar koriste pri komunikaciji.
- *Način komunikacije* (*eng. modality ili mode*) ima višestruko značenje. U računarskim naukama se termin način često koristi kao sinonim za stanje. Na primer, editor može biti u cut-and-paste stanju (*eng. cut-and-paste mode*) za razliku od stanja unosa (*eng. input mode*). Psiholozi, s druge strane, termin način komunikacije koriste kada govore o angažovanju čovekovih senzorskih sposobnosti (*eng. sensory modalities*) poput vida, sluha, dodira, mirisa ili ukusa. U radu ćemo se osloniti na kasnije značenje kada govorimo o interakciji između računara i čoveka, s tim što ćemo ga proširiti u smislu angažovanja i ostalih sposobnosti ćoveka. Takođe, možemo da govorimo i o načinima interakcije kao o stilovima interakcije. Na primer, meniji kao kontrole korisničkog interfejsa angažuju određene stilove interakcije.
- *Višenačinski interfejs* realizuje interakciju između čoveka i računara uključivanjem više čovekovih izražajnih i senzornih načina komunikacije. Ovi načini komunikacije čine sastavne elemente jezika interfejsa.
- *Kontekst upotrebe* (*eng. context of use*) predstavlja triplet <*korisnik, platforma, okruženje*>. Kontekst se može posmatrati kao domensko znanje, odnosno skup činjenica (entiteta i njihovih međusobnih relacija) i pravila koja opisuju okruženje i korisnika u određenoj situaciji. Višekontekstni (*eng. multi-context ili muti-target*) korisnički interfejs podržava veći broj tipova korisnika, platformi i okruženja u smislu obezbeđivanja optimalne komunikacije korisnika i sistema. Višekorisnički (*eng. multi-user*), višeplatformski (*eng. multi-platform*) i interfejsi koji podržavaju veći broj okruženja (*eng. multi-environment*) predstavljaju specifične klase višekontekstnih interfejsa. Interfejs koji podržava veći broj konteksta upotrebe se često nazivaju i interfejsima koji imaju svest o kontekstu (*eng. context-aware user interfaces*).

1.5. *Struktura rada*

Rad je organizovan izlaganjem rezultata istraživanja u devet poglavlja. Poglavlje 2 sadrži detaljan pregled relevantnih oblasti vezanih za predmet istraživanja. Najpre je izvršena klasifikacija korisničkih interfejsa sa naglaskom kontekstno-osetljive korisničke interfejse. Nakon toga su opisane postojeće softverske arhitekture za razvoj korisničkih interfejsa. Zatim se daje pregled postojećih jezika za opis i softverskih alata za razvoj korisničkih interfejsa. Softverski alati za razvoj korisničkih interfejsa su razmatrani i sa aspekta transformacija modela. Rezime poglavlja sadrži diskusiju o problemima i ograničenjima opisanih rešenja sa aspekta kontekstno-osetljivih korisničkih interfejsa. Pored toga, izvršena je i analiza sa kritičkim osvrtom na pogodnosti korišćenja postojećih rezultata u razvoju

kontekstno-osetljivih korisničkih interfejsa. Analiza je izvršena sa aspekta osnovnih komponenata kontekstno-osetljive interakcije (okruženje, računarska platforma i čovek) i aspekta razvoja korisničkih interfejsa.

U poglavlju 3 izlažemo osnovnu ideju razvoja kontekstno osetljivih korisničkih interfejsa pomoću modela kontekstno-osetljive interakcije, kao i primenjena načela i smernice kojima smo se rukovodili u toku naših istraživanja. Najpre opisujemo osnovnu ideju predloženog pristupa razvoju korisničkih interfejsa pomoću modela kontekstno-osetljive interakcije između čoveka i računara. Ovde opisujemo predloženi model kontekstno-osetljive interakcije, kao i mogućnosti njegove primene u procesu izrade korisničkih interfejsa. Potom opisujemo arhitekturu i organizaciju rešenja. Na kraju dajemo sažetak i diskutujemo o očekivanoj dobiti od predloženog pristupa.
U poglavlju 4 detaljnije opisujemo izvore znanja, način izrade i strukturu modela kontekstno-osetljive interakcije. Prvo opisujemo izvore znanja, gde smo naveli oblasti čija smo znanja koristili za definisanje modela kontekstno-osetljive interakcije. Zatim govorimo o razvoju modela kontekstno-osetljive interakcije gde smo opisali pristupe kreiranju modela, alate koje smo koristili pri razvoju i korišćenu notaciju za opis modela. Nakon toga opisujemo njegovu strukturu, a zatim pojedine delove modela kao što su faktori čoveka, okruženje, faktori računarskog uređaja i kontekstno-osetljiva interakcija čoveka i računara.

U poglavlju 5 opisujemo proširenja jezika za modelovanje programskih sistema na osnovu generičkog modela kontekstno-osetljive interakcije između čoveka i računara. U radu smo se opredelili za razmatranje mogućnosti proširenja objektno-orijentisanog pristupa modelovanju sistema. Zbog standardizacije, široke prihvaćenosti, i dostupnosti razvojnih alata, u ovom radu smo prvenstveno razmatrali mogućnosti proširenja jezika UML (*eng*. *Unified Modeling Language*). Najpre opisujemo mehanizam proširenja jezika za modelovanje korišćenjem modela kontekstno-osetljive interakcije između čoveka i računara. Potom opisujemo proširenja jezika za modelovanje zasnovana na predloženom modelu, i proširenja modela specifičnih za razvoj korisničkih interfejsa.

U poglavlju 6 opisujemo neke od mogućih pristupa proširenju standardnog procesa razvoja softvera sa konceptima vezanim za kontekstno-osetljive korisničke interfejse. Osnovna ideja našeg pristupa jeste integracija odgovarajućih UML profila sa modelima koji se realizuju u pojedinim fazama razvoja. Najpre dajemo sažet pregled Unified procesa, kao i neke nedostatke ovog procesa sa stanovišta razvoja korisničkih interfejsa. Potom opisujemo osnovnu ideju proširenja razvojnog procesa. Nakon toga je demonstrirana upotreba predloženih UML proširenja u pojedinim fazama procesa razvoja.

U poglavlju 7 opisujemo mogućnosti transformacija modela kontekstno-osetljivih korisničkih interfejsa kroz proširenja postojećih razvojnih alata. Predmet transformacija su modeli kreirani korišćenjem proširenja za modelovanje kontekstno-osetljivih korisničkih interfejsa. Polazeći od ovih modela, istraživali smo mogućnosti proširenja alata, automatizacije delova razvojnog procesa i transformacija modela korisničkih interfejsa.

U poglavlju 8 demonstrirana je upotreba predloženih rešenja na primeru dve studije slučaja.
Na kraju je dat zaključak u kojem su navedeni najznačajniji doprinosi teze, kao i smernice za dalji rad.

2. Pregled relevatnih naučnih oblasti

U ovom poglavlju je dat pregled relevantnih naučnih oblasti. Ovde definišemo osnovnu terminologiju i klasifikujemo postojeća rešenja. Najpre dajemo sažet opis različitih generacija i vrsta korisničkih interfejsa. Nakon toga, dajemo širi opis kontekstno-osetljivih korisničkih interfejsa. Potom razmatramo korisničke interfejse sa stanovišta razvoja softvera. Drugim rečima, daje se pregled postojećih softverskih arhitektura, alata i jezika za razvoj korisničkih interfejsa. Poseban deo poglavlja je posvećen pregledu postojećih alata za razvoj korisničkih interfejsa vođen modelima. Osnovna svrha poglavlja jeste upoznavanje sa osnovnim konceptima postojećih rešenja, ali i sagledavanje njihovih problema i ograničenja. Poglavlje služi i kao referentna baza termina koji će biti korišćeni u kasnijim poglavljima. Na kraju poglavlja će biti izvršena analiza sa kritičkim osvrtom na pogodnosti korišćenja postojećih rezultata u razvoju kontekstno-osetljivih korisničkih interfejsa.

2.1. *Vrste korisničkih interfejsa*

Prema korišćenoj tehnološkoj bazi i paradigmi interakcije koju su koristili, korisničke interfejse možemo podeliti u sledeće grupe:

- Hardverski korisnički interfejsi. Vezani za početke komunikacije čoveka i računara koje karakteriše ručno povezivanje hardverskih komponenti, upotreba dugmadi, prekidača, signalnih lampica i bušenih kartica.

- Terminalski korisnički interfejsi. Podrazumevaju korišćenje teleprinterskih ili video terminala gde je najviše korišćeni oblik interakcije bio putem tekstualnog interfejsa komandne linije.

- Grafički korisnički interfejsi. Zasnovani na WIMP (*eng. windows, icons, menus, pointer*) paradigmi interakcije [Sutherland63],[English67],[Engelbart68],[Scheifler86].

- Percepcijski korisnički interfejsi. Integrišu perceptivnu korisničke interfejse (*eng. perceptive user interfaces*), višenačinske korisničke interfejse (*eng. multimodal user interfaces*) i multimedijalne korisničke interfejse (*eng. multimedia user interfaces*). Perceptivni korisnički interfejsi imaju sposobnost opažanja korisnika i njegovih aktivnosti [Pentland00]. Multimedijalni korisnički interfejsi prvenstveno prenose informacije od računara ka korisniku [Gonzales00]. Višenačinska komunikacija je obostrana [Oviatt00].

- Korisnički interfejsi zasnovani na pažnji (*eng. attentive user interfaces - AUI*). Kao osnovni cilj imaju prepoznavanje prostora korisnikovih namera u cilju optimizacije resursa za obradu informacija korisnika i računara. Ovo postižu merenjem i modelovanjem usmerenosti korisnikove pažnje ka zadatku, uređajima i drugim ljudima [Vertegaal03].

- Elektro-fiziološke korisničke interfejse. Kombinuju tehnologije za opažanje fizioloških signala iz medicinskih aplikacija sa raznim tehnikama izrade interaktivnih računarskih sistema [Allanson02]. Predstavljaju platformu za mnoge aplikacije kao što su interfejsi između mozga i

računara (*eng. brain-computer interfaces - BCI*) [Millán03], proteze, i različite oblike interakcije bez korišćenja ruku.

- Osetni korisnički interfejsi (*eng. tangible user interfaces - TUI*). Vezani za disciplinu interakcije čoveka i računara zasnovanu na korišćenju metafora koje oblikuju digitalne infornacije u fizičke forme iz realnog sveta (*eng. Tangible Interaction*) [Ishii08]. Cilj je postizanje integracije fizičkog i digitalnog sveta kako bi se unapredila sama komunikacija čoveka i računara. Ovo se postiže angažovanjem znanja i veština čoveka u korišćenju svakodnevnih objekata iz realnog sveta poput prostornih, motoričkih i socijalnih veština [Shaer09]. Osnovni problemi vezani za razvoj ove vrste korisničkih interfejsa su konceptualne, metodološke i tehničke prirode [Shaer09].

- Organski korisnički interfejsi (*eng. Organic User Interfaces - OUI*) predstavljaju vrstu korisničkih interfejsa koji koriste neplanarne objekte kao primarna sredstva ulaza i izlaza [Holman08]. Dakle, ova vrsta interfejsa ne predviđa čisto planarnu strukturu prikaza, kao kod LCD displeja, već displeje koji primaju oblik objekata koji se sreću u realnom svetu. Postojeći OUI su uglavnom zasnovani na korišćenju OLED (*eng. Organic light-emitting diode*) materijala [Piper02], [Poupyrev07] i displejima zasnovanim na papirima (*eng. paper-based displays*) [Holman05].

Kontekstno-osetljive korisničke interfejse ne možemo izdvojiti kao posebnu vrstu sa stanovišta kriterijuma prethodno navedene klasifikacije. Ova vrsta korisničkog interfejsa na određeni načim ispoljava svest o kontekstu interakcije čoveka i računara, te se na taj način pod nju može podvesti bilo koji od navedenih tipova. Zbog toga je opisu kontekstno-osetljivih korisničkih interfejsa posvećena sekcija koja sledi.

2.2. *Kontekstno-osetljivi korisnički interfejsi*

Radi boljeg i sveobuhvatnijeg razumevanja, u okviru ovog odeljka najpre će biti opisana oblast kontekstno-osetljivog računarstva. Sistematizacija oblasti koje obuhvata kontekstno-osetljivo računarstvo će poslužiti kao uvod u opisivanje kontekstno-osetljivih korisničkih interfejsa. Kontekstno-osetljivi korisnički interfejsi će biti opisani na primerima koji su našli širu primenu u praksi.

2.2.1. Kontekstno-osetljivo računarstvo

Svakodnevna međuljudska komunikacija sama po sebi omogućava uspešnu razmenu ideja i informacija među sagovornicima. Razlozi za ovo, između ostalog, jesu bogatstvo jezika, zajedničko razumevanje principa po kojima spoljašnji svet funkcioniše i implicitno razumevanje svakodnevih situacija [Dey01b]. Kada ljudi komuniciraju međusobno, oni su u mogućnosti da implicitno koriste informaciju o situaciji, drugim rečima kontekst, kako bi povećali sam kapacitet komunikacije. Ovo se, međutim, ne može preslikati na komunikaciju čoveka i računara. Današnji računari nemaju ugrađenu mogućnost korišćenja konteksta u komunikaciji sa čovekom. Omogućujući računarima pristup kontekstu obogaćuje se komunikacija čoveka i računara. Ovo zahteva precizniju definiciju koncepta sa konceptualnog i tehničkog stanovišta. Konceptualno stanovište podrazumeva samo razumevanje

pojma konteksta, dok tehnički aspekt omogućava integraciju konteksta u principe razvoja softvera. Prvu ovakvu definiciju konteksta dao je Anind Dey [Dey01b] predstavljajući kontekst kao bilo koju informaciju koja može biti korišćena da opiše situaciju nekog entiteta. Entitet je osoba, mesto ili objekat relevantan za interakciju korisnika i aplikacije, uključujući samog korisnika i aplikaciju. Ovakva definicija omogućava preciznije određenje konteksta za dati scenario aplikacije. Ako se neka informacija može iskoristiti za opis situacije učesnika u interakciji, onda ona spada u kontekst.

Usko povezano sa pojmom konteksta jeste pojam kontekstno-osetljivog računarstva (*eng. context-aware computing*). Kontekstno-osetljivo računarstvo (*eng. context-aware computing*) je oblast računarstva koja se kao takva pojavila i razvila u poslednjoj dekadi. Prvu definiciju dali su Schilit i Theimer [Schilit94] koji pod kontekstnim računarstvom podrazumevaju softver koji se adaptira prema svojoj lokaciji upotrebe, skupu korisnika i objekata koji ga okružuju, kao i promenama ovih objekata tokom vremena. Ovakva definicija je suviše specifična. Prva sistematična razmatranja konteksta u oblasti interakcije čoveka i računara bila su zasnovana na kontekstno-osetljivim aplikacijama (*eng. context-aware applications*). Kontekstno-osetljive aplikacije omogućuju korisniku prijem informacija u realnom vremenu na osnovu akcija u fizičkom svetu [Abowd00]. Dey opet daje potpuniju definiciju kontekstnog računarstva opisujući sistem kao kontekstno-osetljiv ako koristi kontekst kako bi korisniku pružio relevantne informacije i/ili servise, gde je relevantnost informacije određena zadatkom korisnika [Dey01b].

U poslednjoj dekadi, u okviru velikog broj časopisa i konferencija objavljivani se rezultati istraživanja povezanih sa kontekstno-osetljivim računarstvom. Na taj način možemo govoriti o većem broju oblasti koje razmatraju kontekstno-osetljivo računarstvo sa različitih aspekata. Radi boljeg razumevanja ovih oblasti, kao i njihovih međusobnih relacija, predložen je okvir za klasifikaciju kontekstno-osetljivih sistema [Hong09]. Okvir je zasnovan na arhitekturi kontekstno-osetljivih sistema koja se sastoji od pet slojeva:

- Konceptualni i istraživački sloj (*eng. concept and research layer*) obuhvata opšte teorije i principe vezane za konstruisanje i funkcionisanje kontekstno-osetljivih sistema. Ovaj sloj zapravo predstavlja ontološku osnovu za izgradnju ostalih slojeva. Sloj sačinjavaju teorije definisanja i reprezentacije konteksta [Dey01b], [Dourish04], principi upravljanja kontekstnim podacima [Dey05], [Mühlhäuser09] i smernice vezane za razvoj i arhitekturu kontektsno-osetljivih sistema [Dey01a], [Anhalt01], [Coutaz05], [Rehman07], [Lee08], [Serral10]. Pored toga, ovde se javljaju i algoritmi iz oblasti veštačke inteligencije koji se pre svega odnose na problem ekstrakcije konteksta visokog nivoa iz konteksta niskog nivoa [Samaan05] i na problem preporuke odgovarajućeg servisa u zavisnosti od konteksta korisnika [Yu06].

- Mrežni sloj (*eng. network layer*) obuhvata mrežne servise i protokole za podršku kontekstno-osetljivim sistemima [Riva07].

- Sloj sistemskog softvera (*eng. middleware layer*) sadrži servise koji vrše prijem i obradu sirovih kontekstnih podataka kako bi se oni mogli koristiti od strane aplikativnih servisa [Zhou09], [Maalej09], [Ho05], [Pawar08], [Kulkarni10].

- Aplikativni sloj (*eng. application layer*) obuhvata različite kontekstno-osetljive aplikacije koje korisnicima obezbeđuju određene servise kao što su mobilne aplikacije [Raento05], [Häkkilä06], [Hong07], [Chang09], sistemi prostorne navigacije [Haan10], pametna mesta stanovanja [Dey09], [Vale09], [Kawsar10] i dr.

- Sloj infrastrukture korisnika (*eng. user infrastructure layer*) obuhvata rešenja vezana za razvoj kontekstno-osetljivih korisničkih interfejsa [Hatala05], [Collignon08], [Calvary09], [Lohmann06], [Hartmann09a].

2.2.2. Postojeća rešenja u oblasti kontekstno-osetljvivih korisničkih interfejsa

Poslednjih godina primetna je pojava korisničkih interfejsa zasnovanih na principima kontekstno-osetljivog računarstva. Do pojave ove vrste korisničkih interfejsa dovelo je i unapređenje tehnologije, kao i pojave novih tehnika interakcije čoveka i računara. U ovom odeljku biće opisana neka od postojećih rešenja u ovoj oblasti.

Moderni mobilni telefoni predstavljaju multifunkcionalne uređaje koji pružaju raznovrsne aplikacije i servise u bilo koje doba na bilo kom mestu. Jedan od problema koji se javlja kod ovih uređaja jeste pravovremen pristup odgovarajućoj aplikaciji zbog kompleksnosti ili duboke strukture menija. Postoji veliki broj tehnika kod standardnih mobilnih telefona zasnovanih na principima MFU (*eng. Most-Frequently-Used*) ili MRU (*eng. Most-Recently-Used*). Međutim, način upotrebe mobilnih telefona zavisi i od situacije u kojoj se korisnik nalazi. U radu [Kamisaka09] je opisan pristup realizaciji kontekstno-osetljivih korisničkih interfejsa za mobilne telefone. Osnovni cilj ovog rešenja je da omogući korisniku pravovremen pristup odgovarajućoj aplikaciji, uključujući i onima koje nisu često korišćene u skladu sa situacijom (kontekstom) u kojem se korisnik nalazi. Korisnički interfejs mobilnog telefona se posmatra kao model prioriteta (*eng. simple order model*). Realizacija kontekstno-osetljivog interfejsa se svodi na određivanje optimalnog uređenja elemenata za poziv aplikacija na osnovu situacije u kojoj se korisnik nalazi. Redosled prioriteta aplikacija predstavlja redosled verovatnoća korišćenja aplikacija u konkretnoj situaciji. Na ovaj način formirana rang lista korišćenja aplikacija određuje izgled interfejsa u konkretnoj situaciji i u kranjoj meri smanjuje broj korišćenih tastera. Sam algoritam određivanja prioriteta zasnovan je na tehnologiji mašinskog učenja (*eng. Machine Learning*).

Upotreba javnih displeja (*eng. public displays*) za prikaz svakodnevnih servisnih informacija danas je široko rasprostranjena. Pored toga što prikazuju informacije različitim korisnicima, ovi displeji moraju vršiti personalizaciju informaciju za pojedinačne korisnike u zavisnosti od konteksta korišćenja. Istraživanja u ovoj oblasti identifikovala su nekoliko problema koji se pre svega odnose na simultanu vizuelizaciju informacija za veći broj korisnika koja će uzeti u obzir i kontekst interakcije. *Adaptive Content Model* [Aranda07] predstavlja predlog softverske arhitekture za upravljanje kontekstom, sadržajem i prikazom korisničkog interfejsa na višekorisničkim javnim displejima. Model definiše troslojnu arhitekturu koju čine sloj konteksta (*eng. Context Layer*), sloj servisa (*eng. Service Layer*) i sloj displeja (*eng. Display Layer*). Sloj konteksta čine tri komponente - komponenta konteksta

identiteta (*eng. Identity Context Component*) koja identifikuje prisustvo korisnika; komponenta konteksta situacije (*eng. Situated Context Component*) koja opisuje lokaciju korisnika; komponenta vremenskog konteksta (*eng. Temporal Context Component*) koje opisuje vreme izvršavanja akcija. Sloje servisa zapravo predstavlja mehanizme opsluživanja korisnika različitim sadržajima kao što su RSS servisi, XML datoteke i URL adrese. Sloj displeja je zadužen za sam prikaz informacija prilagođen potrebama korisnika.

Kontrola mobilnih uređaja pogledom (*eng. eye-tracking control*) predstavlja novu oblast istraživanja u oblasti kontekstno-osetljivih korisničkih interfejsa [Miluzzo10]. Iako koncept kontrole interfejsa pogledom sam po sebi nije nov [Jacob91], masovnija upotreba se ne može očekivati u narednim godinama [Goth10]. Ovde se pre svega javljaju problemi konceptualne prirode. Uopšteno posmatrano, čovek nema tendenciju upravljanja pogledom. Po prirodi stvari, oči predstavljaju ulazni, a ne izlazni ili upravljački uređaj ljudskog tela. Pored toga, nekih od osnovnih principa dizajna korisničkih interfejsa kao što je *Fitts*-ov zakon, nije primenjiv u procesu upravljanja pogledom. Na primer, usmeravanje pogleda oka ka objektima malih dimenzija kao što je koordinata na ekranu ograničava njegovu upotrebljivost. U naučno-istraživačkoj zajednici još uvek ne postoji jasan konsenzus u pogledu principa dizajna ove vrste korisničkih interfejesa [Goth10].

Prelazak alata za upravljanje ličnim informacijama (*eng. personal information management tools*) sa stonog u Web okruženje uslovljava proširenje ovih alata novim mogućnostima zbog izobilja i lake dostupnosti informacija na Internetu. *Atomate* [Kleek10] predstavlja sistem za semantičku obradu RSS/ATOM sadržaja tako što ih integriše u jedinstven RDF (*eng. Resource Description Framework*) model. U kombinaciji sa servisima kao što su *online* kalendari, klijenti elektronske pošte, *feed* servisi i servisi za razmenu poruka, *Atomate* omogućava automatizaciju radnji korisnika. Ove radnje mogu biti kontekstno filtriranje podataka i slanje poruka, kao i aktivnosti vezane za deljenje i koordinaciju u socijalnim mrežama.

Traganje za informacijama (*eng. information seeking*) predstavlja rutinsku aktivnost u različitim sistemima. Ako se u obzir uzme i kontekst pretrage, ova aktivnost postaje još složenija. Pre svega, ona može podrazumevati dobavljanje većeg broja podataka o različitim, ali povezanim entitetima. Takođe, potrebe korisnika za informacijama se mogu menjati u toku same pretrage. Promena pojedinačnog zahteva korisnika može uticati na kontekst postojećih i povezanih zahteva. *CENTAUR* [Lu09] predstavlja softversko rešenje pametnog sistema za hvatanje beleški (*eng. smart notepad system*). Sistem omogućava korisnika da zahtev za pretragu unosi u formi beleški. Unete beleške se dinamički intrerpretiraju radi formiranja zahteva za pretragu. Drugim rečima, sistem semantički modeluje i interpretira unete beleške kao skup zahteva za podacima. Semantičko, tj. kontekstno modelovanje beleški korisnika zahteva postojanje baze znanja u pozadini. Ovu bazu čine model konteksta beleški koji se kreira i ažurira dinamički u toku interakcije sa korisnikom; domenski specifična ontologija i rečnik termina vezanih za beleške korisnika. Na osnovu formiranih zahteva sistem formira izvršne upite. Osnovu za formiranje upita čini skup domenski nezavisnih šablona upita. Mehanizam za formiranje upita je hibridno rešenje, tj. predstavlja kombinaciju jezika zasnovanih na pravilima i proceduralnih jezika. Rezultati pretrage se korisniku prikazuju u formi beleški. Na ovaj način, korisnik

je usmeren na to *šta* traži, a ne *kako*. Unosom beleški korisnik pravi veći broj zahteva istovremeno umesto da eksplicitno formira upit po upit. I na kraju, sistem detektuje ažuriranje unetih beleški i dinamički menja formirane zahteve u skladu sa izmenama.

Ambijentalni korisnički interfejs (*eng. ambient user interface*) predstavlja skup skrivenih inteligentnih interfejsa koji prepoznaju prisustvo korisnika i pružaju servise njegovim neposrednim potrebama. Pravovremeni i inteligentan odgovor korisničkog interfejsa na akcije korisnika zahteva integraciji saznanja i tehnologija iz različitih oblasti kao što su virtuelna i proširena stvarnost (*eng. Virtual Reality – VR, Augmented Reality - AR*), sveprisutno računarstvo i veštačka inteligencija. CCAA (*eng. Context-aware Cognitive Agent Architecture*) predstavlja softversku arhitekturu za razvoj ambijentalnih korisničkih interfejsa [Lee09]. Arhitektura je zasnovana na agentima i vertikalno organizovana u tri sloja. Vertikalna organizacija ne odgovara nivoima apstrakcije, nego slojevima obrade. U tom smislu predviđa sloj proširene realnosti, sloje konteksta i sloj veštačke inteligencije. Rezultat jednog sloja predstavlja ulaz u drugu sloj. Na ovaj način se interakcija korisnika sa ambijentalnim korisničkim interfejsom zasniva na tehnikama iz oblasti kontekstnog računarstva i veštačke inteligencije.

Jedan od problema u oblasti dizajna kontekstno-osetljivih korisničkih interfejsa predstavlja semantičko mapiranje informacije iz konteksta na ulazne elemente korisničkog interfejsa. Naime, kontekstno-osetljivi korisnički interfejsi olakšavaju interakciju sa korisnikom tako što sugerišu ili unapred nude skup podataka formiran na osnovu tekućeg konteksta korisnika. Ovde je potrebno dodatno odrediti koja informacija izvučena iz konteksta se može koristiti kao ulaz određenog elementa korisničkog interfejsa. Istraživanje [Hartmann09b] daje predlog procesa mapiranja informacija konteksta na elemente korisničkog interfejsa. Proces je ograničen na grafičke korisničke interfejse čiji su elementi opisani tekstualno. Pristup je zasnovan na povezivanju informacija konteksta sa elementima korisničkog interfejsa na osnovu mera semantičke sličnosti (*eng. semantic similarity measures*). Svaki element korisničkog interfejsa je opisan skupom tekstualnih termina koji ga bliže određuje. Sa druge strane, kontekst je predstavljen kao kolekcija objekata sa tekstualnim obeležjima. Proces se dakle svodi na semantički povezivanje objekata iz različitih domena upotrebom definisanih algoritama. Ovakav pristup zahteva definisanje početnog skupa podataka za objekte iz oba domena. Praktičnu demonstraciju pristupa predstavlja AUGUR [Hartmann09a], sistem za kontekstnu-osetljivu navigaciju i pretragu u web aplikacijama.

Mashup predstavlja naziv za aplikaciju koja kombinuje funkcionalnosti i podatke većeg broja web sajtova kako bi korisniku pružila servise i omogućila rešavanje zadataka koji nisu predviđeni u originalnim sajtovima. Na ovaj način oni su postali popularan način integracije podataka sa različitih izvora u cilju integracije specifičnih potreba korisnika. Na ovaj način se mogu posmatrati kao tehnologija usko povezana sa kontekstnim računarstvom. Međutim, kreiranje *mahup* aplikacija zahteva pre svega programersko znanje, što dalje ograničava neprogramerske korisnike u smislu korišćenja *mashup*-a koje su kreirali drugi. Kako bi rešili ovaj problem, istraživači i dizajneri rade na kreiranju alata koji omogućavaju kreiranje *mashup* aplikacija bez programiranja, od strane običnog korisnika. Ovi alati moraju obezbediti izdvajanje podataka iz postojećih izvora (web stranice, servisi i

feed izvori), agregaciju podataka iz više izvora u jedan skup i vizuelizaciju podataka u formatu prihvatljivom za korisnika. Jedan od takvih alata predstavlja *Vegemite* [Lin09], *spreadsheet* okruženje koje koristi tenhike direktne manipulacije (*eng. direct-manipulation*) i pogramiranja po primeru (*eng. programming-by-example*) kako bi krajnji korisnik bez programerskog znanja mogao da kreira *mashup*. Sam alat je implementiran kao *plug-in* komponenta *Mozilla* web pretraživača.

Široka dostupnost informacija na Internetu uslovila je pojavu tehnika inteligentnog pretraživanja podataka. Jedna od njih je smislena aktivnost (*eng. sensemaking acitivity*). Najjednostavniji mehanizmi ove vrste koji se sreću kod standardnih Internet pretraživača su *bookmarking*, *Back* dugme i istorijat pregleda stranica (*eng. browsing history*). Drugi inteligentni pristupi mogu proširiti ovaj osnovni skup aktivnosti. Smisleni zadaci podrazumevaju prikupljanje i razumevanje podataka iz različitih izvora kako bi se dao odgovor na složene upite. Ovakve aktivnosti su uobičajene i uključuju, na primer, istraživanje destinacija za odmor ili analizu tržišta. *InsightFinder* [Cheng09] predstavlja sistem za pametno web pretraživanje koji implementira algoritam za kontekstnu preporuku stranica zajedno sa tradicionalnim mehanizmom za hvatanje beleški. Algoritam vrši dinamičku analizu svake posećene stranice i odabir stranica najrelevantnijih za tekući zadatak korisnika. Eksperimentalni rezultati korišćenja sistema ukazuju da *InsightFinder* pretraživač identifikuje relevantan sadržaj unutar web stranice brže od tradicionalnih pretraživača.

2.3. *Arhitekture za modelovanje korisničkih interfejsa*

Softverska arhitektura predstavlja fundamentalnu organizaciju softverskog sistema sastavljenu od modula, relacija između modula i okruženja, i principa kojima je vođen njen dizajn i evolucija [IEEE00]. Posmatrajući istorijat razvoja oblasti dizajna korisničkih interfejsa, najpre se uočava pojava prototipa sistema čiji je cilj bio dolaženje do korisničkih zahteva ili prikaza konačnog korisničkog interfejsa sistema u ranoj fazi razvoja. Jednostavna *cost/benefit* analiza pokazuje da izrada jednostavnijih prototipa ne zahteva velike resurse softverske kompanije i u ovom slučaju softverska arhitektura nije kritičan faktor [Bass98]. Sa druge strane, ukoliko se u razvoj prototipa ulože značajnija sredstva i donese odluka da se prototip pretvori u softverski proizvod nastaju poteškoće. Naime, u slučaju nepostojanja jasno definisanog okvira arhitekture i principa njenog korišćenja, održavanje i proširivanje rezultujućeg softverskog proizvoda postaje otežano. Pored prakse kreiranja prototipa, na pojavu softverskih arhitektura za modelovanje korisničkih interfejsa je uticala i pojava novih tehnika interakcije kao što su:

- *Groupware* sistemi koji omogućavaju većem broju korisnika da rade na istom zadatku istovremeno ili asinhrono, udaljeni jedni od drugih ili na istoj lokaciji [Ellis94].
- Multimodalna interakcija, tj. korisnički interfejsi koji podržavaju veći broj načina komunikacije kao na primer kombinacija govora i pokreta [Oviatt00].
- Sistemi proširene stvarnosti (*eng. Augmented reality systems*) koji kombinuju realan svet i obradu informacija [Crowley00].

- Kontekstno računarstvo (*eng. Context-aware computing*) koje proučava kontekst interakcije u kojoj se nalazi korisnik, koji između ostalog čine faktori kao što su tip i karakteristike korisnika lokacija i uslovi osvetljenja [Crowleyoo].

- Sistemi univerzalnog pristupa (*eng. Universal Accessibility*) projektovani tako da budu upotrebljivi za širok spektar korisnika, uključujući i one sa posebnim potrebama [Shneidermanoo].

Postojeći *off-the-shelf* alati kao što su okviri aplikacija i generatori korisničkih interfejsa donekle mogu nadomestiti problem dizajna arhitekture, ali se oni uglavnom koriste u dizajnu klasičnih GUI aplikacija. Slučajevi u vezi sa gore pomenutim tehnikama interakcije iziskuju reverzni inženjering okvira aplikacije kako bi postojeći kôd mogao biti proširen i ponovno upotrebljen. Sličan problem se javlja i kod generatora korisničkih interfejsa. Programeri moraju imati sveobuhvatan uvid u funkcionalnost generisanog kôda kako bi ga prilagodili specifičnom slučaju. Pored toga, moraju razumeti kako da integrišu kôd koji menjaju sa ostatkom sistema kako bi podržali njegove interne funkcionalnosti. Bez postojanja jasno definisanog okvira arhitekture veoma je teško podržati navedene aktivnosti na odgovarajući način.

U vezi sa prethodnom analizom, modelovanje softverske arhitekture ima dva praktična i komplementarna cilja - direktni dizajn i reverzni dizajn softverskih struktura. Sa jedne strane, arhitektura upravlja razvojem budućeg sistema, dok sa druge strane omogućava razumevanje organizacije postojećeg softverskog sistema.

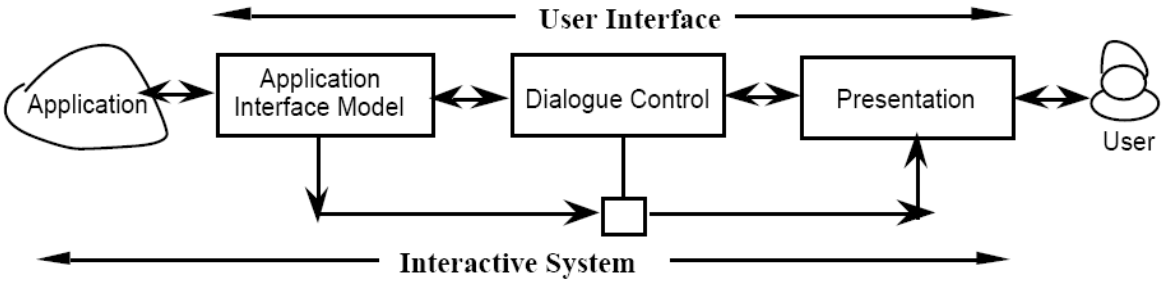
U nastavku će biti opisane softverske arhitekture za razvoj korisničkih interfejsa koje su u svoje vreme našle širu prihvaćenost i primenu u praksi. Ostatak poglavlja je organizovan u četiri celine. Postojeće arhitekture su klasifikovane u tri grupe - objektno-orijentisane arhitekture, arhitekture zasnovane na zadacima i arhitekture za razvoj kontekstno-osetljivih korisničkih interfejsa. Svakoj od arhitektura je posvećeno posebno potpoglavlje. Na kraju je dat odeljak u kojem su opisane neke specifične metodologije razvoja korisničkih interfejsa.

2.3.1. Objektno-orijentisane arhitekture

Na osnovu prethodno iznetih principa softverske arhitekture razmatraćemo konkretne slučajeve razvijene u oblasti projektovanja korisničkih interfejsa interaktivnih sistema. Prve softverske arhitekture za modelovanje korisničkih interfejsa su zasnovane na principima objektno-orijentisane paradigme. Najpre će biti opisani *Seeheim* i *Arch/Slinky* arhitektura. Na osnovu njih su razvijeni multiagentski sistemi, *MVC i PAC*. Osnovni motiv za uvođenje tehnologije inteligentnih agenata jeste povećanje granularnosti i paralelizam u radu sa korisnicima. Na kraju opisujemo arhitekture koje koriste UML kao jezik za modelovanje i predstavljaju prve pokušaje integracije dizajna korisničkih interfejsa u standardizovani proces razvoja softvera.

2.3.1.1. *Seeheim model*

Prvi predlog funkcionalne dekompozicije interaktivnih sistema predstavlja Seeheim model [Pfaff85]. Model struktuira interakciju sa sistemom u tri sloja – sloj interfejsa aplikacije, sloj dijaloga i sloj prezentacije (slika 2.1).



Slika 2.1. *Seeheim model.*

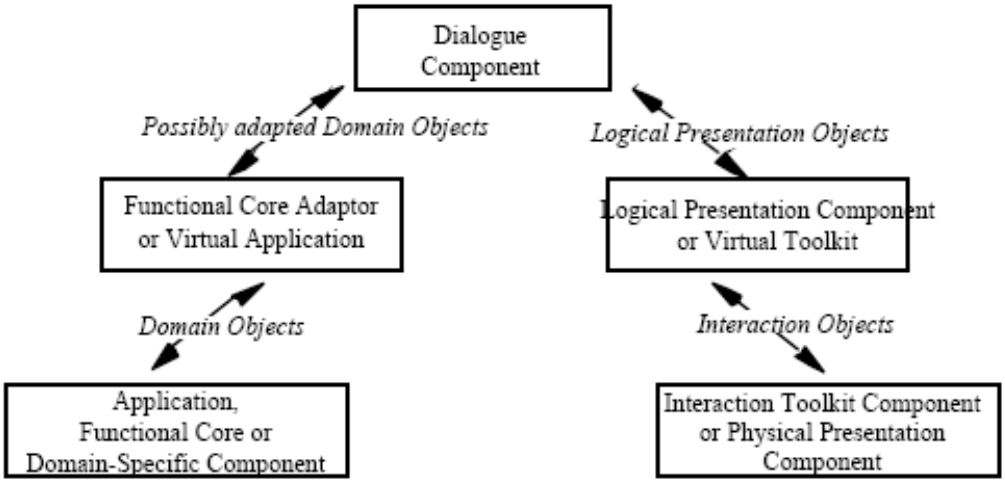
Podsistem aplikacije (*eng. Application*) obuhvata interne funkcionalnosti sistema. Model interfejsa aplikacije (*eng. Application Interface Model*) opisuje semantiku podsistema aplikacije sa stanovišta korisničkog interfejsa. Drugim rečima, opisuje strukture podataka i procedure koje podsistem aplikacije stavlja na raspolaganje korisničkom interfejsu. Model prezentacije (*eng. Presentation*) definiše ponašanje sistema koje vidi i kojim upravlja korisnik. Model kontrole dijaloga (*eng. Dialog Control*) posreduje između prethodna dva. Model predviđa i mogućnost direktne komunikacije aplikacije i prezentacije radi poboljšanja, ali je inicijator ove komunikacije opet model dijaloga. Seeheim arhitektura je vođena principima portabilnosti i modifikacije. Praksa je pokazala da je korisnički interfejs interaktivnog sistema najčešće podložan modifikaciji. Model interfejsa aplikacije predstavlja način razdvajanja izmena korisničkog interfejsa od ostatka sistema.

Glavni nedostatak arhitekture ogleda se u tome što je vođena principom dijaloga korisnika i sistema. Naime, sredinom osamdesetih godina prošlog veka interakcija korisnika i sistema je pretežno razmatrana kao dijalog zasnovan na jeziku. Na ovaj način je uloga svake komponente opisana preko semantičkih, sintaksnih i leksičkih aspekata interakcije. Drugim rečima, interakcija korisnika i sistema je bila zasnovana na formalnim jezicima. Centralnu ulogu u komunikaciji imao je dijalog koji interakciju svodi na strogo definisanu sekvencu zadataka. Međutim, sa pojavom tehnika direktne manipulacije [Jacob86], ovakav pristup ispoljava bitan nedostatak koji se ogleda u formiranju interfejsa sa rigidnim sekvencama zahtevanih akcija. Predefinisani korisnički interfejs koji strogo definiše tok interakcije je u suprotnosti sa konceptom direktnosti u kojem dalji tok interakcije zavisi i od povratne sprege korisnika i sistema zasnovanoj na mehanizmu obrade događaja. U odeljku koji sledi daje se opis arhitekture nastale kao rezultat težnji da se prevaziđe opisani nedostatak.

2.3.1.2. *Arch model*

Arch model [Bass92] predstavlja unapređenje *Seeheim* modela. Model definiše funkcionalnu dekompoziciju sličnu prethodnom, ali sa većim nivoom granularnosti. Unapređenja se ogledaju u uvođenju nivoa apstrakcije komponenti, eksplicitne definicije struktura podataka koje se razmenjuju

između komponenti i adaptacija interfejsa između komponenti kako bi se unapredila portabilnost i olakšale modifikacije sistema (slika 2.2).



Slika 2.2. *Arch model. Strelice odgovaraju tokovima podataka između komponenti.*

Model aplikacije, identično *Seeheim* modelu, obuhvata domenski specifične koncepte i funkcije. Suprotno njemu, komponenta alata interakcije (*eng. Interaction Toolkit Component*) kreira interfejs koncepata i funkcija domena koji čine fizički objekti za interakciju ili vidžeti (*eng. widgets*). Centralna komponenta modela je komponenta dijaloga koja je odgovorna za regulisanje redosleda izvršavanja akcija. Generatori korisničkog interfejsa zasnovani na modelima kreiraju komponentu dijaloga na osnovu specifikacije modela zadataka.

Model je razdvojio nivoe apstrakcija funkcionalnih komponenti opisima struktura podataka koje oni razmenjuju u koje spadaju domenski objekti (*eng. domain objects*), logički prezentacioni objekti (*eng. logical presentation objects*) i fizički objekti za interakciju (*eng. physical interaction objects*). Domenski objekti predstavljaju strukture podataka visokog nivoa koje opisuju domenski specifične koncepte. Logički prezentacioni objekti su apstraktni entiteti koji opisuju platformski nezavisnu prezentaciju domenskih objekata. Na primer, logička prezentacija izbora može biti realizovana različitim fizičkim objektima za interakciju.

Glavne funkcionalne komponente sistema ne razmenjuju podatke direktno, već preko za to posebno dizajniranih komponenti (slika 2.2), adaptera funkcionalnog jezgra (*eng. Functional Core Adaptor - FCA*) i komponente logičke prezentacije (*eng. Logical Presentation Component*). FCA se može posmatrati i kao virtuelni sloj aplikacije. Njegov zadatak je da prilagodi domenske objekte funkcionalnog jezgra mentalnom modelu korisnika o konceptima koje oni opisuju. Komponenta logičke prezentacije razdvaja platformski nezavisnu prezentaciju domenskih objekata od platformski zavisne prezentacije koju određuje specifična tehnologija. Ona je sastavljena od skupa logičkih prezentacionih objekata koji zajedno čine virtuelnu biblioteku komponenti (*eng. virtual toolkit*). Na ovaj način prelazak na specifičnu tehnologiju implementacije zahteva definisanje skupa pravila preslikavanja logičkih u fizičke komponente, dok logički opisi ostaju nepromenjeni. Primer virtuelne biblioteke predstavlja Java AWT (*eng. Abstract Window Toolkit*) [Geary97]. AWT u sebi integriše

mapiranje logičkih vidžeta u fizičke vidžete dostupne na konkretnoj platformi. Iako biblioteka u načelu podržava portabilnost kôda, u praksi ne garantuje konzistentnost korisničkog interfejsa na različitim platformama. Java Swing bibioteka [Geary99] otklanja ovaj nedostatak tako što reimplementira ponašanje komponenti dostupnih na specifičnim operativnim sistemima. Iako adapterske komponente umanjuju efekte promena, one mogu ispoljiti negativan uticaj na efikasnost sistema. U slučaju da su performanse značajnije od portabilnosti komponenta logičke prezentacije se može eliminisati i tada će prezentacija interaktivnog sistema biti direktno opisana jezikom platforme. Takođe, u slučaju da funkcionalno jezgro pruža interfejs koji odgovara potrebama korisnika, adapter funkcionalnog jezgra se može svesti na običan konektor (tj. skup poziva procedura).

Arch model daje opštu funkcionalnu strukturu interaktivnih sistema. Arhitektura je konceptualna i visokog nivoa opštosti. Zamišljena je kao opšti referentni okvir koji treba da prate konkretnija rešenja. Jedan od pokušaja njene konkretizacije jesu modeli zasnovani na agentima.

2.3.1.3. Sistemi zasnovani na agentima

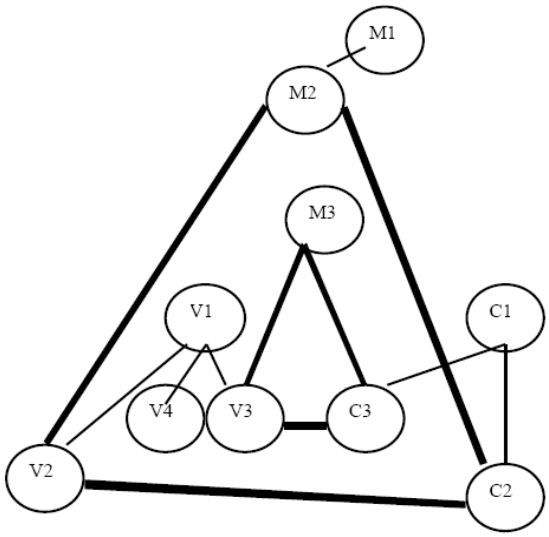
Modeli zasnovani na agentima struktuiraju interaktivni sistem kao kolekciju jedinica obrade koje se nazivaju agenti. Agent poseduje stanje, znanje potrebno za vršenje određenih aktivnosti i sposobnost iniciranja i reakcije na događaje. Agenti koji komuniciraju direktno sa korisnikom se nazivaju i objekti za interakciju ili interaktori. Termini interaktor i agent se ponekad koriste kao sinonimi iako interaktor ne učestvuje u direktnoj interakciji sa korisnikom. Termin objekat se može posmatrati kao koncept ili tennička struktura objektno-orijentisane paradigme. U ovom kontekstu objekat se razmatra kao generički koncept koji obuhvata element obrade koji poseduje stanje.

Koncept agenta korišćen u oblasti dizajna korisničkog interfejsa predstavlja jedan aspekt opšte definicije u oblasti distribuirane veštačke inteligencije. U veštačkoj inteligenciji agenti mogu biti kognitivni ili reaktivni u zavisnosti od sposobnosti rezonovanja i reprezentacije znanja. Kognitivni agenti poseduje sposobnost zaključivanja i donošenja odluka. Suprotno njima, reaktivni agenti imaju ograničenu moć obrade. Njihovo ponašanje specificiraju direktno dizajneri. U interaktivnim sistemima agenti su uglavnom reaktivni.

Svi modeli zasnovani na agentima prate princip Seeheim i Arch modela, ali sa većim nivoom granularnosti. Dok Seeheim i Arch model daju grubu funkcionalnu dekompoziciju interaktivnog sistema (funkcionalno jezgro, dijalog i prezentacija), modeli agenata interakti vni sistem posmatraju kao kolekciju agenata koji sarađuju, gde svaki agent predstavlja minijaturizaciju Seeheim strukture.

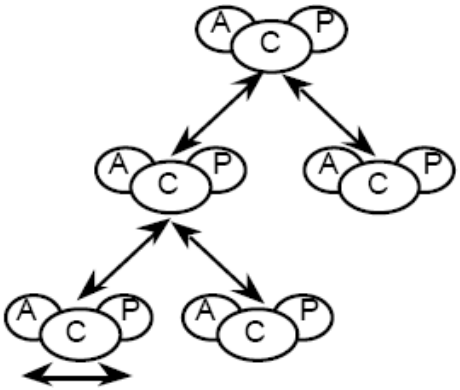
Pored velikog broja razvijenih alata zasnovanih na agentima, nešto širu prihvaćenost i primenu u praksi našli su MVC [Krasner88] i PAC [Coutaz87] modeli.

MVC (*eng. Model, View, Controller*) arhitektura modeluje agenta sa stanovišta tri funkcionalna aspekta: interne strukture, interfejsa i ponašanja. Model (*eng. Model*) definiše stanje agenta. Pogled (*eng. View*) predstavlja spolja vidljivo ponašanje agenta. Kontroler (*eng. Controller*) definiše ponašanje agenta u pogledu reagovanja na ulazne događaje. Pogled i kontroler određuju korisnički interfejs, tj. spolja vidljivo ponašanje agenta (slika 2.3).



Slika 2.3. *Implementacija MVC arhitekture u Smalltalk jeziku. Podebljane linije označavaju relaciju naleđivanja, dok tanke linije označavaju pozive metoda.*

PAC (*eng. Presentation, Abstraction, Control*) arhitektura modeluje agenta kao kompoziciju prezentacije, abstrakcije i kontrole. Prezentacija predstavlja spolja vidljivo ponašanje, apstrakcija funkcionalno jezgro, dok aspekt kontrole obuhvata veći broj oblika zavisnosti. Pored toga što služi kao sprega apstrakcije i prezentacije, kontrolni deo je isključivo zadužen za komunikaciju sa drugim agentima (slika 2.4). Za razliku od MVC modela, hijerarhija agenata je uspostavljena relacijom komunikacije.



Slika 2.4. *PAC arhitektura. Strelice označavaju tokove informacija.*

Posmatrajući dva modela, uočavaju se dve bitne razlike. Dok MVC razdvaja ulazne i izlazne tehnike komunikacije, PAC iste enkapsulira u aspektu prezentacije. Za razliku od PAC modela, MVC eksplicitno ne predviđa posrednika u međuagentskoj komunikaciji i koordinaciji. Izbor stila agentske arhitekture zavisi od skupa kriterijuma konkretnog interaktivnog sistema.

Modeli zasnovani na agentima naglašavaju modularnu organizaciju i distribuirano stanje interakcije između kolekcije agenata koji sarađuju. Modularnost, paralelizam i distribuiranost predstavljaju pogodne mehanizme za podršku iterativnog razvoja korisničkih interfejsa, za implementaciju fizički

distribuiranih korisničkih interfejsa i realizaciju dijaloga sa većim brojem niti kontrole. Agent definiše funkcionalnu jedinicu te postoji mogućnost da modifikacija njegovog ponašanja ne utiče na ostatak sistema. Agent se može tretirati i kao nit aktivnosti korisnika. U slučaju da je nit aktivnosti suviše složena da bi bila realizovana jednim agentom, može se koristiti kolekcija agenata koji sarađuju. Agent se može posmatrati i kao jedinica obrade koja migrira u mrežnom okruženju. Ovo svojstvo omogućava realizaciju višepovršinske (*eng. multi-surface*) interakcije [Rekimoto97] u kojoj korisnik interaguje sa većim brojem fizičkih displeja koji predstavljaju logičku celinu. Pored toga, može se uspostaviti analogija između agentskih modela i koncepata objektno-orijentisane paradigme. Klasa definiše kategoriju agenata gde atributi i operacije klase odgovaraju stanju i servisima koje pruža agent.

Modeli agenti nisu uvek pogodni u praksi zbog ograničenja koja nameće okruženje kao što je heterogenost. Problem kod arhitekture zasnovane na agentima predstavlja odsustvo nivoa apstrakcije. Naime, kod agentskih modela informacije koje agenti dobijaju dalje obrađuje skup agenata pre nego što bivaju prosleđene funkcionalnom jezgru. U suprotnom smeru, agenti transformišu informacije dobijene od funkcionalnog jezgra u spolja vidljivo ponašanje. Koraci pomenutih ulaznih i izlaznih transformacija, zvanih funkcija interpretacije i funkcija prikaza respektivno, nisu struktuirani u jasno definisane nivoe apstrakcije. Jedan od načina da se prevaziđe nedostatak jeste korišćene heuristika i obrazaca. Pored toga, agenti modeluju interaktivni sistem na homogen način: svi funkcionalni aspekti sistema su izraženi jednim, istovetnim stilom. U opštem slučaju, realan sistem nije homogen i javlja se problem interoperabilnosti u slučaju korišćenja agenata.

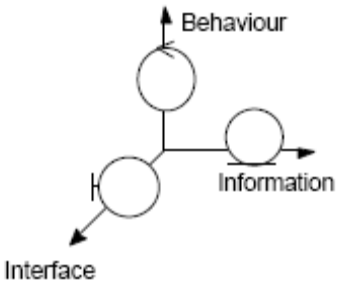
2.3.1.4. Wisdom arhitektura

Jedan od prvih pokušaja integracije koncepata interakcije čoveka i računara u standardni razvojni proces softvera predstavlja Wisdom arhitektura (*eng. Whitewater Interactive System Development with Object Models*) [Nunes00a]. Model polazi od definicije arhitekture interaktivnog softverskog sistema kao opisa elemenata od kojih je sistem sastavljen, strukture i organizacije korisničkog interfejsa, obrazaca njihove strukture i mehanizama interakcije sa korisnikom. Pored toga, HCI metode i tehnike zahtevaju modele za

- opise korisnika i njihovih relevatnih karakteristika,
- opise ponašanja korisnika koji izvršavaju predviđene zadatke,
- specifikaciju apstraktnih (konceptualnih) i konkretnih (fizičkih) korisničkih interfejsa.

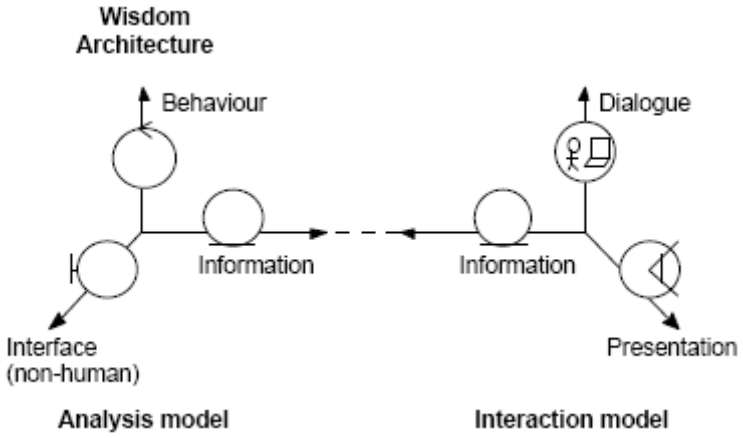
Kao nedostatak postojećeg objedinjenog procesa razvoja softvera (*eng. Rational Unified Process - RUP*) [Jacobson99] autori ističu da granične klase često unapred određuju tehnologiju ili stil korisničkog interfejsa. Isto tako tvrde da je logika specifična za korisnički interfejs pridružena kontrolnim ili graničnim klasama te je na taj način spregnuta sa prezentacionim aspektima sistema ili njegovim internim funkcionalnostima. Na ovaj način se dobija struktura koja nije prilagodljiva promenama i umanjene ponovne upotrebljivosti.

U tom kontekstu, arhitektura daje predlog proširenja RUP metodologije razvoja [Jacobson99]. Naime, polazi se od dimenzija analizacionog okvira objektno-orijentisanog softverskog inženjerstva (OOSE) (slika 2.5).



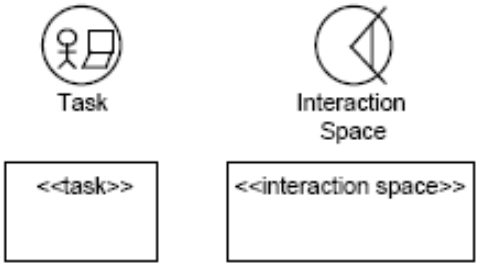
Slika 2.5. *Dimenzije analizacionog OOSE okvira [Nunesoob].*

Prema RUP metodologiji razvoja arhitektura softverskog sistema je vođena slučajevima upotrebe gde termin akter podrazumeva čoveka ali i druge softverske sisteme. Odavde proizilazi priroda slučaja upotrebe koji je orijentisan ka sistemu (*eng. system-centric*). Sa druge strane, korisnički interfejs posreduje u komunikaciji čoveka i softverskog sistema i mora uzeti u obzir karakteristike korisnika i samog okruženje, a u isto vreme omogućiti korišćenje servisa koje sistem pruža. U skladu sa ovim, autori u ranoj fazi razvoja sistema predlažu proširenje prostora analize (slika 2.6).



Slika 2.6. *Wisdom proširenje faze analize [Nunesoob].*

Analizacioni okvir se sastoji iz dva modela, modela analize i modela interakcije. Model analize je preslikan iz RUP metodologije i razmatra internu arhitekturu sistema. Funkcionalni zahtevi su struktuirani pomoću analizacionih klasa, dok se nefunkcionalni zahtevi razmatraju u modelima dizajna i implementacije. Model interakcije analizira organizaciju korisničkog interfejsa. Model struktuira interakciju sa korisnikom pomoću interakcionih klasa, dok se zahtevi u vezi sa stilovima korisničkog interfejsa, tehnologija i drugim ograničenjima obrađuju u modelima dizajna i implementacije. Interakcione klase su uvedene u obliku dva stereotipa klasa (slika 2.7).



Slika 2.7. *Stereotipovi klasa Wisdom modela interakcije [*Nunesoob*].*

Stereotipovi klasa modela interakcije su:

- <<Interaction space>> stereotip – modeluje interakciju sistema i korisnika (čoveka). Klase prostora interakcije su odgovorne za fizičku interakciju sa korisnikom koja obuhvata skup tehnika interakcije koje formiraju izgled sistema (izlaz) i obrada događaja koje generiše korisnik (ulaz). Ove klase izoluju promene u korisničkom interfejsu sistema i često predstavljaju apstrakcije prozora, formi, panela, i dr.
- <<Task>> stereotip – modeluje strukturu dijaloga između čoveka i sistema. Enkapsuliraju složeno ponašanje koje ne može povezano sa specifičnim klasama entiteta. Klase zadataka izoluju promene u strukturi dijaloga korisničkog interfejsa.

Za opis ponašanja korisnika preuzet je opšteprihvaćeni postojeći model zadataka [Paterno99]. Koncepti modela zadataka su uvedeni proširenjem rečnika jezika UML odgovarajućim steretipovima asocijacija, ograničenjima i pridodanim vrednostima za svaki od osnovnih stereotipa klasa i njihovih relacija. Za opise konkretnih oblika ponašanja korisnika prilikom njihove interakcije sa sistemom koriste se djagrami aktivnosti i dijagrami interakcije. Detalji specifikacije arhitekture sa svim navedenim proširenjima mogu se naći u [Nunesoob].

Iako predložena arhitektura uvodi koncepte specifične za razvoj korisničkih interfejsa u domenu analize, ona nije razrađena u kasnijim fazama razvoja sistema. Drugim rečima, ona predstavlja predlog analizacionog okvira za razvoj korisničkih interfejsa upravljan modelima. Pored toga, arhitektura ne predviđa mogućnost modelovanja korisničkih interfejsa na višim nivoima apstrakcije te je otežano uvođenje primitiva za opis okruženja interakcije, platforme interakcije, kao i karakteristika samog korisnika. Arhitektura takođe ne uzima u obzir korišćeni način komunikacije čoveka i računara koji dominantno određuje i vrstu korisničkog interfejsa gde je grafički samo jedan od mogućih tipova.

2.3.1.5. UMLi profil

Predlog UMLi profila za modelovanje korisničkih interfejsa motivisan je nedostacima modelski zasnovanih razvojnih okruženja za korisničke interfejse (*eng. model-based user interface development environments – MB-UIDEs*) razvijenih u to vreme [da Silva03]. Ova okruženja su uglavnom koristila deklarativne opise korisničkih interfejsa. Sa jedne strane, ovakav pristup je omogućavao efikasno obuhvatanje funkcionalnosti korisničkih interfejsa. Sa druge strane, on je pružao ograničene mogućnosti u pogledu modelovanja korisničkih interfejsa zajedno sa drugim aspektima programskog

sistema. Drugim rečima, *MB-UIDEs* nisu bila namenjena opisu funkcionalnosti sistema za koji se kreira korisnički interfejs. Sa obzirom da se ove funkcionalnosti opisuju jezikom UML, ideja je bila da se dizajnerima korisničkog interfejsa i funkcionalnosti softverskog sistema omogući opisivanje modela korišćenjem jedinstvene notacije, tj. jezika UML. Pored toga, ne postoji šire prihvaćeno rešenje za modelovanje korisničkih interfejsa korišćenjem jezika UML.

UMLi profil predstavlja predlog specijalizacije jezika UML za modelovanje korisničkih interfejsa. Profil je specijalizovani metamodel UML jezika i koristi standarde elemente i dijagrame za modelovanje korisničkih interfejsa. Zadaci korisnika se modeluju proširenim dijagramima aktivnosti. Profil takođe u potpunosti opisuje relacije između zadataka korisnika i podataka nad kojima se oni izvršavaju. Sam profil je realizovan kao proširenje *ArgoUML* alata i može se integrisati sa drugim UML modelima.

Strukturna organizacija interfejsa se opisuje apstraktnim prezentacionim modelima (*eng. abstract presentation models*) koji u suštini predstavljaju dijagrame klasa. Oni prikazuju individualne apstraktnie komponente kao gradivne elemente korisničkog interfejsa, i relacije između njih. Korisnik interaguje sa sistemom preko objekata za interakciju (*eng. interaction objects*). Objekti za interakciju se obični klasifikuju kao konkretni i apstraktni. Konkretni objekat za interakciju predstavlja fizičku implementaciju apstraktnog objekta za interakciju. Na primer, meni i padajuća lista predstavljaju konkretizacije apstraktnog objekta za biranje (*eng. abstract chooser interaction object*) jer obe omogućavaju korisniku izvor stavke iz ponuđene kolekcije. Postojeći alati za kreiranje grafičkih korisničkih interfejsa omogućavaju interaktivan izbor i razmeštaj konkretnih objekata za interakciju iz ponuđenih paleti ovih komponenti. Međutim, ova aktivnost je definisana na nivou same implementacije softverskog sistema. Rano uvođenje apstraktnih objekata za interakciju i njihova specijalizacija u kasnijim fazama razvoja softverskog sistema, tj. korisničkog interfejsa predstavlja realnu potrebu. Prvi pristupi su bili zasnovani na korišćenju produkcionih pravila za izbor konkretnih objekata za interakciju na osnovu kriterijuma kao što su rezolucija ekrana, iskustvo korisnika i zahtevana preciznost ulaznih vrednosti. Kako ovi objekti poseduju poseduju određene osobine i operacije, UMLi ih modeluje kao sledeće stereotipe klasa:

- <<Interaction class>> stereotip – korena klasa hijerarhije apstraktnih objekata za interakciju.
- <<Container>> stereotip – predstavlja mehanizam grupisanja apstraktnih objekata za interakciju.
- <<Inputter>> stereotip – apstraktni objekat za interakciju koji prima informacije od korisnika.
- <<Displayer>> stereotip - apstraktni objekat za interakciju koji šalje informacije korisniku.
- <<ActionInvoker>> stereotip – prima instrukcije od korisnika i poziva funkciju sistema.

Na ovaj način je moguće opisati apstraktnu strukturu korisničkog interfejsa preko apstraktnih objekata za interakciju i relacija između. Drugim rečima, opisati korisnički interfejs bez specificiranja konkretne realizacije i izgleda interfejsa za datu platformu.

Ponašanje korisničkog određeno je zadatkom koji korisnik treba da izvrši. Apstraktan opis zadataka predstavlja ključnu aktivnost za veliki broj alata za razvoj korisničkih interfejsa vođenih modelima. Model zadataka se obično modeluje kao struktura stabla kod kojeg su krajnji čvorovi elementarni zadaci, dok višu čvorovi hijerarhije grupišu i opusuju relacije između zadataka čvorova-dece. Ono što se može uočiti kao zajedničko kod mnogih modela zadataka je:

- Hijerarhijska dekompozicija – zadaci višeg nivoa su dekomponovani na manje apstraktne zadatke sistematično.
- Vremenske relacije – redosled izvršavanja elementarnih zadataka kompozitnog zadatka određen je vremenom izvršavanja roditeljskog zadatka.
- Elementarni zadaci – najniži čvorovi modela zadataka su elementarni zadaci. Ove zadatke može izvršavati program ili se mogu izvršavati u interakciji čoveka i računara.

U dotadašnjoj praksi mogla su se uočiti dva pravca u pogledu razmatranja modela zadataka. Prvi pristup posmatra zadatak kao primitivu dizajna koja omogućava identifikovanje ciljeva koje korisnik želi da postigne kada interaguje sa sistemom koji se dalje sastoje od podciljeva. Drugi pristup razmatra zadatak kao primitivu dizajna koja opisuje izvršavanje akcija koje predstavljaju podciljeve. U svakom slučaju, većina istraživača prihvata i ističe modelovanje zadataka na visokom nivou kao centralnu aktivnost u razvoju korisničkih interfejsa. UMLi modeluje zadatke korišćenjem dijagrama slučajeva upotrebe i dijagrama aktivnosti. Dijagrami slučajeva upotrebe identifikuju funkcionalnosti korisničkog interfejsa koje ispunjavaju zahteve, tj. ciljeve korisnika. Dijagrami aktivnosti razrađuju ove funkcionalnosti tako što opisuju akcije koje ih realizuju. Na ovaj način su objedinjena oba pravca posmatranja koncepta zadatka. Važno je napomenuti da UMLi za modelovanje zadataka koristi proširenu notaciju dijagrama aktivnosti. Ovi dijagrami zapravo predstavljaju kombinaciju dijagrama stanja i aktivnosti. Proširenje se ogleda u skupu makroa za modelovanje kategorija ponašanja uobičajenih za korisničke interfejse kao što su opciona ponašanja, ponašanja nezavisna od redosleda i ponavljajuća ponašanja. U okviru dijagrama aktivnosti koriste se i tokovi objekata koji povezuju objekte za interakciju sa aktivnostima i stanjima akcija.

Sa stanovišta objektno-orijentisanih dizajnerskih metrika UMLi predstavlja tehnički potpuniji i napredniji način za modelovanje korisničkih interfejsa u odnosu na standardni UML [da Silva03]. Međutim, može se primetiti da je profil prvenstveno namenjen modelovanju grafičkih korisničkih interfejsa. Profil uvodi dva tipa dijagrama što donekle otežava njegovu integraciju u postojeće alate. Pored toga, profil razmatra korisnički interfejs sa tehničkog stanovišta, tj. tehnologije i ne uzima u obzir faktore kao što su karakteristike korisnika, načini komunikacije, okruženje i uređaji za interakciju.

2.3.2. Arhitekture zasnovane na zadacima

Modeli zadataka (*eng. task models*) su našli značajnu primenu u razvoju korisničkih interfejsa [Paterno00]. Ovo pre svega proizilazi iz činjenice da oni opisuju logičke aktivnosti koje program mora

podržavati. Zadatak predstavlja aktivnost čije izvršenje dovodi do ispunjenja određenog cilja. Cilj može biti promena stanja entiteta ili upit o tekućem stanju. Pored toga, zadaci mogu varirati od onih visokog nivoa apstrakcije (kao što je određivanje strategije za rešavanje problema) do konkretnih zadatka (npr. izbor uređaja za štampu).

Modeli zadataka povezuju oblast dizajna korisničkih interfejsa i formalnih metoda softverskog inženjerstva tako što obezbeđuju mehanizme za opis aktivnosti čijim se izvršavanjem ispunjavaju ciljevi korisnika. Definisanje modela zadataka za konkretan sistem zahteva interdisciplinaran pristup. Na ovaj način će model imati smisla u kontekstu podrške konkretnim aktivnostima. Drugim rečima, korisnički model zadataka (model zadataka koji opisuje izvršavanje aktivnosti sa stanovišta korisnika) i sistemski model zadataka (model zadataka koji opisuje izvršavanje aktivnosti sa stanovišta programa) će biti usaglašeni. Modeli zadataka su uglavnom korišćeni kao podrška različitim fazama životnog ciklusa razvoja softvera.

Jedan od glavnih problema u procesu modelovanja zadataka je da iziskuje znatno vreme i napor. Zbog toga su razvijana okruženja za automatizaciju modelovanja zadataka. Međutim, većina ovih okruženja predstavlja istraživačke prototipe koji omogućavaju editovanje modela zadataka. Sa druge strane, postoji realna potreba za integracijom modela zadataka u proces razvoja softvera. U nastavku će biti opisana neka od rešenja namenjenih dizajnu interaktivnih aplikacija zasnovanih na modelu zadataka.

2.3.2.1. Concurent Task Trees (CTT)

Modelski zasnovani pristupi dizajna korisničkih interfejsa uglavnom su bili bazirani na apstrakovanju objekata koji čine korisnički interfejs, nihovih ponašanja i međusobne povezanosti. Jedan od prvih pristupa zasnovanih na aktivnostima korisnika je GOMS (*eng. Goals, Operators, Methods, Selection Rules*) [John96]. GOMS pristup opisuje sekvence akcija kojima se postižu ciljevi. Ove sekvence se grupišu u metode kojima se pridružuju pravila selekcije koje govore kada je primena metoda preporučena. Međutim, ovaj pristup razmatra samo sekvencijalne zadatke. Pored toga, podrška alata je i dalje ograničena i samim tim je otežana i njihova integracija u standardni proces razvoja softvera.

CTT predstavlja konkurentnu formalnu notaciju za opis korisničkih interfejsa. Notacija je zasnovana na aktivnostima korisnika i omogućava kreiranje hijerarhijskih struktura. Pored toga, uvodi operatore za definisanje vremenskih relacija između zadataka kao što su sekvence, iteracije, konkurentnost i dr. Na ovaj način se mogu predstaviti konkurentne aktivnosti. CTT predviđa četiri tipa zadatka: korisnički zadatak (kognitivne radnje čoveka), aplikativne zadatke (funkcije programskog sistema), zadaci interakcije (akcije korisnika u povratnoj sprezi sa sistemom) i apstraktne zadatke (zadaci koji sadrže podzadatke različitih kategorija). Svaki tip sadrži i odgovarajuće podtipove zadataka. Nakon identifikacije zadataka, mogu se modelovati i objekti nad kojima zadaci rade. Ovde se mogu uočiti dve vrste objekata, objekti korisničkog interfejsa i domenski objekti. Na ovaj način korisnički interfejs se modeluje skupom dijagrama koji opisuju aktivnosti korisnika. Svaki dijagram ima oblik grafa gde su čvorovi aktivnosti, a veze između čvorova relacije (operatori) između zadataka. Detaljan pregled operatora je dat u [Mori02]. Svaki zadatak je opisan skupom atributa kao što su kategorija, frekvencija

izvršavanja, objekti nad kojima manipuliše i procenjeno vreme izvršavanja. Višekorisnički zadaci se modeluju kao hijerarhijske strukture povezane vremenskim operatorima.

TERESA (*eng. Transformation Environment for inteRactivE Systems representAtions*) predstavlja alat koji demonstrira upotrebu notacije za kreiranje korisničkih interfejsa [Mori04]. Alat funkcioniše po principu *One Model, Many Interfaces* [Paterno00]. Ovo znači da se u dizajnu korisničkog interfejsa polazi od modela zadataka. Nakon toga se formira model apstraktnog korisničkog interfejsa koji čine apstraktni objekti interakcije koji izvršavaju prethodno definisane zadatke. Zatim se dolazi do modela konkretnog korisničkog interfejsa koji je određen načinom komunikacije koji će biti korišćen. Na kraju, konkretni korisnički interfejs se transformiše u konačni oblik koji je napisan na specifičnom jeziku. Za kreiranje modela koristi se XML jezik, dok su transformacije između modela realizovane u XSLT tehnologiji. Važno je napomenuti da su transformacije ugrađene u alat i ne mogu se kastomizovati, dakle, nisu definisane eksterno. Pored toga, alat ne podržava kreiranje pararelnih ulaza i izlaza u slučaju kompleksnih dijaloga.

Predloženi pristup opisuje korisnički interfejs sa stanovišta logičkih aktivnosti. Samim tim, on je komplementaran pristupu koji opisuje strukturu i ponašanje objekata korisničkog interfejsa. Svaki od pristupa naglašava aspekte koje ne pokriva drugi. Kao jedan od nedostataka predloženog pristupa važno je napomenuti složenu sintaksu koja nije standardizovana, kao i slabu podršku u pogledu alata. Pored toga, modeli zadataka su često glomazani i nepregledni. Otežana je i integracija kreiranih modela zadataka sa delom koji definiše domensku logiku programskog sistema.

CUP (*eng. Context-Sensitive User Interface Profile*) [Bergh05] predstavlja prvi pokušaj integracije CTT notacije u standardni jezik za modelovanje programskih sistema, tj. UML. Realizovan je u obliku UML profila koji modeluje koncepte CTT notacije kao proširenja UML koncepata aktivnosti i stanja. Rad [Bergh07] opisuje translaciju CTT modela u UML mašine stanja. Na ovaj način profil omogućava specifikaciju ponašanja korisničkog interfejsa u ranim fazama razvoja.

Postoje i pokušaji realizacije dinamičkog izvršavanja modela zadataka u smislu njihove adaptacije akcijama korisnika u vreme izvršavanja programa [Klug05]. Konkretno, pristup proširuje CTT notaciju opisima pojedinačnih zadataka pomoću mašina stanja i uvodi operatore za razmenu informacija između zadataka. I pored semantički bogatijeg opisa zadataka, predložena proširenja dodatno usložnjavaju izradu modela zadataka i njegovu integraciju sa ostatkom sistema, kao i njegovu standardizovanu upotrebu.

U radu je [Sinnig10] je opisano proširenje standardne metodologije razvoja programskih sistema. Proširenje polazi od pretpostavke da slučajevi upotrebe opisuju funkcionalne zahteve sistema, dok modeli zadataka opisuju zahteve vezane za korisnički interfejs. Na taj način se proširenje svodi na integraciju CTT modela zadataka i UML slučajeva upotrebe u fazi prikupljanja zahteva. Konkretno, pojedinačni koraci glavnog i alternativnih scenarija slučajeva upotrebe se proširuju zadacima koji detaljnije razrađuju funkcije sistema u kojima korisnik interaguje sa sistemom. Pristup predviđa integraciju CTT notacije i UML modela u alatima što bi znatno usložnjavalo njihovo korišćenje.

U radu [Barboni10] je opisana HAMSTERS notacija za opise modela zadataka zasnovanih na CTT principima. Notacija se koristi u specijalizovanom okruženju za razvoj interaktivnih aplikacija koje je zasnovano na formalizmu Petri mreža.

U novije vreme javljaju se pristupi koji kombinuju objektno-orijentisane i pristupe zasnovane na zadacima [Wolff09]. U projektovanju korisničkih interfejsa oni polaze od modela zadataka i navigacionih modela koji opisuju vođenje korisnika kroz korisnički interfejs. Navigacioni modeli se još nazivaju i grafovi dijaloga (*eng. dialog graphs*). Čvorovi ovih grafova predstavljaju poglede na zadatke u smislu različite složenosti, hijerarhije i modaliteta, dok su tranzicije usmerene relacije između čvorova. Na ovaj način se jednom modelu zadataka mogu pridružiti različiti grafovi dijaloga i to se može posmatrati kao jedan oblik adaptacije kontektsu korišćenja. Na osnovu početnih modela dolazi se do modela apstraktnog korisničkog interfejsa. Apstraktni korisnički interfejs je sastavljen od elemenata interakcije sa korisnikom visokog nivoa pomoću kojih se realizuju specifični zadaci i koji će kasnije biti preslikani u konkretne komponente korisničkog interfejsa. Svi početni modeli su opisani UML jezikom, dok se transformacije između modela realizuju tehnologijama kao što su EMF (*eng. Eclipse Modeling Framework*) i OAW (*eng. Open Architecture Ware*). Na osnovu apstraktnih korisničkih interfejsa dolazi se do konkretnih korisničkh interfejsa. Oni su vezani za specifičnu tehnologiju implementacije i uglavnom su realizovani korišćenjem deklarativnih jezika kao što su XAML, UsiXML i XUL. Prednost korišćenja deklarativnih jezika za opisivanje korisničkih interfejsa ogleda se u dobro definisanoj gramatici i hijerarhijskoj strukturi.

2.3.3. Arhitekture kontekstno-osetljivih korisničkih interfejsa

U ovoj sekciji će biti opisane arhitekture za razvoj kontekstno-osetljivih korisničkih interfejsa. Pojava ovih arhitektura je uslovljena razvojem kontekstnog računarstva, odnosno kontekstno-osetljivih korisničkih interfejsa. Razvoj novih uređaja i tehnologija za interakaciju uslovio je realnu potrebu da se u dizajn korisničkih interfejsa uvrste faktori kao što su karakteristike okruženja i uređaja za interakciju. Na ovaj način oblast je usko povezana sa svepriosutnim računarastvom. Sa druge strane, u poslednje vreme u obzir se sve više uzimaju i faktori korisnika. Ovako se oblast povezuje sa tehnikama interakcije čoveka i računara. U odeljku koji sledi biće opisan referentni okvir namenjen razvoju kontekstno-osetljivih korisničkih interfejsa. Okvir je konceptualne prirode i zapravo opisuje metodologiju razvoja kontekstno-osetljivih interfejsa na visokom nivou. Pored toga, ukratko će biti opisana neka značajnija rešenja zasnovana na okviru.

2.3.3.1. Cameleon Reference Framework

Cameleon referentni okvir predstavlja konceptualnu softversku arhitekturu interaktivnih računarskih sistema [Demeure05] i služi kao referenca za klasifikaciju korisničkih interfejsa koji podržavaju veći broj ciljnih platformi, ili konteksta upotrebe u oblasti kontekstnog računarstva [Calvary03]. Okvir vrši dekompozciju konteksta upotrebe u tri kategorije: krajnji korisnici interaktivnog sistema, hardverska i softverska platforma preko kojih korisnik vrši interaktivne zadatke i fizičko okruženje u kojem se

korisnik nalazi. U skladu sa tim, uvodi se pojam kontekstno-osetljivog (*eng. context-sensitive*) interfejsa koji se definiše kao korisnički interfejs koji ispoljava svest o kontekstu i na osnovu toga reaguje na promene u istom. Rad [Calvary03] daje opis kontekstno-osetljivih korisničkih interfejsa pre nego preporuke različitih načina ili metoda za različite faze razvoja. Okvir struktuira životni ciklus razvoja u četiri nivoa apstrakcije: zadatak i koncepti (*eng. Tasks and Models*), apstraktan korisnički interfejs (*eng. Abstract User Interface - AUI*), konkretan korisnički interfejs (*eng. Concrete User Interface - CUI*) i finalni korisnički interfejs (*eng. Final User Interface - FUI*). Identifikacija četiri nivoa apstrakcije i uspostavljanje hijerarhije između njih je izvršeno na osnovu njihove nezavisnosti u odnosu na kontekst u kojem se koristi finalni korisnički intefejs. Drugim rečima, modeli koncepata i zadataka su računski nezavisni, apstraktni korisnički interfejs je nezavisan od načina komunikacije, dok je konkretni korisnički interfejs nezavisan od tehnologije implementacije. Pomenuti nivoi su povezani relacijom reifikacije (specijalizacije) od apstraktnog ka konkretnom nivou i relacijom apstrakcije od konkretnog ka apstraktnom nivou.

Okvir razlikuje dve vrste konteksta upotrebe:

- Prediktivni kontekst (*eng. predictive context*) koji se predviđa u fazi dizajna.
- Efektivni kontekst (*eng. effective context*) koji se zaista javlja u vreme izvršavanja.

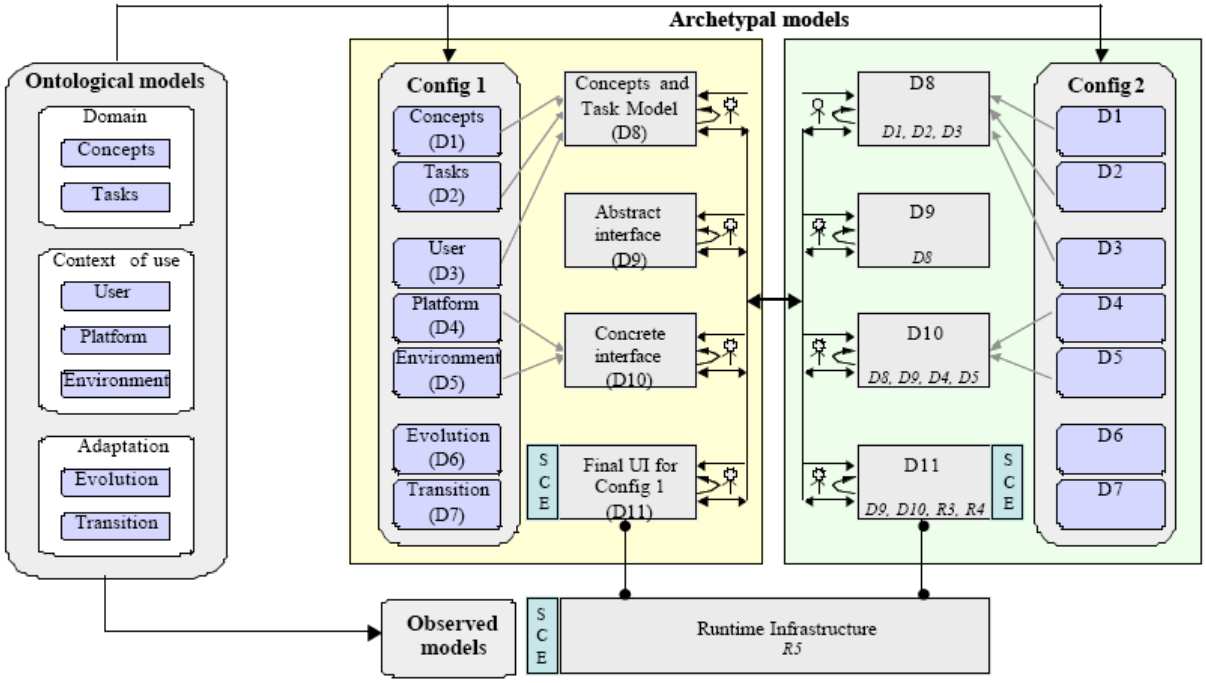
Prediktivni kontekst se javlja u fazi dizajna korisničkog interfejsa. On služi kao prototip za definisanje domena upotrebljivosti korisničkog interfejsa. U vreme izvršenja, prediktivni kontekst može obuhvatiti skup efektivnih konteksta upotrebe. Na primer, prediktivna PDA platforma može postojati na Palm, Psion ili IPAQ uređaju.

Strukturna i funkcionalna organizacija *Cameleon* je prikazana na slici 2.8. Ona se sastoji od sledećih celina:

- Skup ontoloških modela (*eng. ontological models*) koji čine osnovu za postojanje arhetipskih (*eng. archetypal*) i modela posmatranja (*eng. observed models*). Arhetipski modeli su prediktivni modeli koji služe kao ulaz u fazu dizajna. Modeli posmatranja su efektivni modeli koji vode proces adaptacije u vreme izvršavanja. Ontološki modeli su generičke prirode i dodatno klasifikovani kao:
    - Modeli domena (*eng. domain models*) opisuju koncepte i zadatke korisnika u odnosu na domen.
    - Modeli konteksta (*eng. context models*) opisuju kontekst upotrebe terminima korisnika, platforme i okruženja.
    - Modeli adaptacije (*eng. adaptation models*) specificiraju reakciju u slučaju promene konteksta i dobro definisan postupak prilagođavanja novonastalom kontekstu.
- Razvojni proces (gornji desni deo) koji opisuje razvoj korisničkog interfejsa za dati arhetipski kontekst upotrebe. Ovaj deo spada u fazu dizajna.

- Izvršni proces (donji desni deo) koji opisuje način saradnje i integracije korisničkih interfejsa i izvršne infrastrukture kako bi sistem bio prilagođen specifičnom kontekstu upotrebe.

Većina metoda i alata može biti klasifikovana i međusobno poređena koristeći predloženi okvir kao referencu. Pored toga, okvir opisuje kada, gde i kako se promena konteksta interakcije odražava u kontekstno-osetljivom korisničkom interfejsu zahvaljujući relaciji translacije.



Slika 2.8. *Cameleon – struktura i modeli* [Calvary03].

Cameleon predstavlja pristup klasifikaciji i opisu modela, metoda i procesa razvoja korisničkih interfejsa za veći broj konteksta upotrebe, tj. višekontekstnih korisničkih interfejsa. Kontekst upotrebe se dekomponuje u tri dela: korisnik, računarska platforma i okruženje u kojem korisnik obavlja interaktivne zadatke sa specifiranom platformom. Svaka varijacija ovih delova se može posmatrati kao promena konteksta koja se na određeni način može ili mora odraziti na korisnički interfejs.

Objedinjeni razvojni okvir pokušava da ponudi mehanizam za sveobuhvatno razumevanje i zajedničku prezentaciju šema modela, metoda i procesa vezanih za razvoj korisničkih interfejsa. Ovo se pokušava postići na više načina:

- Specifikacijom modela koji se mogu koristiti (na primer, zadaci, koncepti, prezentacija, dijalog, korisnik, platforma, okruženje, objekti za interakciju).
- Specifikacijom faze u kojoj se modeli koriste (vreme dizajna ili izvršavanja).
- Definisanjem mesta na kojem se koriste (definisanjem četiri nivoa: zadaci i kocepti, AUI, CUI, FUI).
- Određivanjem procesa za koji se koristi odgovarajući model.

- Određivanjem toka direktnog inžnjerstva (od najvišeg sloja apstrakcije (zadaci i koncepti) ka nižim (konačni korisnički interfejs) i reverznog inženjerstva. Tokovi reinženjeringa se takođe mogu definisati.

- Definisanjem relacija između nivoa apstrakcije: reifikacija (direktno inženjerstvo), apstrakcija (reverzno inženjerstvo) i translacija (migracija iz jednog konteksta u drugi na istom nivou apstrakcije).

- Različita mesta ulaska u proces razvoja omogućavaju da isti bude iniciran na bilo kom nivou apstrakcije okvira, tj. ne samo na vrhu (*top-down* pristup) ili dnu (*bottom-up* pristup).

- Na kraju, posebna pažnja je posvećena uvođenjem struktura za izražavanje mehanizama izvršavanja.

Pored navedenih pogodnosti, autori okvira ističu otvorenost predloženog rešenja u smislu identifikovanja neotkrivenih i neistraženih delova okvira kako bi dalje bio proširivan i unapređivan. Kombinacija ulaznih tačaka procesa razvoja korisničkih interfejsa, identifikovanih modela i relacija obogaćuje okvir u smislu mogućnoti izražavanja mogućih konfiguracija interaktivnih sistema. Važno je napomenuti da je okvir poslužio i kao osnova za formiranje posebne vrste modelski zasnovanih (*eng. model-based*) jezika za opis korisničkih interfejsa koji imaju modularnu strukturu i čija je semantika opisana odgovarajućim metamodelima, dok je sintaksa uglavnom deklarativna. Primeri ovih jezika, kao što su UsiXML [Limbourg04] i MARIA [Paterno09], biće ukratko opisani u potpoglavlju 2.3.

Cameleon okvir je najčešće upotrebljavan u kontekstu problema migracionih (*eng. migratable*) i distrubuiranih (*eng. distributed*) korisničkih interfejsa. Naime, u eri sveprisutnog računarstva korisnički interfejs više nije vezana isključivo za stonu platformu, već mora posedovati svojstvo migracije i biti distribuiran na dinamičkom skupu raznovrsnih resursa za interakciju. Jedan takav pristup definiše polazni metamodel koji obuhvata fizički i logički domen za razvoj korisničkih interfejsa koji ispoljavaju navedena svojstva [Demeure05]. Ključne apstrakcije oba domena su izvedene iz zajedničkog, konceptualnog domena. Fizički domen obuhvata računarske resurse, i ulazne i izlazne uređaje preko kojih se ostvaruje interakcija. Logički domen se sastoji od tri celine – komponente korisničkog interfejsa, komponente funkcionalnog jezgra i komponente mapiranja. Komponente mapiranja su dodatno klasifikovane u dve kategorije – komponente mapiranja korisničkog interfejsa i funkcionalnog jezgra (konektori), i komponente mapiranja logičkog i fizičkog domena. Mapiranje između fizičkog i logičkog domena se ostvaruje na nivou elemenata korisničkog interfejsa i ulaznih i izlaznih uređaja. Ključna apstrakcije konceptualnog domena je prostor (*eng. space*). U konkretnom slučaju ona podrazumeva matematičku definiciju koordinatnog sistema, kao što je dvodimenzionalni prostor displeja. Ona je dodatno specijalizovana u zone (*eng. zones*) koje označavaju specifične delove prezentacije preko kojih se obavlja interakcija (npr. kontrola korisničkog interfejsa kao što je okvir ili dugme). Svaki od domena nasleđuje ove apstrakcije i prilagođava na sebi svojstven način. Pored toga, u okviru konceptualnog domena modelovana je i apstrakcija zadatka koja ostvaruje vezu i sa komponentama korisničkog interfejsa i sa komponentama funkcionalnog jezgra. Upotreba predloženog metamodela je demonstrirana na primeru korišćenja *CamNote* softverskog

paketa za pregled slajdova prezentacije na dve različite platforme, PC i PDA, kao i kombinacije ova dva tipa uređaja. Metamodel je napisan na UML jeziku, ali je razrađen korišćenjem pseudo-jezika i specifične notacije, tako da pristup nije u potpunosti formalno dosledan i jedinstven.

Višeciljni korisnički interfejs je sastavljen od serija povezanih varijacija jednog korisničkog interfejsa prilagođenih različitim platformama ili kontekstima korišćenja. U procesu adaptacije korisničkog interfejsa u većem broju konteksta potrebno je voditi računa o očuvanju upotrebljivosti prilikom promene konteksta korišćenja. *PlastiXML* [Collignon08] je alat koji implementira model i tehnike prezentacije za opis i manipulaciju domena plastičnosti korisničkog interfejsa. Domen plastičnosti obuhvata skup konteksta upotrebe koje korisnički interfejs podržava tako da očuva njegovu upotrebljivost. Pri tome se alat fokusira na jedan aspekt konteksta upotrebe – veličinu prozora grafičkog korisničkog interfejsa. Sam domen je definisan kao mašina stanja koje obuhvata skup mogućih prezentacija i tranzicija između njih. Model omogućava definiciju površine prozora u terminima veličine i pozicije prozora. Tehnika vizuelizacije omogućava pregled skupa prezentacija koje odgovaraju dostupnoj površini prozora, i opažanje operacija pomoću kojih se prelazi sa jedne prezentacije na drugu u slučaju promene konteksta, tj. veličine prozora. Alat je zasnovan na UsiXML jeziku za opis korisničkih interfejsa. Na ovaj način on omogućava dizajn višeprezentacionih korisničkih interejsa specifikacijom različitih prezentacija i mehanizma za prelazak sa jedne prezentacije na drugu u slučaju promene veličine prikaza. Alat generiše i programski kôd za nekoliko platformi na osnovu UsiXML specifikacija. I pored navedenih prednosti i novina, alat razmatra relativno jednostavan primer promene kontekst upotrebe (promena veličine prozora). Složenije promene konteksta kao što su promene vezane za okruženje ili korisnika, kao i proces adaptacije korisničkog interfejsa nisu obuhvaćene pristupom. Sličan alat, ali specijalizovan za grafičke korisničke interfejse predstavlja GrafiXML [Michotte08].

2.3.4. Ostale arhitekture za razvoj korisničkih interfejsa

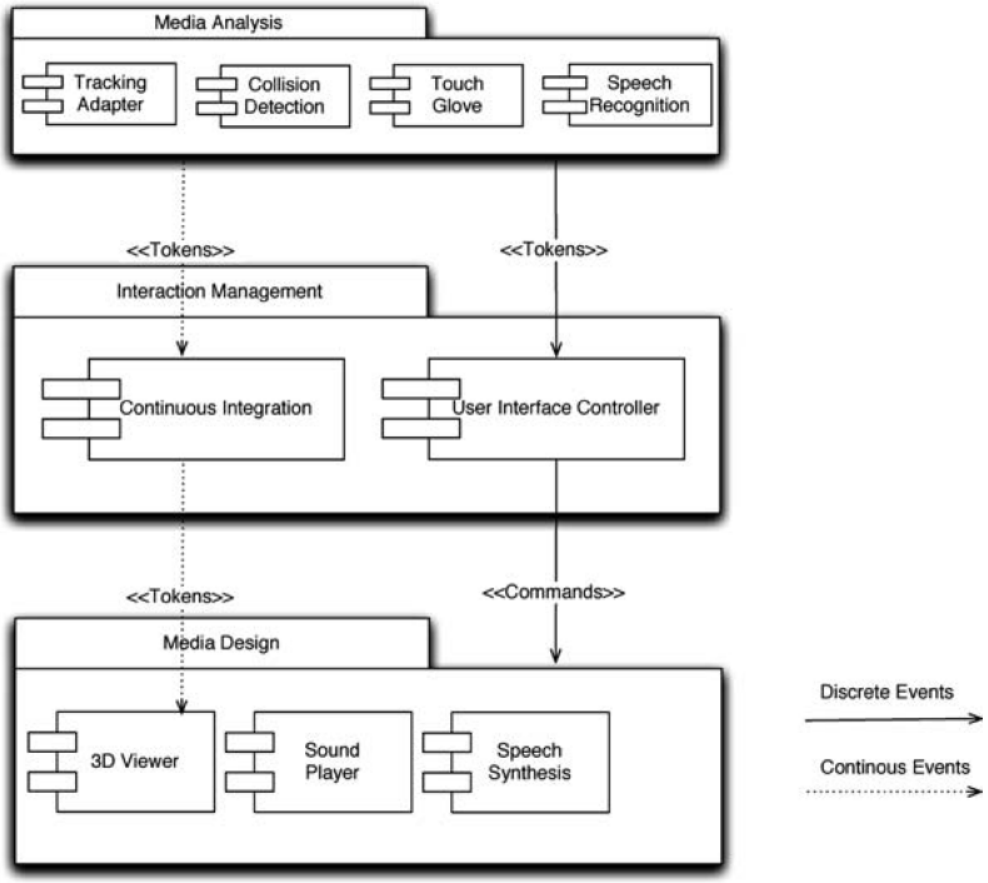
U ovom odeljku će biti opisane istraživačke arhitekture za razvoj specifičnih vrsta korisničkih interfejsa. Najpre će biti arhitektura za razvoj korisničkih interfejsa koji se koriste u kontekstu proširene stvarnosti. Nakon toga će biti opisana dva pristupa razvoja grafičkih korisničkih interfejsa. Jedan pristup razmatra razvoj sa stanovište principa softverskog inženjerstva, dok drugi razmatra dizajn interfejsa sa stanovišta raznolikosti korisnika, kao i korisnika sa posebnim potrebama.

Koncepti vezani za razvoj korisničkih interfejsa, kao što su višenačinska komunikacija, mobilnost korisnika i trodimenzionalna kombinacija realnog i virtuelnog sveta, posmatraju interakciju čoveka i računara sa različitih aspekata. Svaki od pristupa naglašava specifičan kontekst interakcije u odnosu na druge pristupe, dok zajedno daju celovitu predstavu interakcije čoveka i računara. Dizajneri interakcije moraju odabrati odgovarajući stil interakcije iz skupa različitih stilova koji stoje na raspolaganju. Sa druge strane, mora postojati softverska infrastruktura koja realizuje izabrane stilove interakcije. Pored toga, ona mora omogućiti dizajnerima brzo i jednostavno testiranje stilova interakcije, kao i dodavanje novih stilova interakcije. Predlog ovakve infrastrukture opisan je u radu [Sandor05]. *DWARF* (*eng. Distributed Wearable Augmented Reality Framework*) predstavlja okvir

za razvoj korisničkih interfejsa u sveprisutnoj proširenoj stvarnosti *(eng. Ubiquitous Augmented Reality - UAR*). Pojam *UAR* podrazumeva kombinacije korisničkih interfejsa vezanih za:

- Višenačinsku komunikaciju – višenačinski i multimedijalni korisnički interfejsi.
- Mobilnost korisnika – ovde se razmatraju WUI (*eng. Wearable User Interfaces*) koji se sastoje od lakih i prenosivih uređaja koji postaju sastavni deo odeće korisnika [Schmidt00].
- Interakciju ugrađenu u realan svet – ovde spadaju dodirni korisnički interfejsi i interfejsi proširene stvarnosti (*eng. Augmented Reality*) koji naglašavaju prezentaciju informacija u trodimenzionalnom prostoru u zavisnosti od trenutne pozicije korisnika.

DWARF ima komponentnu strukturu i zapravo definiše funkcionalnu organizaciju korisničkog interfejsa (slika 2.9).



Slika 2.9. *Funkcionalna dekompozicija DWARF komponenti* [Sandor05].

Podsistem analize medija (*eng. Media Analysis*) vrši transformaciju fizičkog ulaza korisnika, tj. ulaznih tokova podataka u apstraktne tokene, po analogiji sa leksičkom analizom koju vrši kompajler. Podsistem za upravljanje interakcijom (*eng. Interaction Management*) interpretira tokene, vrši integraciju tokena iz različitih ulaznih kanala i određuje izlaz koji će biti prezentovan korisniku. Integracija može biti kontinualna (kombinacija tokena koji uzimaju realne vrednosti u određenom opsegu kao što je pozicija miša) ili diskretna (vezana za uređaje koji generišu diskretne vrednosti kao

što je uređaj za prepoznavanja govora koji generiše tokene izgovorenih reči). Mehanizam rada podsistema zasnovan je na Petri mrežama. Sami modeli su napisani na XML jeziku. Podsistem dizajna medija (*eng. Media Design*) transformiše obrađene tokene i prosleđuje izlaznim komponentama. Okvir razlikuje četiri tipa tokena: analogne vrednosti ograničenog (rotacije) ili neograničenog (translacija) opsega, i diskretne vrednosti tipa Boolean (pritisak na dugme) ili tekstualne niske (izlaz procesa prepoznavanja govora). Osnovna namena okvira je brzo kreiraje prototipa (*eng. rapid prototyping*) korisničkih interfejsa koji kombinuju različite uređaje za interakciju i realizovan je na većem broju platformi.

Jedan od glavnih problema u formalnim metodama za specifikaciju korisničkog interfejsa je složenost njihove upotrebe. Učenje jezika formalne specifikacije iziskuje napor. Tehnike formalne verifikacije takođe zahtevaju visok stepen poznavanja korišćene notacije. Korišćenje formalnih gramatika u dizajnu korisničkih interfejsa nije naišlo na širu primenu jer ne poseduju konstrukcije za rešavanje problema kao što su paralelizam, mehanizmi komunikacije i sinhronizacije, kao i integracija aspekata strukture podataka i kontrole korisničkog interfejsa. VEG (*eng. Visual Event Grammars*) predstavlja formalnu gramatiku za specifikaciju, dizajn i implementaciju grafičkih korisničkih interfejsa [Berstel05]. Pristup proširuje tradicionalne BNF gramatike kako bi ih učinio prihvatljivijim i pogodnim za modelovanje dijaloga korisničkih interfejsa. VEG specifikacije su platformski nezavisne i mogu se integrisati sa standardnim bibliotekama za razvoj korisničkih interfejsa. Pored toga, one mogu biti verifikovane pomoću alata za proveru modela koji testiraju konzistentnost i ispravnost modela, detektuju zastoje i nedostižna stanja, i generišu testove za validaciju. VEG generiše i programski kôd aplikacije. Razvijen je alat za unos i editovanje formalnih gramatika u obliku sintaksnih dijagrama sa specifičnom vizuelnom notacijom. Gramatike se interno kreiraju u obliku XML dokumenata za potrebe razmene modela. Sam alat je realizovan u Java programskom jeziku i pored vizuelnog editora VEG modela, sadrži parsere, alate i biblioteke za povezivanje modela sa specifičnim platformama implementacije. Tehnika je demonstrirana na primeru aplikacija kao što je editor u Notepad stilu, biblioteka za kreiranje grafova i softver za upotrebu u medicini. Osnovni doprinos predložene tehnike se ogleda u kombinaciji mehanizama klasičnog dizajna grafičkih korisničkih interfejsa, programskih prevodilaca, softverskog inženjerstva i formalne verifikacije. Na ovaj način su kombinacijom različitih tehnologija podržane sve faze razvoja GUI, tj. specifikacija, dizajn, verifikacija i validacija, povezivanje sa aplikacijom i kôdiranje.

SUPPLE [Gajos10] predstavlja sistem za automatsko generisanje personalizovanih grafičkih korisničkih interfejsa. Generisani interfejs je prilagođen uređajima, zadacima, navikama i karakteristikama korisnika. U tom pogledu pristup naglašava dizajn za osobe sa posebnim potrebama, i to prvenstveno u pogledu korišćenja motoričkih funkcija i funkcija vida [Gajos07]. Jedan je od sistema koji na specifičan način uvodi faktore korisnika u proces dizajna korisničkih interfejsa. Formalno gledano, generisanje korisničkog interfejsa se svodi na problem optimizacije postojećeg skupa rešenja korišćenjem funkcija koštanja. Dinamička personalizacija kreiranih interfejsa je takođe podržana. Sistem za automatsko generisanje korisničkih interfejsa se oslanja na funkcionalnu specifikaciju interfejsa (*I*) koje definiše funkcionalosti koje interfejs stavlja na raspolaganje korisniku, model uređaja (*eng. device model*) (*D*) koji opisuje mogućnosti i ograničenja platforme na kojoj će

interfejs biti generisan, i model upotrebe (*eng. usage model*) koji se sastoji od tragova korišćenja (*eng. usage trace*) (*T*) koji opisuju različite načine korišćenja interfejsa.

Funkcionalna specifikacija korisničkog interfejsa je izražena preko tipova podataka koji se razmenjuju između aplikacije i korisnika (*eng. data-oriented*). Ovo je u suprotnosti sa specifikacijama zasnovanim na zadacima (*eng.task-oriented*). Svaki element funkcionalne specifikacije (jedinice informacija) se mapira na određeni element korisničkog interfejsa. Elementi funkcionalne specifikacije mogu biti primitivni (predstavljeni osnovnim tipovima podataka) i kontejneri (semantičko grupisanje primitivnih elemenata koje omogućava njihovu ponovnu upotrebljivost). Vrednosti svakom od pomenutih elemenata se može pridružiti skup ograničenja, pa su uvedeni i elementi sa ograničenjima kao poseban tip. Ograničenja se mogu specificirati u vreme dizajna i menjati u vreme izvršavanja. Atributi ograničenja se određuju prilikom definisanja odgovarajućeg tipa elementa. Tipovi se mogu nasleđivati. Poseban tip elemenata funkcionalne specifikacije predstavljaju vektori. Oni predstavljaju uređene kolekcije određenog tipa i uvedeni su kako bi podržali višestruke selekcije. I na kraju, model funkcionalne specifikacije uvodi tip elementa akcija koji omogućava pozive metoda aplikacije, za razliku od ostalih tipova koji opisuju stanje aplikacije. Ovaj tip elementa ima karakterističnu strukturu koju čine parametri (drugim rečima, tip objekta koji sadrži parametre akcije) i povratni tip (drugim rečima, komponenta korisničkog interfejsa koja će biti prikazana po izvršenju akcije). Oba elementa tipa mogu imati nultu vrednost u zavisnosti od vrste konkretne akcije nad korisničkim interfejsom.

Model uređaja sadrži elemente koji modeluju postojeće elemente korisničkog interfejsa (*eng. widget*) na datom uređaju i elemente ograničenja specifičnih za dati uređaj. Pod vidžetima se podrazumevaju objekti koji pretvaraju elemente funkcionalne specifikacije u komponente prikazanog interfejsa. Po analogiji sa odgovarajućim elementima, vidžeti mogu biti primitivni i kontejnerski. Ograničenja vezana za uređaj predstavljaju funkcije koje postavljaju čitav ili parcijalan skup element-vidžet relacija na jednu od logičkih vrednosti (na primer da li interfejs prevazilazi veličinu ekrana). Da bi se unapredila upotrebljivost vidžeta uvedeni su i dodatni parametri za opis – minimalna ciljna veličina (ograničava minimalnu veličina vidžeta koja se može kontrolisati pokazivačem) i minimalna veličina vizuelnih znakova (ograničava minimalnu veličinu vizuelnih elemenata kao što su font i slike).

Model korišćenja zapravo predstavlja način za uključivanje faktora korisnika u dizajn korisničkog interfejsa. Ovde se polazi od pretpostavke da većina korisnika koristi mali podskup funkcija aplikacije, tako da različiti korisnici koriste različite podskupove. Na ovaj način, adaptacija interfejsa korisniku i dugoročnim obrascima korišćenja se svodi na prikaz koji obezbeđuje olakšanu manipulaciju i navigaciju važnim funkcionalnostima, tj. onima koje konkretan korisnik uglavnom koristi. Drugim rečima, SUPPLE se oslanja na tragove korišćenja, koji mogu odgovarati pravoj ili predviđenoj upotrebi. Tragovi korišćenja daju frekvenciju interakcije sa primitivnim vidžetima, kao i frekevencije tranzicija između različitih elemenata interfejsa. U kontekstu optimizacije procesa adaptacije, oni pružaju mogućnost računanja očekivane cene u pogledu predviđene upotrebe. Trag ili obrazac korišćenja predstavlja skup tragova, gde se pod tragom podrazumevaju koherentne sekvence elemenata kojima korisnik manipuliše (ovde se misli na apstraktne elemente funkcionalnih modela). Trag se modeluje kao sekvenca događaja, gde je svaki događaj modelovan tripletom <element

interfejsa, stara vrednost elementa, nova vrednost elementa>. Dakle format traga je nezavisan od platforme.

Dakle, osnovni cilj je prikaz elementa funkcionalne specifikacije pomoću odgovarajućeg vidžeta. Pri tome se mora poštovati odgovarajući skup ograničenja. Može postojati veći broj preslikavanja element-vidžet, pa se zbog toga uvodi funkcija koštanja radi pronalaženja optimalnog rešenja. Funkcija koštanja predstavlja kvantitativnu metriku kvaliteta korisničkog interfejsa. Funkcija koštanja se definiše za bilo koju meru kvaliteta korisničkog interfejsa, kao što je konzistentnost sa navikama korisnika ili očekivana brzina upotrebe. Na kraju, problem generisanja interfejsa se svodi na problem optimizacije, drugim rečima pronalaženje minimalne cene funkcije koštanja za dati profil korisnika, funkcionalnu specifikaciju korisničkog interfejsa i dostupne uređaje.

Jedna od važnih odlika sistema je i da podržava dinamičku personalizaciju automatski generisanih interfejsa. Personalizaciju može inicirati sistem (*eng. system-driven*) ili korisnik (*eng. user-driven*). Personalizacija koju vrši sistem može biti dvojaka. Jedan način podrazumeva korišćenje dugoročnih obrazaca korišćenja interfejsa od strane konkretnog korisnika. Drugi način je kratkoročan i odnosi se na specifičan tekući zadatak. Zasnovan na principu deljenog interfejsa (*eng. Split Inteface*) [Gajos06]. Princip predviđa da se često korišćena funkcionalnost (koja može biti teže pristupačna) kopira na odgovarajuće mesto u okviru glavnog interfejsa, dok izgled ostatka glavnog interfejsa ostaje nepromenjen. SUPPLE sadrži mehanizam kastomizacije koji omogućava dizajnerima i krajnjim korisnicima da vrše eksplicitne promene na korisničkom interfejsu kao što su, na primer, preuređivanje vidžeta, dupliciranje funkcionalnosti, uklanjanje vidžeta i ograničavanje izbora vidžeta vezanih za prikaz dela funkcionalne specifikacije. Sve kastomizacije koje vrši korisnik se čuvaju u planu kastomizacije (*eng. customization plan*) i predstavljene su kao modifikacije originalne funkcionalne specifikacije pre nego promene konkretnog korisničkog interfejsa. Na ovaj način je proces generisanja interfejsa proširen i uključuje preprocesuirajući korak u kojem se plan kastomizacije primenjuje na funkcionalnu specifikaciju. Odvajanje plana modifikacije od prezentacije interfejsa omogućava portabilnost intefejsa između različitih uređaja. Pored toga, planovi kastomizacije se mogu editovati i menjati od strane korisnika. Na ovaj način je podržana kolaboracija korisnika kod modifikacija korisničkog interfejsa.

Kao dodatak sistemu za modelovanje korisnika koriste se dva podsistema. Podsistem ARNAULD pronalazi naklonjest korisnika u pogledu prezentacije grafičkih korisničkih interfejsa. Ovo se svodi na paralelne prikaze fragmenata korisničkog interfejsa nakon čega se korisnici izjašanjavaju za onaj koji im više odgovara. Fragmenti su funkcionalno ekvivalentni, ali su prezentacije različite. Upiti se generišu automatski na osnovu rezultata upita prethodnih sesija za konkretnog korisnika. Podsistem ABILITY MODELER kreira modele motoričkih sposobnosti korisnika. Prediktivni model se kreira na osnovu merenja performansi korisnika pri vršenju četiri osnovna tipa zadatka: pokazivanje (*eng. pointing*), prevlačenje (*eng. dragging*), selekcija liste i višestruki kliktaji nad jednim objektom. Dakle, metrike za modelovanje korisnika su sklonosti korisnika i motoričke sposobnosti. U budućnosti ove metrike mogu obuhvatiti kogntivne osobine i pažnju.

Osnovna namena sistema nije da zameni korisnika dizajnera, već da pruži alternativu u dizajnu korisničkih interfejsa za korisnike sa specifičnim individualnim potrebama, uključujući i one sa posebnim potrebama. Pored toga, usled raznolikosti korisnika ne postoji dovoljan broj dizajnera koji će kreirati interfejse koji će odgovarati kontekstu svake osobe ponaosob. Opisani pristup na odgovarajući način rešava ovaj problem skalabilnosti korisničkih interfejsa.

2.4. *Jezici za opis korisničkih interfejsa*

Jezici za opis korisničkih interfejsa se mogu podeliti u dve grupe – deklarativni i imperativni. Deklarativni jezici govore *šta* treba prikazati na korisničkom interfejsu, dok imperativni jezici govore *kako* nešto treba prikazati i definišu ponašanje komponenti korisničkog interfejsa. U imperativne jezike spadaju konvencionalni programski jezici kao što su proceduralni, objektno-orijentisani i skript jezici. Korišćenje odgovarajuće klase jezika određeno je kontekstom izrade interfejsa, tj. namenom i okruženjem korišćenja interfejsa. U kontekstu Web okruženja, za izradu korisničkog interfejsa koriste se deklarativni jezici. Ponašanje interfejsa se donekle može definisati kombinacijom sa skript jezicima. U slučaju stonih platformi koriste se imperativni jezici. Tekući trend razvoja korisničkih interfejsa predviđa hibridno rešenje kod kojeg je izgled interfejsa definisan deklarativnim jezikom, dok je programska logika opisana korišćenjem imperativnog jezika. Na ovaj način se ide ka unifikaciji korišćenja korisničkog interfejsa u Web i stonom okruženju. Dve najpoznatije tehnologije koje prate ovaj pristup jesu Adobe FLEX AIR i Microsoft WPF (*eng. Windows Presentation Foundation*). Proteklih godina pojavio se veliki broj jezika za opis korisničkih interfejsa (*eng. User Interfaces Description Languages - UIDLs*). Većina tih jezika je deklarativna. Pojava ovih jezika bila je uslovljena sledećim potrebama [Paterno08]:

- Apstraktna definicija korisničkog interfejsa koja obezbeđuje njegovu portabilnost između različitih platformi.
- Dizajn jednog korisničkog interfejsa za različite uređaje i platforme.
- Ponovna upotrebljivost kreiranih interfejsa.
- Proširivost i adaptibilnost korisničkog interfejsa.
- Automatsko generisanje programskog kôda korisničkog interfejsa.

Novi jezici su se javljali uporedo sa okvirima za razvoj korisničkih interfejsa za veći broj platformi, konteksta upotrebe i profila korisnika. U nastavku će biti ukratko opisani jezici za opis korisničkih interfejsa koji su našli širu prihvaćenost, kako u komercijalnoj upotrebi, tako i u naučno-istraživačkoj zajednici.

2.4.1. Deklarativni jezici

U tekućem odeljku će biti opisani deklarativni jezici za opis korisničkih interfejsa koji su našli širu primenu u praksi. Na kraju će odeljka će biti opisana i dva predloga specifikacije samih jezika koji su

ujedno i radovi u specijalnom izdanju časopisa *ACM Transactions on Computer-Human Interaction* koji je bio posvećen ovom problemu.

2.4.1.1. XUL

XUL (*eng. XML User Interface Language*) je deklarativni jezik za opis korisničkih interfejsa. Jezik je kreiran za potrebe *Mozilla* proizvođača i koristi se u *Firefox* pretraživaču i *Thunderbird* klijentu elektronske pošte. Jezik je našao široku primenu u Web aplikacijama u kombinaciji sa Java programskim jezikom. Primer ove kombinacije predstavlja i okvir za razvoj korisničkih interfejsa za mobilne uređaje [Butter07]. Okvir razmatra korišćenje XUL jezika za opis korisničkog interfejsa, dok je logika aplikacije opisana u Java ME tehnologiji. Pored toga, razvijene su i odgovarajuće komponente za korišćenje u stonim aplikacijama.

2.4.1.2. XIML

XIML (*eng. eXtensible Interface Markup Language*) je jezik za prikaz i manipulaciju podacima koji su povezani sa interakcijom čoveka i računarima, tj. podacima interakcije [Puerta02]. U ove podatke spadaju zadaci korisnika, domenski objekti, elementi dijaloga i elementi prezentacije. Kombinacijom ovih podataka jezik formira čitave modele korisničkih interfejsa, kako apstraktne, tako i konkretne. Jezik se koristi za dizajn korisničkih interfejsa u alatima poput *UIFin* [Puerta09]. Alat koristi deklarativne modele za kreiranje korisničkih interfejsa. Ovi modeli su razdvojeni u dva sloja, sloj dizajna i izvršni sloj. Modeli dizajna su napisani na XIML jeziku. XIML sloj kreira modele korisničkih visokog nivoa apstrakcije i omogućava interoperabilnost alata za dizajn korisničkih interfejsa. Izvršni sloj je sastavljen od modela napisanih na komercijalnim XML jezicima za opis korisničkih interfejsa. U konkretnom slučaju to su MXML i XAML. Sa obzirom da su oba sloja napisana na XML jezicima, moguće je na definisati XSL transformacije između odgovarajućih modela. Na ovaj način je moguće ostvariti spregu se komercijalnim izvršnim platformama i razvojnim okruženjima kao što su Microsoft Visual Studio i Adobe Flex Builder.

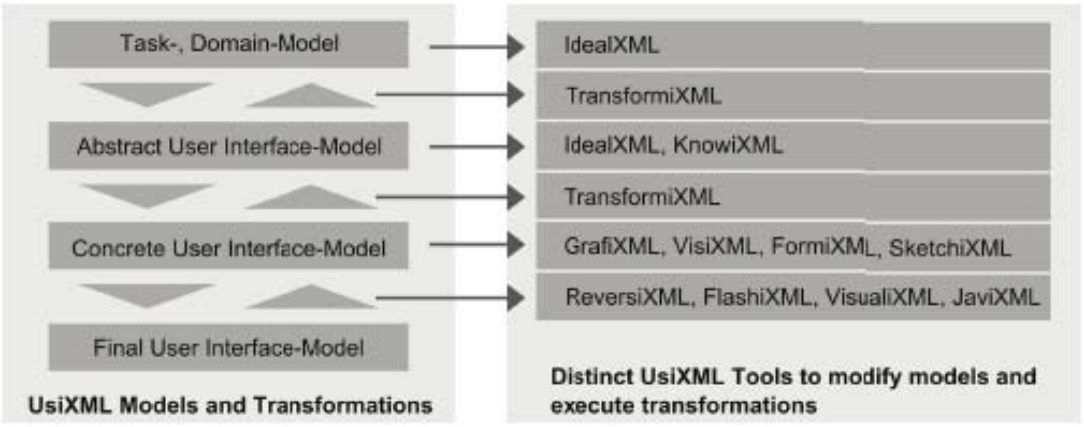
2.4.1.3. UIML

UIML (*eng. User Interface Markup Language*) [Abrams99] je jezik za opis korisničkih interfejsa koji nastoji da kreira platformski nezavisne korisničke interfejse. Realizacija platformske nezavisnosti zasnovana je na korišćenju datoteka sa stilovima (*eng. Cascading Style Sheets – CSS*). Specifikacija jezika predviđa tagove za povezivanje sa komponentama koje realizuju aplikativnu logiku, kao i programski kôd koji će se izvršavati po nastanku događaja. Od svog nastanka jezik je proširivan u cilju unapređenja platformske nezavisnosti [Helms08]. MONA projekat [Simon04] predstavlja generičku platformu za multimodalne servise za mobilne uređaje. Generički opisi korisničkih interfejsa se kreiraju korišćenjem UIML i nakon toga se transformišu u oblik pogodan za korišćenje na specifičnim platoformama kao što su WML, HTML, VoiceXML. Značajno proširenje UIML specifikacije predstavlja DISL (*eng. Dialog and Interface Specification Language*) [Schaefer06]. Cilj proširenja je da omogući kreiranje korisničkih interfejsa za veći broj različitih interfejsa i načina komunikacije. Da

bi se ovo postiglo UIML jezik je modifikovan tako što su (1) konkretni vidžeti zamenjeni generičkim, slično MONA projektu i (2) proširene UIML specifikacije ponašanja novim modelima koji obuhvataju svojstva kao što su varijable stanja i mehanizmi njihove promene. Za opisivanje korisničkih interfejsa na UIML jeziku postoji razvijen alat, UIML Development Toolkit. Alat integriše vizuelni dizajner za kreiranje korisničkih interfejsa i dostupan je kao plug-in komponenta Eclipse razvojnog okruženja. Pored toga, podržan je i od strane LiquidUI alata koji integriše skup konvertora za različite platforme kao što su Java, HTML, VoiceXML i C++.

2.4.1.4. UsiXML

USIXML (*eng. USer Interface eXtensible Markup Language*) [Limbourg04] je jezik za opis korisničkih interfejsa razvijen na Luven univerzitetu u Belgiji. Jezik služi opisu korisničkih interfejsa za veći broj različitih konteksta upotrebe. Sam jezik je dekomponovan u veći broj metamodela od kojih svaki opisuje zaseban aspekt korisničkog interfejsa u skladu sa CAMELEON referentnim okvirom [Calvary03]. Na ovaj način on omogućava specifikaciju različitih tipova modela korisničkih interfejsa, uključujući zadatke, domen, prezentaciju i modele konteksta upotrebe. Posebno je definisan i transformacioni model koji služi kao osnova za definisanje transformacija između različitih modela. Pored toga, jedan od ciljeva ovog jezika jeste da objedini i podrži karakteristike svih prethodno definisanih deklarativnih jezika za opis korisničkih interfejsa. Model konkretnog korisničkog interfejsa (u skladu sa CAMELEON okvirom iz sekcije 2.2.3) predstavlja hijerarhijsku organizaciju konkretnih objekata za interakciju (*eng. Concrete Interaction Object - CIO*), gde se CIO definiše kao entitet korisničkog interfejsa koji korisnik može opaziti. Iz svakog CIO se mogu izvesti podtipovi za specifične načine komunikacije. Grafički objekti za interakciju dalje mogu biti kontejnerski (komponente grupisanja) i individualni (primitivni vidžeti). Ovakav pristup modelovanja kontejnera kao podtipova konkretnih objekata za interakciju može biti problematičan, jer je njihova namena kompozicija elemenata pre nego modelovanje jednostavnih interakcija. Za realizaciju transformacija modela se koriste transformacije grafova [Stanciulescu08] što se na odgovarajući načina odražava na performanse. U cilju podrške celokupnog procesa razvoja korisničkih interfejsa u skladu sa CAMELEON metodologijom, razvijen je skup alata koji koriste UsiXML (slika 2.10)



Slika 2.10. *UsiXML skup alata* [Heinrich07].

2.4.1.5. *Komercijalne specifikacije deklarativnih jezika*

MXML (*eng. Macromedia eXtensible Markup Language*) [Conraets04] je deklarativni jezik za opis korisničkih korisničkih interfejsa u okviru Adobe Flex okvira za razvoj RIA (*eng. Rich Internet Applications*) aplikacija. Ove aplikacije karakteriše bogat multimedijalan sadržaj i visok stepen interakcije sa korisnikom. Logika u pozadini ovih aplikacija se definiše korišćenjem Action Script jezika. Pored toga, postoje dobro razrađeni mehanizmi integracije sa Java Web tehnologijama. Za potrebe ove tehnologije razvijen je i binarni protokol za razmenu podataka na aplikativnom nivou poznat kao AMF (*eng. Action Messsage Format*). Tehnologija predstavlja proširenje Adobe Flesh platforme za kreiranje multimedijalnih sadržaja, sa posebnim naglaskom na animacije. Tehnologija ima široku zajednicu korisnika i koristi se u velikom broju istraživačkih i komercijalnih projekata [TourDeFlex10]. Tehnologija je otvorenog kôda u smislu da se razvojno okruženje, back-end tehnologija i izvršno okruženje mogu dobiti besplatno.

XAML (*eng. Microsoft's eXtensible Markup Language*) predstavlja jezik za opis korisničkih interfejsa razvijen u Microsoft-u [Microsoft06]. Deo je WPF tehnologije. Predstavlja izlaznu notaciju GUI palete alatki u okviru Visual Studio razvojnog okruženja. U potpunosti se oslanja na Microsoft-ove razvojne tehnologije i izvršne platforme [Bishop06].

2.4.1.6. *Izvedene specifikacije deklarativnih jezika*

PUC (*eng. Personal Unversal Controler*) projekat predstavlja apstraktni uređaj za udaljenu kontrolu uređaja koji se mogu naći u domaćinstvu korisnika [Nichols09]. Ovo se postiže preko odgovarajućeg korisničkog interfejsa ovog uređaja koji omogućuje dvosmernu komunikaciju sa uređajima u okruženju. Ova komunikacija se obavlja tako što se najpre dobavlja opis funkcija uređaja napisan u posebno definisanom jeziku. Potom se automatski generiše odgovarajući interfejs. Nakon toga se posredstvom kreiranog interfejsa šalju kontrolni signali prema uređajima i dobijaju povratne informacije o promeni stanja uređaja. Da bi se opisala funkcionalnost uređaja koji se mogu naći u domaćinstvu razvijen je prototip deklarativnog jezika. Jezik je korišćen za opis preko trideset različitih uređaja koji se mogu naći u domaćinstvu, uključujući mobilne uređaje i skrivene računare kojima se upravlja glasovnim komandama. Na osnovu opisa uređaja automatski su generisani grafički i govorni korisnički uređaji na handheld računarima, mobilnim telefonima i stonim računarima. Detalji specifikacije se mogu naći u [Nichols09].

Jedan od pravaca evolucije softvera jeste pojava i razvoj servisno-orijentisanih arhitektura koje integrišu interaktivna okruženja. Ova okruženja karakteriše širok skup interaktivnih uređaja na kojima se izvršavaju različite aplikacije i koji biva enkapsuliran u skup Web servisa. Pojava Web servisa za interaktivne aplikacije uslovljava razvoj odgovarajućih interfejsa preko kojih im se pristupa, tj. SFE (*eng. Service Front-End*). MARIA (*eng. Modelbased lAnguage foR Interactive Applications*) je deklarativni jezik za specifikaciju korisničkih interfejsa aplikacija koje se koriste u servisno-orijentisanoj arhitekturi [Paterno09]. Jezik je namenjen za razvoj višeplatformskih korisničkih interfejsa koji pri tome integrišu anotacije Web servisa. Jezik je zasnovan na modelima i ima

modularnu strukturu. To konkretno znači postojanje jednog jezika (metamodela) za apstraktni opis korisničkog interfejsa i većeg broja platformski specifičnih jezika koji predstavljaju specijalizacije apstraktnog jezika. Apstraktni opis je nezavisan od resursa interakcije, dok je konkretni opis određen načinom komunikacije, ali je nezavisan od tehnologije implementacije. Na taj način jezik sadrži apstraktne/konkretne modele podataka i događaja, dinamički skup elemenata korisničkog interfejsa i podršku za AJAX skriptove. Apstraktni i konkretni modeli korisničkih interfejsa su opisani preko XML šema. Tehnologija koja se koristi za transformacije je XSLT. Za kreiranje transformacija dizajnerima je na raspolaganju vizuelni alat. U tom pogledu, transformacije nisu ugrađene u alat, već se se mogu kreirati i prilagođavati definisanjem pravila preslikavanja, na primer apstraktnih u konkretne elemente ili konkretnih elemenata u vidžete specifične tehnologije implementacije.

2.4.2. Imperativni jezici

U ovom odeljku će biti opisane karakteristike imperativnih jezika koji su našli širu prihvaćenost u kreiranju korisničkih interfejsa. Ove karakteristike će biti opisane u kontekstu alata koji koriste ove jezike, a koji se mogu klasifikovati kao:

- Alati za rad sa prozorima (*eng. window managers and toolkits*),
- Jezici za obradu događaja (*eng. event languages*),
- Interaktivni grafički alati,
- Komponentni sistemi,
- Skript jezici,
- Objektno-orijentisano programiranje

2.4.2.1. Alati za rad sa prozorima

U svojoj doktorskoj tezi iz 1969. godine, Alan Kay je uveo ideju preklapajućih prozora, koji su se prvi put komercijalno koristili 1974. godine u *Smalltalk* sistemu realizovanom od strane *Xerox PARC* laboratorije. Mnoga druga istraživanja i komercijalni sistemi su preuzeli ideju iz ovih ranih radova, na primer, *Xerox Start*, *Apple Macintosh* i *Microsoft Windows*. Jedan od razloga za uspeh preklapajućih prozora jeste i njihova mogućnost da, poput operativnih sistema, pomažu u upravljanju sa ograničenim resursima. Pri tome, ograničenja uključuju ne samo ograničenja računarskih resursa, poput broja piksela i broja boja, već i čovekova percepcijska i saznajna ograničenja, kao što su ograničeno vidno polje i pažnja korisnika. Kontrolom skupa preklapajućih prozora od strane korisnika, korisnički interfejsi su mogli da se prilagode fokusu i pažnji korisnika potrebnoj da bi se uspešno realizovao neki zadatak na računaru. Pored toga, delimično preklapanje prozora predstavlja pomoć za čovekovo pamćenje u slučaju kada nije moguće odjednom prikazati sve potrebne objekte na ekranu.

Prvi alati za rad sa prozorima su bili u obliku programskih biblioteka koje su pružale primitive prilagođene konceptima prozora. Biblioteke za rad sa prozorima su obično sastavni deo operativnih

sistema. One obezbeđuju mehanizme za crtanje i osvežavanje prikaza, kao i mehanizme za prihvatanje unosa od strane korisnika. Iako ove biblioteke obezbeđuju primitive na visokom nivou apstrakcije, programiranje na ovom nivou je detaljno, naporno i zahteva mnogo vremena. Pored toga, ako svaki programer razvija komponente od samog početka, teško je obezbediti konzistentan i standardizovan korisnički interfejs.

Da bi se prevazišli neki od navedenih problema, razvijeni su alati na višem nivou apstrakcije od biblioteka. Ovi alati tipično obezbeđuju biblioteku već razvijenih interaktivnih komponenti, kao i arhitektonski skelet za integraciju ovih komponenti (*eng. architectural framework)*. Kao što je prvi put demonstrirano od strane *Apple Macintosh Toolbox* alata, pored pojednostavljenja programiranja, olakšano je i kreiranje konzistentnog stila korisničkih interfejsa. Još jedan od razloga za uspeh ovih alata jeste rad na relativno niskom nivou apstrakcije, sa primitivama koje su dobro shvaćene od većeg broja programera, tako da su konzistentnost i ponovna upotreba već razvijenog koda olakšani.

2.4.2.2. Jezici za obradu događaja

U sistemima koji rade sa događajima, pojava svakog događaja od interesa za sistem, poput korisnikovog rada sa ulaznim uređajem, se smešta u instancu odgovarajuće strukture podataka za opis tog događaja. Ova struktura podataka se često skraćeno naziva *događaj*. Događaji se nakon detekcije šalju rutinama za obradu događaja (*eng. event handlers*), koje sadrže opis reakcije na ulazni događaj.

Jedan od osnovnih razloga za uspeh jezika za obradu događaja jeste njihovo dobro preslikavanje na direktnu manipulaciju sa grafičkim objektima. Pri direktnoj manipulaciju sa grafičkim objektima, operativni sistem generiše niz događaja za svaku korisnikovu akciju nad mišem ili tastaturom, te ih šalje aplikaciji na obradu. Važno je uočiti da je na ovaj način moguće razdvojiti logiku programa od detalja komunikacije između čoveka i računara, jer program radi sa događajima kao apstrakcijama, a ne sa konkretnim uređajima. Više različitih uređaja mogu da generišu iste događaje, omogućujući širu upotrebljivost aplikacija. Međutim, jezici za obradu događaja ne mogu da se uspešno koriste za nove stilove interakcije, poput govora ili prepoznavanja pokreta i izraza lica korisnika.

2.4.2.3. Interaktivni grafički alati

Interaktivni grafički alati su uveli grafički jezik za kreiranje korisničkih interfejsa. Ovi alati omogućuju kreiranje elemenata korisničkih interfejsa, poput prozora ili dijaloga, vizuelnim "slaganjem" i povezivanjem interaktivnih komponenti na ekranu. Jedan od bitnih razloga za uspeh interaktivnih grafičkih alata bio je upotreba grafičkog jezika za opis grafičkih koncepata. Prenošenjem nekih od koncepata iz programerskog sveta u oblast interaktivnog grafičkog programiranja, korisnički interfejsi su mogli da se razvijaju i od strane ljudi koji nisu programeri. Ovo je omogućilo ekspertima iz raznih domena da izrađuju prototipska rešenja i interfejse prilagođene njihovim potrebama, a takođe je omogućilo i aktivno učešće grafičkih dizajnera u procesu izrade korisničkih interfejsa. Interaktivni grafički alati su doneli dobit i programerima, jer je vreme razvoja značajno smanjeno. Jedna od osnovnih prednosti ovih sistema je što "jednostavne stvari mogu biti urađene na jednostavan način" [Myersoo].

2.4.2.4. *Komponentni sistemi*

Komponentni sistemi se zasnivaju na ideji izrade sistema kombinovanjem nezavisno napisanih i prevedenih komponenti. Ova ideja je komercijalno realizovana u sistemima kao što su *Sun JavaBeans*, *Microsoft OLE* i *ActiveX*, i *Apple OpenDoc*. Jedan od osnovnih razloga za uspeh komponentnih modela jeste mogućnost modularnog razvoja programa i korišćenja gotovih komponenti.

2.4.2.5. *Skript jezici*

Prvi alati za razvoj korisničkih interfejsa su bili razvijeni u interpreterskim programskim jezicima. Interpreterski jezici omogućili su projektantima da brzo naprave prototip rešenja, te da brzo prave izmene i primete efekte tih izmena. Sa sve širom upotrebom jezika C i C++, većina programera je prešla na jezike koji su se prevodili, tako da su se izgubile neke od osnovnih prednosti interpreterskih jezika. Zato su istraživači ispitivali načine da se vrate neke od prednosti korišćenja interpreterskih jezika, što je rezultiralo u pojavi jezika kao što su *Python* i *Perl*. Povezivanje skript jezika sa komponentama i interaktivnim grafičkim alatima se potvrdilo kao veoma uspešna kombinacija. Korišćenje grafičkih alata za razmeštanje ovih komponenti, a zatim korišćenje nekog skript jezika za povezivanje ovih komponenti, omogućilo je da ljudi koji se na bave profesionalnim razvojem korisničkih interfejsa mogu da kreiraju složene i funkcionalne aplikacije.

2.4.2.6. *Objektno-orijentisano programiranje*

Razvoj objektno orijentisanih jezika i alata korisničkih interfejsa je dosta povezan i obostrano uticajan. Na primer, *Smalltalk* je razvijen sa namerom da olakša izradu interaktivnih grafičkih programa. C++ je postao popularan kada se pojavila potreba za programiranjem grafičkih korisničkih interfejsa u operativnom sistemu Windows. Objektno orijentisano programiranje se prirodno uklapa u razvoj grafičkih korisničkih interfejsa. Komponente korisničkih interfejsa, poput dugmadi i menija, su vidljivi objekti koji imaju svoje stanje, kao i operacije preko kojih se ovo stanje može menjati, što odgovara definiciji objekata u objektno orijentisanim sistemima.

2.5. *Alati za razvoj korisničkih interfejsa vođeni modelima*

Pojava i raznovrsnost novih tehnologija usložnjava razvoj interaktivnih sistema. Ovo za posledicu ima pojavu modelski zasnovanih pristupa koji opisuju koncepte interakcije čoveka i računara na visokom nivou apstrakcije. Sa druge strane, u oblasti softverskog inženjerstva uporedo se javlja MDE (*eng. Model Driven Engineering*) koji treba da obezbedi tehnike i alate za automatizaciju rada sa modelima vezanim za određeni domen. MDE podrazumeva korišćenje metamodela, modela i transformacije između modela kako bi se unapredila produktivnost razvoja. U poglavlju će biti opisani alati za razvoj korisničkih interfejsa vođeni modelima. Najpre će biti opisani sistemi za upravljanje korisničkih interfejsa koji su prvi uveli korišćenje modela za opisivanje korisničkih interfejsa. Nakon toga će biti ukratko biti opisani osnovni principi MDE, kao i prateći alati za razvoj korisničkih interfejsa. Na kraju je dat uporedni pregled postojećih alata.

2.5.1. Sistemi za upravljanje korisničkim interfejsima

Sistemi za upravljanje korisničkim interfejsima (*eng. User Interface Management Systems*) počeli su da se razvijaju ranih osamdesetih godina dvadesetog veka. Osnovna namena ovih sistema je bila automatizacija dizajna korisničkih interfejsa, tako da programeri koji nisu vični dizajnu mogu kreirati aplikacije sa korisničkim interfejsima prihvatljivog kvaliteta. Pored toga, generisani inerfejsi su mogli biti modifikovani od strane dizajnera kako bi se unapredila njihova upotrebljivost. Interfejsi kreirani na ovaj način sve češće su dobijali prefiks modelski zasnovani (*eng. model-based*) zbog postojanja odgovarajućih modela u pozadini. Rani modelski-zasnovani sistemi su imali nekoliko važnih nedostataka. Pre svega, kreiranje modela za generisanje korisničkih interfejsa je predstavljalo veoma apstraktan i vremenski zahtevan proces. Tadašnji jezici za modelovanje su imali strme krive učenja i zbog toga je često više vremena bilo potrebno za kreiranje modela nego za ručno kreiranje korisničkog interfejsa. I na kraju, sam postupak generisanja interfejsa iz modela je bio vrlo složen proces koji je često rezultovao interfejsima niskog kvaliteta [Myers00].

Značaj modelski zasnovanih pristupa prostiče iz realnih potreba:

- Korisnički interfejsi velikih skala sastavljeni od različitih tehhologija su složeni u pogledu implementacije i kasnije modifikacije. Detaljni modeli korisničkog interfejsa mogu pomoći u organizaciji i automatizaciji procesa implementacije. Pored toga, modeli mogu biti od koristi pri modifikacijama budućih verzija interfejsa.

- Potreba za interfejsima nezavisnim od uređaja na kojima se prikazuju predstavlja važan podsticaj istraživanju modelski zasnovanih interfejsa. U tom kontekstu, posebna pažnja je posvećena izradi alata za automatsko generisanje korisničkih interfejsa.

Kao što je prethodno rečeno, oslonac na modele koji opisuju relevantne aspekte korisničkog interfejsa treba da olakša i unapredi rad dizajnera i programera. Kada govorimo o modelski zasnovanim pristupima razvoju korisničkih interfejsa, možemo uočiti nekoliko generacija ovih sistema. Prva generacija je bila zasnovana na izvođenju apstrakcija za grafičke korisničke interfejse. U to vreme, akcenat je bio na identifikovanju relevantnih aspekata vezanih za ovaj način komunikacije [Foley94]. Potom su sistemi evoluirali u drugu generaciju kod koje je akcenat na definisanju semantike interakcije na visokom nivou. Ovo je uglavnom bilo podržano korišćenjem modela zadataka podržanih od strane odgovarajućih alata, drugim rečima, opisom aktivnosti koje korisnik izvršava kada interaguje sa sistemom [Paterno00]. Nakon toga, kao posledica pojave novih platformi i tehnologija interakcije, a naročito mobilnih, razvoj modelski zasnovanih sistema je bio usmeren na obezbeđivanje svojstva migracije korisničkog interfejsa između različitih uređaja, a da se pri tome očuva njegova upotrebljivost. Kao što je istaknuto u [Myers00], pojava novih, raznovrsnih platformi interakcije dovela je do povećanja interesovanja za modelski zasnovane pristupe kako bi dizajneri definisali potrebne ulaze i izlaze aplikacija, proizvođači opisali ulazne i izlazne karakteristike uređaja i krajnji korisnici izrazili svoje potrebe. Pored toga, ovi sistemi moraju i dalje dozvoljavati kontrolu nad finalnim interfejsom kako bi on bio što kvalitetniji. Veliki broj istraživanja je bio posvećen višeplatformskim korisničkim interfejsima identifikacijom relevantnih informacija koje sadrže

odgovarajući modeli (i jezici) [Helms08], [Limbourg04], [Mori04]. Modelski zasnovani pristupi razvoja korisničkih interfejsa su pokrenuli i inicijative za definisanje međunarodnih standarda u oblasti[1] ili za njihovo usvajanje od strane industrije[2]. Danas se mogu identifikovati trendovi razvoja vezani za interaktivne aplikacije koji vode ka četvrtoj generaciji ovih sistema. U ove trendove pre svega spadaju: pristup aplikacijama koje mogu biti razmeštene na bilo kojoj lokaciji i na različitim tipovima uređaja; korišćenje tehnologija lokalizacije (kao što su GPS, RFID); korišćenje različitih načina komunikacije sa korisnikom; dizajn orijentisan ka korisniku [Paterno09].

2.5.2. Koncepti inženjerstva vođenog modelima

Radi boljeg i obuhvatnijeg razumevanja, koncepti inženjerstva vođenog modelima su podeljeni u dve grupe:

- Modeli i metamodeli,
- Transformacije između modela.

U nastavku je dat opis svake grupe ponaosob.

2.5.2.1. Modeli i metamodeli

MDE je disciplina softverskog inženjerstva kod koje su modeli centralni deo razvoja, dok je MDA (*eng. Model-Driven Architecture*) predlog softverske arhitekture metamodelovanja zasnovana na MDE principima. Model predstavlja opis sistema koji se posmatra i koji je dat kao skup izjava o sistemu [Seidewitz03]. Modeli se pišu korišćenjem jezika za modelovanje (npr. UML), dok su jezici za modelovaje definisani preko odgovarajućih metamodela. Drugim rečima, metamodel je model jezika za modelovanje. Relacije između modela i metamodela su opisane kroz arhitekture metamodelovanja, gde arhitektura metamodelovanja obično sadrži četiri sloja (npr. MDA) [Kleppe03]:

- M0 sloj ili sloj instanci u kojem se nalaze objekti kao instance odgovarajućih modela;
- M1 sloj ili sloj modela u kojem su smešteni modeli napisani na jeziku za modelovanje;
- M2 sloj ili sloj metamodela sadrži modele jezika za modelovanje (tj. metamodele). Ovi modeli su definisani jezicima za metamodelovanje kao što je OMG MOF[3] (*eng. Meta Object Facility*) ili Eclipse Ecore[4].
- M3 sloj ili sloj metametamodela u kojem je definisan jezik za metamodelovanje (MOF, ECore). MOF predstavlja OMG standard koji definiše jezik za definisanje jezika za modelovanje. To je jezik na kojem se pišu UML metamodeli. ECore praktično predstavlja varijantu MOF za *Eclipse Modeling Framework* platformu.

[1] W3C Radna grupa za modelski zasnovane korisničke interfejse, http://www.w3.org/2005/Incubator/model-based-ui/
[2] Radna grupa NESSI NEXOF-RA IP, http://www.nexof-ra.eu/
[3] Meta Object Facility (MOF) Core, v2.0, http://www.omg.org/cgi-bin/doc?formal/2006-01-01
[4] Eclipse Modeling Framework, http://www.eclipse.org/emf

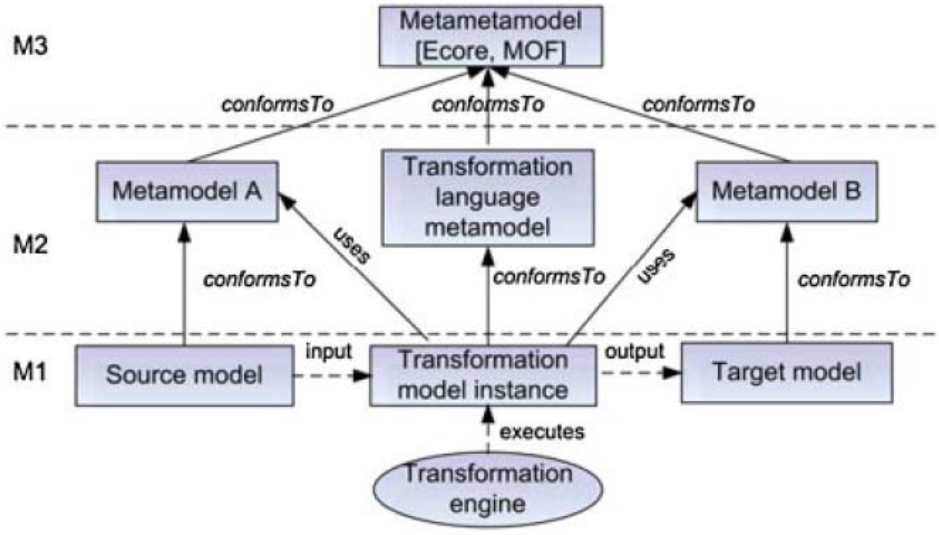
Relacije između različitih MDE slojeva su tipa instanca (*eng. instance-of ili conformant-to*). To konkretno znači da je objekat instanca modela, model instanca metamodela, i metamodel instanca metametamodela. Obrazloženje za definisanje jednog jezika na M3 sloju je postojanje jedinstvene gramatika (prostora) za definisanje različitih jezika za modelovanje na sloju M2. Na ovaj način, različiti jezici za modelovanje mogu biti obrađivani na isti način, na primer, korišćenjem istog API-ja ili jezika transformacije. Najzastupljeniji jezik metamodelovanja, MOF, zasnovan je na UML sintaksi, tj. predstavlja podskup UML dijagrama klasa.

U kontekstu arhitektura metamodelovanja postoje i pravila ispravnosti modela (*eng. well-formedness rules*). Ova pravila se definišu na nivou metamodela, koristeći jezik za metamodelovanje (MOF) i jezik za specifikaciju ograničenja (OCL – *Object Constraint Language*). Ograničenja se definišu nad metamodelima i definišu skup validnih modela.

Pored toga, za svaki MOF metamodel i model se može generisati odgovarajuća deklarativna reprezentacija, tj. XML šema. Ovaj standard je poznat pod nazivom XMI[5] (*eng. XML Metadata Interchange*) i omogućava razmenu MOF modela i metamodela između različitih repozitorijuma MOF modela.

2.5.2.2. Transformacije između modela

Dok su modeli i metamodeli koncepti prvog reda, transformacije između modela predstavljaju centralnu aktivnost u MDE [Sendall03]. Transformacija podrazumeva konverziju izvornog modela (ili skupa ulaznih modela) koji je izveden iz jednog metamodela u odredišni model (ili skup modela) koji je izveden iz drugog metamodela [Bezivin06]. Naravno, pod transformacijama se podrazumevaju i one koje se vrše na niskom nivou apstrakcije, tj. generisanje programskog kôda iz modela. U kontekstu MDA, mesto i uloga transformacija je prikazana na slici 2.11.



Slika 2.11. *Mesto i uloga transformacija u MDE [Bezivin06].*

[5] OMG Meta Object Facility (MOF) 2.0 XMI Mapping Specification, v2.1. http://www.omg.org/cgi-bin/doc?formal/2005-09-01

Konverzija se vrši definisanjem pravila preslikavanja između elemenata izvornog i odredišnog modela. Sama transformacija je model koji je izveden iz odgovarajućeg metamodela. Postoji veći broj jezika za transformacije. Najzastupljeniji su deklarativni zasnovani na XSLT[6]. Zatim su tu jezici izvedeni iz programskih jezika kao što je JMI[7]. I na kraju, jezici zasnovani na OMG QVT[8] standardu među kojima je najzastupljeniji ATL[9] (*eng. Atlas Transformation Language*).

2.5.3. Postojeći modelski zasnovani pristupi u dizajnu korisničkih interfejsa

Postojeći MDE pristupi u dizajnu korisničkog interfejsa će najpre biti uopšteno opisani. Ovi opisi će najpre biti dati u kontekstu postojećih modela i metamodela, a nakon toga će u kratkim crtama biti prikazani postojeći oblici transformacija.

MDE pristupi su najvećim delom zasnovani na UML metamodelima i MOF jeziku za definisanje metamodela. UML predstavlja široko prihvaćen industrijski standard, koristi se u velikom broju alata, uči na većini studijskih programa univerziteta i rasprostranjen je kako u komercijalnim oblastima, tako i u istraživačkoj zajedinici softverskog inženjerstva. U oblasti interakcije čoveka i računara razvijene su specifične notacije za opis korisničkih interfejsa kao što su modeli zadataka. Ove notacije su razvijene pre nastanka UML jezika, pa sa njegovom pojavom i prihvatanjem uporedo dolazi do integracije postojećih koncepata u UML. Nekoliko UML metamodela je kreirano u oblasti kontekstno-osetljivih korisničkih interfejsa [Sotte05]. Korišćenje UML profila, kao mehanizama proširenja UML metamodela, takođe je našlo primenu u dizajnu korisničkih interfejsa [da Silva03]. U pogledu poštovanja OMG standarda razvijane su metodologije razvoja korisničkih interfejsa, kao što su UML Wisdom arhitektura [Nunes00a] ili skup UsiXML alata [Heinrich07]. CAMELEON referentni okvir [Calvary03] predstavlja konsenzus o tipovima modela korisničkih interfejsa i nivoima apstrakcije na kojima se oni koriste. U novije vreme, javljaju se pristupi zasnovani na izvršnim modelima (*eng. executable models*). Ovi pristupi su zasnovani na ostvarivanju direktne povratne sprege između modela višeg nivoa apstrakcije sa jedne strane, i eksternih sistema i logike i strukture izvršavanja sa druge strane. Cilj je obezbediti adaptaciju korisničkog interfejsa interaktivne aplikacije u vreme izvršavanja [Blumendorf10]. Međutim, ponuđena rešenja koja ilustruju pristup su platformski specifična. To konkretno znači da su realizovana u EMF (*eng. Eclipse Modeling Framework*) okviru pri čemu se u velikoj meri oslanjaju na ugrađenu mogućnost prikaza strukture i ponašanja metamodela mehanizmima Java jezika kao što su refleksija, obaveštavanje o događajima (*eng.event observer*) i predstavnici (*eng. proxy*). Pored toga, neka od njih predviđaju kreiranje domenski specifičnih jezika i specijalizovanih alata za rad sa ovim jezicima [Falb09]. U ovim rešenjima nema jasne granice između nivoa apstrakcije i semantike modela. Postojanje specifičnih, nestandardizovanih jezika i alata otežava njihovu integraciju u postojeće, šire prihvaćene procese i tehnologije razvoja.

---

[6] W3C. World Wide Web Consortium, http://www.w3.org/TR/2007/REC-xslt20-20070123/
[7] JMI. Java Metadata Interface, http://java.sun.com/products/jmi/
[8] Query/View/Transformation. OMG Specification, http://www.omg.org/docs/ptc/05-11-01.pdf
[9] ATLAS Transformation Language, http://www.eclipse.org/m2m/atl/

Za razliku od softverskog inženjerstva, u oblasti interakcije čoveka i računara ne postoji opšti konzenzus oko korišćenja modela za opisivanje koncepata interakcije. Pored toga, za opis određene oblasti, kao što su modeli zadataka postoji veći broj različitih notacija. Na ovaj način postoji veći broj metamodela za modele zadataka što otežava njihovu integraciju i svodi na upotrebu u specifičnim domenima. Integracija SE i HCI modela podrazumeva korišćenje jednog metajezika. Ovo je naročito značajno u pogledu povezivanja logike korisničkog interfejsa sa poslovnom logikom sistema. Kako SE i MDE zajednica koriste UML jezik za modelovanje, od značaja je da i HCI zajednica prihvati ovu praksu.

2.5.4. Uporedni pregled alata za razvoj korisničkih interfejsa vođen modelima

U ovom odeljku će biti opisani postojeći alati za razvoj korisničkih interfejsa vođen modelima koji su našli značajniju primenu u praksi (tabela 2.1). Obuhvaćeni alati su komercijalni i istraživački. Pored toga, realizovani su kao samostalni alati lili *plug-in* komponente. Pri izboru alata nismo se ograničili na one koje koriste UML jezik za opisivanje modela. Uporedna analiza alata je izvršena na osnovu ključnih MDE koncepata: modela i metamodela, transformacija između modela i drugih funkcionalnosti.

Tabela 2.1. *Pregled MDE alata korišćenih za razvoj korisničkih interfejsa.*

Alat	Opis
ACCELEO GPL Open source	MDA Plug-in za Eclipse platformu zasnovan na šablonima. http://www.acceleo.org/pages/accueil/fr
AndroMDA Open source	Proširivi okvir generatora. Transformacije UML modela u komponente J2EE, Spring, .NET. http://galaxy.andromda.org/index.php?option=com_frontpage &Itemid=48
ADT Open source	ATL Development Tools – Eclipse Plug-in alat za ATL transformacije. http://www.eclipse.org/m2m/atl/
AtoM3	Alat za metamodelovanje kompleksnih sistema. http://atom3.cs.mcgill.ca/index_html
DSL Tools (Microsoft Visual Studio SDK)	Omogućava kreiranje grafičkih dizajnera i generisanje izvornog kôda koristeći domenski specifične notacije za dijagrame. http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx
Kermeta	Jezik za metamodelovanje koji opisuje strukturu i ponašanje modela. http://www.kermeta.org/
Merlin Open source	MDE alat koji omogućava transformacije modela i generisanje kôda. http://merlingenerator.sourceforge.net/merlin/index.php
MDA Workbench Open source	MDA alat implementiran kao Eclipse plug-in. http://sourceforge.net/projects/mda-workbench
MOFLON Open source	Okvir za metamodelovanje kao plug-in komponenta Fujaba alata za transformacije grafova http://www.moflon.org/
OptimalJ Professional Edition	Generator J2EE aplikacija korišćenjem obrazaca transformacija poslovnih modela radne aplikacije. http://www.compuware.com/ products/optimalj/
QVT Partners BSD like license	Alati zasnovani na QVT za transformacije između modela i generator kôda. http://qvtp.org/downloads/qvtp-eclipse/
Smart QVT Open source	Alat za transformacije modela zasnovan na QVT. http://smartqvt.elibel.tm.fr/
UMLX Open source	Eksperimentalna sintaksa jezika za transformacije http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/UMLX/

2.5.4.1. *Alati u pogledu notacija za opis modela i metamodela*

U pogledu modela i metamodela upoređivali smo notacije za njihov opis. Konkretno, da li su opisani tekstualno ili grafički. Pored toga, razmatrana je mogućnost dodavanja ograničenja modelima i metamodelima. Razmatranje ograničenja je svedeno na upotrebu OCL jezika.

Tabela 2.2. *MDE alati u pogledu notacija za opis modela i metamodela.*

Alat	Notacija (metamodel)		Notacija (model)	
	Grafička (G) ili Tekstualna (T)	Ograničenja	Grafička (G) ili Tekstualna (T)	Ograničenja
ACCELEO	G,T	OCL	G,T	OCL
AndroMDA	T	OCL	G,T	OCL
ADT	T	OCL	T	OCL
AtoM3	G	-	G	-
DSL Tools	G,T	-	G,T	-
Kermeta	G,T	OCL	G,T	OCL
Merlin	G,T	OCL	G,T	OCL
MDA Workbench	G,T	OCL	G,T	OCL
MOFLON	G,T	OCL	G,T	OCL
OptimalJ	G	OCL	G	OCL
QVT Partners	G,T	OCL	T	OCL
Smart QVT	T	OCL	T	OCL
UMLX	G,T	OCL	G,T	OCL

Korišćenje alata sa grafičkom notacijom za modele i metamodele se preporučuje zbog lakšeg rada i prihvatljivosti za širi spektar korisnika koji se bave dizajnom korisničkog interfejsa.

2.5.4.2. *Alati u pogledu transformacija između modela*

Iako postoji standard za specifikaciju transformacija (QVT), ne postoji standardizovani jezik u smislu njegove implementacije. Sa jedne strane, veliki broj MDE alata podržava QVT što bi trebalo da garantuje interoperabilnost alata. Međutim, u praksi se razlikuju implementacije standarda i samim tim kompatibilnost alata nije garantovana. Najčešće korišćeni jezici za transformacije u dizajnu korisničkih interfejsa su ATL i XSLT [Medina07]. Korišćenjem oba jezika mogu se dobiti tekstualni ili deklarativni modeli. Pored toga, ATL ima široku primenu u domenu softverskog inženjerstva [Milanovic09]. Detaljniji pregled jezika za transformacije biće dat u sledećem potpoglavlju.

Tabela 2.3. *MDE alati u pogledu transformacija između modela.*

Alat	Transformacija			
	Jezik	Grafički (G) ili Tekstualni (T)	Generisani model	
			XMI	Tekst
ACCELEO	QVT, JMI	T	-	+
AndroMDA	ATL, MofScript	T	+	+
ADT	ATL	T	+	+
AtoM3	Python	G	-	+
DSL	XML notacija	T	+	+
Kermeta	QVT	T	-	+
Merlin	QVT, JET	T	-	+
MDA Workbench	QVT	T	-	+

MOFLON	JMI	G	-	+
OptimalJ	QVT	T	-	+
QVT Partners	QVT	T	-	+
Smart QVT	QVT	T	+	+
UMLX	XSLT, QVT	T	+	+

2.5.4.3. Alati u pogledu ostalih kriterijuma

Većina alata je zasnovana na MOF specifikaciji. Postoji nekoliko impementacija MOF standarda, od kojih je u praksi najzastupljeniji ECore. Veliki broj alata integriše biblioteke predefinisanih modela i metamodela zasnovanih na ECore. Zbog toga izbor ECore alata u velikoj meri obezbeđuje razvoj i razmenu ponovno upotrebljivih modela i metamodela. U pogledu interoperabilnosti Eclipse obezbeđuje *de facto* metode za skladištenje i restauraciju XMI modela. Zbog toga je veliki broj alata zasnovan na Eclipse platformi i može komunicirati sa drugim srodnim alatima.

Tabela 2.4. *Pregled MDE alata u pogledu ostalih kriterijuma.*

Alat	Repozitorijum			Interoperabilnost sa drigim alatima
	Metamodelovanje	Transformacije	Ograničenja	
ACCELEO	DSL, MDR, ECORE	-	XMI	Eclipse, Netbeans
AndroMDA	MOF	-	XMI	Eclipse
ADT	DSL, KM3, MDR, ECORE	Tekst (ATL)	XMI	Eclipse, Netbeans
AtoM3	Specifična grafička notacija			-
DSL	DSL specifična notacija	XML/XMI	-	Eclipse
Kermeta	ECORE	Tekst (QVT)	XMI	Eclipse
Merlin	ECORE	Tekst (QVT)	XMI	Eclipse
MDA Workbench	ECORE	XMI	XMI	Eclipse
MOFLON	ECORE	-	XMI	Eclipse
OptimalJ	CWM, ECORE	XMI	XMI	Eclipse
QVT Partners	ECORE	Tekst (QVT)	XMI	Eclipse
Smart QVT	ECORE	Tekst (QVT)	XMI	Eclipse
UMLX	ECORE	XMI, XSLT	XMI, XSLT	Eclipse

2.6. *Transformacije u alatima vođenim modelima*

U MDE razvoju korisničkih interfejsa obično veći broj modela opisuje različite aspekte korisničkog interfejsa. Relacije između ovih modela se uspostavljaju kroz transformacije. Opsežna taksonomija MDA transformacija je data u [Czarnecki06]. U kontekstu korisničkih interfejsa, različiti transformacioni alati se koriste za prelaske između i u okviru različitih slojeva apstrakcije. Raznovrsnost semantike modela, formata njihove prezentacije i alata uslovila je pojavu različitih oblika transformacija u praksi [Schaefer07]. Neke rade direktno nad modelima, a neke nad njihovim izvedenim formatima. Jedne su integrisane u modele, dok se druge primenjuju eksterno. Jedne se opet mogu editovati i modifikovati, dok su druge ugrađene u alate i ne može im se pristupiti. Zbog toga će u ovom potpoglavlju biti opisani i analizirani oblici transformacija koji su našli primenu u modelski zasnovanom dizajnu korisničkih interfejsa.

2.6.1. Oblici transformacija u modelski zasnovanom dizajnu korisničkih interfejsa

Pre uporednog pregleda izabranih jezika transformacija, ukratko ćemo opisati svaki od njih, kao i način na koji su korišćeni u transformacijama modela korisničkih interfejsa.

2.6.1.1. Transformacije grafova

GT (*eng. Graph Transformations*) predstavalja formalni i deklarativni pristup za transformacije modela koji imaju strukturu usmerenog grafa [Czarnecki03]. Od jezika koji koriste ovu vrstu transformacije u dizajnu korisničkih interfejsa najpoznatiji je UsiXML. Modeli napisani na UsiXML jeziku imaju strukturu grafa tako da se mapiranja između modela svode na transformacije grafova koje se sastoje od skupa pravila transformacija [Limbourg04], [Stanciulescu08]. Svako pravilo se sastoji od šablona leve strane (*eng. Left Hand Side - LHS*), uslova (*eng. Negative Application Condition - NAC*) i šablona desne strane (*eng. Right Hand Side -RHS*). Postupak transformacije se svodi na pronalaženje šablona desne strane u izvornom modelu i njegovom zamenom šablonom leve strane, pri čemu se razmatra i specificirani uslov. Ovaj pristup zahteva modele koji imaju strukturu grafa. U tom pogledu, on ne omogućava generisanje finalnih korisničkih interfejsa koji nemaju strukturu grafa. Pored toga, primena transformacija se na odgovarajući način odražava na performanse.

2.6.1.2. ATL

ATL (*eng. Atlas Transformation Language*) predstavlja hibridni jezik za pisanje transformacija [Jouault05]. U tom pogledu, korisnik ima na raspolaganju deklarativnu sintaksu čiju semantiku može proširivati imperativnim blokovima. Deklarativni aspekt se ogleda definisanju pravila preslikavanja kod kojih je izvorni šablon opisan skupom izvornih tipova i OCL izrazima koji mogu definisati ograničenja nad njima. Odredišni šablon se kreira slično, specifikacijom skupa odredišnih tipova na osnovu odgovarajućeg metamodela. Sa jedne strane, deklarativni pristup jasno definiše način specifikacije transformacionih pravila. Sa druge strane, definisanje kompleksnijih pravila je otežano. U tom slučaju, ATL omogućava dodavanje akcionih blokova sa imperativnom sintaksom prethodno definisanim pravilima. Čak je omogućeno i pozivanje eksternog kôda za definisanje složene logike. Korišćnje jezika je podržano i odgovarajućim skupom alata., ADT, u okviru Eclipse platforme. Ovaj skup alata omogućava osnovne aktivnosti vezane za korišćenje jezika uopšte: editovanje, prevođenje, izvršavanje i debagovanje [Jouault08]. ATL omogućava pisanje transformacija između modela na višim nivoima apstrakcije. U tom pogledu ne omogućava generisanje izvornog kôda, ali za tu namenu se integriše sa čitavim skupom alata u okviru Eclipse platforme namenjenih transformacijama modela u različite tekstualne formate [Wolff10]. Ovaj skup alata je poznat pod nazivom MDT (*eng. Model-To-Text*) projekat.

2.6.1.3. TXL

TXL je dizajniran kao jezik za transformacije opšte namene [Cordy06]. Između ostalog, omogućava transformacije između programskih jezika. Jezik čine dve komponente:

- Strukturna komponenta, tj. specifikacija strukture koja se transformiše zasnovana na gramatikama u BNF (*eng. Backus Naur Form*) obliku,
- Funkcionalna komponenta, tj. skup transformacionih pravila zasnovanih na šablon/zamena obrascu i funkcionalnom programiranju.

Kako su pravila preslikavanja definisana funkcionalno i strukturni deo opisan preko gramatika, TXL se može smatrati pretežno deklarativnim. Upotreba gramatika omogućava transformacije iz konkretnih modela u neki od programskih jezika, što nije podržano u transformacijama zasnovanim na grafovima.

2.6.1.4. 4DML

4DML (*eng. four-dimensional markup language*) je jezik za transformacije originalno razvijen za adaptaciju Web sadržaja korisnicima sa posebnim potrebama [Brown04]. Dok je TXL namenjen transformaciji programskih jezika koji mogu biti predstavljeni kao sintaksno stablo, 4DML podržava transformaciju matričnih struktura. U tom pogledu njegova upotreba je komplikovana jer zahteva n-dimenzionalnu strukturu dokumenata koji su uglavnom organizovani kao stabla i grafovi.

2.6.1.5. UIML transformacije

UIML opisuje sve relevantne aspekte korisničkog interfejsa kao što su struktura, stil, sadržaj i ponašanje [Abrams99]. Važne osobine jezika su mogućnost povezivanja sa pozadinskom logikom i postojanje rečnika za međusobno mapiranje UIML modela, kao i za mapiranje u druge jezike. Kada govorimo o zvaničnoj specifikaciji jezika[10], ova dva aspekta su opisana u "*peers*" sekciji odgovarajućeg dokumenta. "*Presentation*" sekcija specifikacije opisuje mapiranje komponenata i njihovih atributa u konstrukcije odredišnog formata, dok "*logic*" sekcija daje opise povezivanja sa logikom aplikacije. Na primer, UIML komponente mogu biti mapirane na VoiceXML i HTML. Pored toga, povezivanje nije ograničeno na deklarativne formate, već se može ostvariti i veza sa, na primer, Java konstrukcijama. Kako su transformacije zasnovane na povezivanju imena komponenata i definisanju novih vrednosti za mapirane objekte, ovaj pristup se može smatrati deklarativnim. Međutim, ova preslikavanja su linearna i isuviše jednostavna za definisanje kompleksnijih transformacija. Očigledna prednost pristupa je što su apstraktni opisi korisničkih interfejsa i transformacije definisani istim jezikom.

2.6.1.6. XSLT

XSLT[11] (*eng. eXtensible Stylesheet Language Transformations*) transformiše XML ulaz u tekstualni (ugalvnom XML) izlaz. Ulaz u transformaciju predstavlja jedan ili više XML dokumenata. XSLT predstavlja skup šablonskih pravila (*eng. template rules*). Svako pravilo je sastavljeno od izvornog šablona, opcionog režima rada, atributa prioriteta i odredišnog šablona. Izvorni šablon je definisan XPath jezikom i upoređuje se sa čvorovima izvornog dokumenta. Sam proces mapiranja podrazumeva obilazak izvornog dokumenta i razmatranje pozicije, imena i atributa čvora. Kada se pronađe

[10] UIML Version 4.0 http://docs.oasis-open.org/uiml/v4.0/cd01/uiml-4.0-cd01.doc

[11] XSL Transformations (XSLT) Version 2.0, W3C Working Draft

odgovarajući čvor, on se transformiše u odredišni šablon. Na taj način se formira izlazni dokument. XPath razlikuje pet osnovnih tipova čvorova: logički čvorovi, brojevi, stringovi, skupovi čvorova i čvorovi podstabla. Za obradu čvorora izvornog dokumenta jezik obezbeđuje odgovarajuće tipove podataka (literatl, konstante, varijable, ključevi sa uslovima) i kontrolne strukture (iteracije, rekurzija, sortiranje). Pored toga, za naprednije obrade jezik sadrži skup ugrađenih stringovnih funkcija za kreiranje, brisanje, zamenu, kopiranje i konkatenaciju. Izlaz transformacije može biti XML dokument ili drugi tekstualni format poput HTML ili VoiceXML. Jezik je po svojoj prirodi deklarativan, ali omogućava imperativne iskaze poput interacija i uslova te se može smatrati i hibridnim.

2.6.1.7. GAC

GAC (*eng. General Adaptation Component*) jezik je razvijen u cilju adaptacije Web komponenti [Fiala05]. Za razliku od drugih jezika za transformacije, njegova namena je specifična i razmatra proces adaptacije kroz pristup i modifikaciju kontekstualnih podataka. U tom pogledu njegova primena je ograničena na XML i HTML dokumente. Njegova arhitektura je zasnovana na RDF *(eng. Resource Description Framework*) jeziku koji je namenjen deklarativnom opisu resursa u Web okruženju. Same transformacije čine pravila kojima se pridruženi uslovi. Pravila se mogu odnositi na adaptaciju ili ažuriranje konteksta korišćenja. Pravila adaptacije omogućavaju brisanje i zamenu XML fragmenata. Kako pravila jasno definišu operaciju koju treba treba izvršiti, ovaj pristup se može smatrati pretežno imperativnim.

2.6.1.8. RDL/TT

RDL/TT (*eng. Rule Description Language for Tree Transformation*) je jezik za kontekstno-osetljive transformacije deklarativnih opisa korisničkih interfejsa [Schaefer02]. Sam jezik je najpre bio zamisljen kao domenski specifičan jezik za adaptaciju Web sadržaja različitim uređajima. Jezik koristi Java sintaksu za definisanje transformacionih pravila koja rade na DOM XML stablima. On definiše obrasce pretrage na osnovu naziva tagova i složena pravila restruktuiranja nad identifikovanim tagovima. Pored toga, jezik predviđa upotrebu varijabli za čuvanje kontekstualnih informacija u transformacijama tako da one mogu biti prilagođene različitim ciljnim platformama i kontekstima korišćenja. Transformaciona pravila su imperativna i integrišu kontrolne strukture kao što su petlje i grananja.

2.6.2. Uporedni pregled transformacija u modelski zasnovanom dizajnu korisničkih interfejsa

Tabela 2.5 daje uporedni pregled prethodno opisanih tipova transformacija. Upotreba zagrada označava da je svojstvo u principu podržano (uz dodatne napore), ali da jezik generalno nije dizajniran da ga podržava. Model programiranja se razmatra iz praktičnih razloga. Posmatraju se transformacije na različitim nivoima apstrakcije, tj. transformacije na nivou modela i transformacije na nivou kôda. Odvojeno se posmatra svojstvo transformacije nad XML reprezentacijama modela. Pored toga, razmatra se i svojstvo generisanja kôda koje zaokružuje čitav proces modelovanja. Proširivost i

mogućnost parametrizacije su naročito značajni u pogledu specifičnih zadataka razvoja korisničkog interfejsa, kao što je razvoj kontekstno-osetljivih aplikacija.

Tabela 2.5. *Uporedni pregled jezika za transformacije.*

Svojstvo	ATL	GT	TXL	4DML	XSLT	GAC	UIML	RDL
Deklarativan	+	+	+	+	+	-	+	-
Imperativan	+	-	-	-	+	+	-	+
Transformacija modela	+	+	(+)	(+)	(+)	(+)	(+)	(+)
XML transformacija	-	-	(+)	(+)	+	+	-	+
Transformacija kôda	-	-	+	(+)	-	-	-	-
Generisanje kôda	-	-	+	+	(+)	-	+	(+)
Složena mapiranja	+	+	+	+	+	+	-	+
Proširivost	+	-	-	-	-	-	-	+
Parametrizovanost	-	-	-	-	-	+	-	+

Razlika između deklarativnog i imperativnog programiranja je značajna za dizajnere kojima može više odgovarati jedan od modela. Sa druge strane, jasnu razliku između ova dva modela programiranja nije uvek moguće napraviti. Na taj način neki jezici kombinuju oba modela programiranja, gde je jedan obično dominantniji.

U pogledu transformacije UI modela na višim slojevima apstrakcije, izdvajaju se ATL i GT. Pored njih, alati koji rade sa XML dokumentima mogu obrađivati XML reprezentacije modela. Zbog toga se XSLT, GAC i RDL/TT mogu svrstati u ovu grupu. TXL i 4DML omogućavaju transformacije modela, ali modeli moraju biti u odgovarajućem formatu (BNF gramatika ili matrična stuktura). Kod UIML transformacije su jednosmerne, počinju od apstraktnih interfejsa, preko konkretnih i kao rezultat daju finalne korisničke interfejse.

XSLT, RDL/TT i GAC rade sa XML formatima i ne omogućavaju transformacije kôda. U ovom pogledu izdvaja se TXL.

Mogućnost generisaja kôda poseduju 4DML, TXL, UIML i XSLT.

Osim UIML, svi jezici omogućavaju kompleksna mapiranja. Ovde se konkretno misli na restruktuiranje izvornih operacija. U tom kontekstu UIML obezbeđuje linearno mapiranje jedan-na-jedan.

RDL/TT se može proširivati dodatnim funkcionalnostima i parametrizovati. ATL je proširiv u pogledu poziva *native* operacija. GAC može obrađivati i menjati kontekstualne informacije. Dok se kod GAC modifikacija konteksta odigrava unutar pravila, RDL vrši razdvajanje tako što se podaci o kontekstu čuvaju u varijablama, a njihova obrada vrši u okviru pravila.

U opštem slučaju, izbor alata za transformaciju zavisi od modela, njihovog formata i konteksta aplikacije koja će se koristiti. To može biti i kombinacija različitih oblika transformacija na različitim nivoima apstrakcije ili u različitim fazama razvoja. Veliki broj alata za modelovanje korisničkih interfejsa ima ugrađene transformacije koje se ne mogu editovati i kontrolisati. Ovi alati obično imaju mogućnost generisanja modela u XML formatu (najčešće kao XMI). Na taj način su ovi modeli dostupni dizajnerima van konteksta alata i mogu ih dalje obrađivati u skladu sa potrebama.

2.7. *Rezime poglavlja*

U ovom odeljku je data analiza sa kritičkim osvrtom na pogodnosti korišćenja postojećih rezultata u razvoju kontekstno-osetljivih korisničkih interfejsa. Analiza je izvršena kroz glavne aspekte kontekstno-osetljivih interfejsa, tj. aspekt modelovanja računarske platforme, aspekt modelovanja okruženja i aspekt modelovanja korisnika. Na osnovu izvršene analize uočili smo neke elemente koji se dalje mogu unaprediti i na taj način poboljšati kvalitet korisničkih interfejsa kako sa stanovišta korišćenja, tako i sa stanovišta razvoja.

2.7.1. Aspekt okruženja

Kada govorimo o fizičkom okruženju interakcije, ono generalno nije eksplicitno modelovano i uvedeno u razvoj korisničkih interfejsa. Sirovi podaci o kontekstu okruženja niskog nivoa obično se prikupljaju preko različitih fizičkih senzora. Ovi podaci mogu biti podaci o fizičkom okruženju, podaci vezani za lokaciju, zvuk, pritisak vazduha, uslovi osvetljenja i dr. Tipovi podataka, formati i nivo apstrakcije senzora se razlikuju. Uređaji i fizički senzori kontekstno-osetljivih korisničkih interfejsa koriste različite skale i merne jedinice. Kontekstno-osetljivi korisnički interfejsi čuvaju podatke različitih formata i nivoa apstrakcije, kao i sa različitim relacijama u kontekstnoj bazi. Pored toga, neki od ovih sistema čuvaju istoriju konteksta koju čine podaci dobijeni od senzora kako bi pružili proaktivne servise korisnicima [Congleton08]. Istorijat čine podaci o lokaciji, temperaturi, nivou osvetljenosti, aktivnostima korisnika, korišćenim uređajima, stanjima uređaja, izabranim servisima i dr. Ovi podaci se koriste u kontekstu specifičnih okruženja i aplikacija kao što pametna okruženja (na primer mesta stanovanja, bolnice, učionice), muzeji, turistički vodiči, informacioni sistemi kao što su sistemi za podršku odlučivanju, aplikacije socijalnih mreža, sistemi elektronske trgovine i web servisi).

Kontekstno-osetljivi korisnički interfejsi moraju posedovati mehanizme za upravljanje velikom količinom raznovrsnih podataka koji opisuju okruženje interakcije. Na ovaj način će korisnicima blagovremeno biti pruženi odgovarajući servisi. Postojanje ontologije koja na uniforman način opisuje podatke o okruženju olakšaće razvoj i nadogradnju sistema. Korišćenje kreirane ontologije u metodologijama razvoja kroz korišćenje odgovarajućih alata obezbediće unificirani razvoj kako kontekstno-osetljivih korisničkih interfejsa, tako i kontekstno-osetljivih sistema uopšte.

2.7.2. Aspekt računarske platforme

Računarska platforma može biti razmatrana u širem smislu i u užem smislu. Modelovanje računarske platforme u širem smislu ima slojevitu strukturu, tj. može se posmatrati kroz mrežni sloj, sloj sistemskog softvera (*eng. middleware*), aplikativni sloj i sloj korisničkog interfejsa. Mrežni sloj čine implementacije protokola i servisa koje oni obezbeđuju. Sloj sistemskog softvera modeluje softverske komponente niskog nivoa dostupne na konkretnoj platformi koje realizuju funkcionalnosti na koje se direktno oslanjaju aplikativni servisi. Aplikativni sloj obuhvata konkretne aplikativne servise sa kojima korisnik interaguje putem odgovarajućeg interfejsa. Sloj korisničkog interfejsa obezbeđuje korišćenje servisa aplikativnog sloja od strane korisnika.

Modelovanje računarske platforme u užem smislu obuhvata modele fizičkih uređaja za interakciju i modele konkretnih entiteta korisničkog interfejsa preko kojih se vrši interakcija. Kada govorimo o konkretnim entitetima, u najvećem broju slučajeva oni su predstavljeni kao vidžeti grafičkih korisničkih interfejsa. Apstrakcije komponenata korisničkog interfejsa se mogu naću u odgovarajućim jezicima. Ovi oblici modela korisničkih interfejsa se mogu naći u gotovo svim arhitekturama i metodologijama razvoja [Nunes00a], [da Silva03], [Mori04], [Demeure05]. Naravno, svaka od arhitektura ovaj aspekt računarske platforme opisuje na specifičan način. U pogledu korišćenih načina komunikacije najzastupljeniji su modeli grafičkih i glasovnih korisničkih interfejsa.

Za opise modela računarskih platformi primetno je korišćenje tehnologija koje su našle široku prihvaćenost kako u industriji, tako i u akademskoj zajednici. Za definicije modela koristi se UML jezik koji predstavlja industrijski standard za modelovanje. Za razmenu podataka koriste se tehnologije zasnovane na XML jeziku. Pored toga, pomenute tehnologije su direktno podržane i u metodologijama razvoja softvera kao što su RUP i MDD.

2.7.3. Aspekt čoveka

Većina kontekstno-osetljivih korisničkih interfejsa se fokusira na spoljašnji kontekst koji se još naziva i fizički kontekst. On obuhvata podatke dobijene od fizičkih senzora. Spoljašnji kontekst je važan sa stanovišta mogućnosti pružanja odgovarajućeg servisa na osnovu analize podataka o okruženju. Sa druge strane, poznavanje spoljašnjeg konteksta nije dovoljno da bi se obezbedili personalizovani servisi kao što su dobijanje informacija (*eng. infromation retrieval*), donošenje odluka, nadgledanje sistema i dr. Sa stanovišta interakcije korisnika sa sistemom, servisi se moraju prilagoditi potrebama i karakteristikama pojedinih korisnika ili grupe korisnika. Drugim rečima, u obzir se mora uzeti i kontekst korisnika. Kontekst korisnika je uglavnom razmatran kroz kognitivne aktivnosti korisnika niskog nivoa [Calvary03]. Ove aktivnosti su najčešće opisane modelima zadataka. Pored kognitivnih aktivnosti, potrebno je uzeti u obzir senzorske i percepcijske mehanizme ljudi u kontekstno-osetljivim okruženjima jer oni direktno utiču na izvršenje kognitivnih aktivnosti. Aspekt motoričkih osobina čoveka, kao što je ekstrakcija pokreta u fizičkom okruženju, se takođe mora uzeti u obzir.

Postojeća istraživanja vezana za pružanje personalizovanih servisa u kontekstno-osetljivim sistemima imaju nekoliko ograničenja. Ova ograničenja se pre svega ogledaju u tome da korisnici svoje potrebe moraju unositi direktno i da je dodavanje novog korisnika u sistem složeno. Mehanizmi modelovanja korisnika i personalizacije servisa se svode na evidenciju i praćenje istorijata korišćenja sistema, na osnovu kojeg se korisniku na odgovarajući način pruža određeni servis [Bellotti08], [Hartmann09a].

Kada govorimo o razvoju korisničkih interfejsa, možemo uočiti nekoliko načina za modelovanje korisnika. Najelementarniji, ujedno i najzastupljeniji, je zasnovan na opisivanju logičkih aktivnosti korisnika. Ovo je realizovano korišćenjem modela zadataka. U ranim fazama razvoja model zadataka se povezuje sa domenskim modelom koji opisuje specifične domenske objekte nad kojima korisnik izvršava zadatke. U kasnijim fazama razvoja, identifikovani zadaci se transformišu u odgovarajuće elemente korisničkog interfejsa. Ovakav pristup modeluje proizvoljnog korisnika sa pretpostavkom da će sistem biti korišćen na identičan način od strane svakog korisnika. Međutim, u praksi se javlja širok

spektar korisnika sa svojim specifičnostima u pogledu preferencija (*eng. user preferences*) i karakteristika. To je uslovilo pojavu sistema za razvoj korisničkih interfejsa koji odvojeno integrišu modele korisnika. Ovi modeli najčešće opisuju preferencije korisnika (*eng. user preferences*). Postoji nekoliko tehnika interakcije pomoću kojih se formiraju ovi modeli. Jedna tehnika podrazumeva formiranje modela preferencija tako što korisnik u interakciji sa sistemom predlaže ili bira inkrementalna poboljšanja konkretnog interfejsa koji generiše sistem (*eng. user-driven example critiquing*). Ova tehnika ne garantuje potpunu pokrivenost prostora dizajna korisničkog interfejsa zbog raznolikosti korisnika. Druga tehnika predviđa da sistem prezentuje korisniku konkretna rešenja i prikuplja povratne informacije od korisnika (*eng. system-driven preference elicitation*). Hibridno rešenje. tj. kombinacija ove dve tehnike daje optimalno rešenje [Gajos10].

Uopšteno posmatrano, korisnik kao učesnik u interakciji nije eksplicitno modelovan u postojećim sistemima. Drugim rečima, ne postoje *a priori* modeli korisnika koji opisuju senzorske, percepcijske, kognitivne i motoričke osobine čoveka na visokom nivou. Korišćenje ovakvih modela, zajedno sa postojećim tehnikama modelovanja korisnika, unapredilo bi razvoj i upotrebljivost korisničkih interfejsa. Ovo zahteva interdisciplinaran pristup i korišćenje znanja iz prirodnih i društvenih naučnih disciplina.

2.7.4. Aspekt razvoja

Razvoj korisničkih interfejsa možemo razmatrati sa konceptualnog stanovišta, stanovišta metodologija i arhitektura i stanovišta korišćenih tehnologija.

Posmatrajući koncept razvoja možemo uočiti dva pristupa, pristup orijentisan ka korisniku i pristup orijentisan ka sistemu. Pristup orijentisan ka korisniku je zasnovan na logičkim aktivnostima korisnika koje se opisuju modelima zadataka. Glavne prednosti ovakvog pristupa se ogledaju u jasno definisanom skupu aktivnosti koje korisnički interfejs treba da podrži. Međutim, kreiranje modela zadataka može biti složeno u smislu inicijalnog posmatranja korisnika sistema, kao i potrebnog znanja i iskustva. Učenje i korišćenje predloženih notacija za opis zadataka [Mori04], [Klug05], [Barboni10] iziskuje znatno vreme i napor. Samim tim što nisu kreirane korišćenjem standardizovanih i opšteprihvaćenih tehnologija, otežana je njihova integracija u standardne metodologije razvoja i nisu šire prihvaćene. Pored toga, specifikacije zadataka se tipično prevode u funkcionalne specifikacije orijentisane ka podacima [Paterno08b]. Veliki broj sistema za direktnu manipulaciju podržava širok spektar mogućih zadataka i omogućava simultano izvršavanje aktivnosti što dalje usložnjava odgovarajuće modele zadataka. Pristup orijentisan ka sistemu zasnovan je na funkcionalnoj specifikaciji korisničkog interfejsa u smislu objekata i servisa koje oni pružaju. Ovde postoji prirodno preklapanje sa konceptima kao što je objektno-orijentisana paradigma, te je stoga ovaj pristup prilično rasprostranjen. Pretpostavlja se da će sistem biti korišćen na identičan način za svakog korisnika. Sa druge strane, postoji velika raznolikost korisnika u pogledu njihovih mogućnosti i preferencija. Iako neki sistemi raspolažu mehanizmima personalizacije korisnika, njihovo korišćenje je otežano zbog razlika od sistema do sistema.

Kada govorimo o metodologiji i arhitekturi razvoja korisničkih interfejsa uočava se široka upotreba modelski zasnovanih pristupa. Modelski zasnovani pristupi u dizajnu korisničkih interfejsa su u početku bili kritikovani [Myers00]. Međutim, pojava OMG MDA standarda i razvoj MDE discipline je u praksi dovela do znatnog unapređenja u modelski zasnovanom razvoju korisničkih interfejsa. Prednosti modelski zasnovanog pristupa se mogu posmatrati u kontekstu metodologije, ponovne upotrebljivosti i konzistentnosti. U pogledu metodologije jasno su definisane faze razvoja koje prati odgovarajuća arhitektura. Pored toga, prirodno je podržan dizajn orijentisan ka korisniku tako što je omogućeno kreiranje modela korisnika i zadataka nezavisnih od konretnih domena i platformi. Ponovna upotrebljivost je unapređena portabilnošću modela između različitih uređaja što je obezbeđeno odgovarajućim alatima. Platformski nezavisni modeli sami po sebi obezbeđuju ponovnu upotrebljivost komponenti korisničkog interfejsa. Konzistentnost je obezbeđena sa stanovišta razvoja i stanovišta upotrebe. U pogledu razvoja može se posmatrati konzistentnost između ranih faza razvoja i konačnog interfejsa. U kontekstu većeg broja platformi postiže se određen stepen konzistentnosti između korisničkog interfejsa generisanog na različitim ciljnim platformama. Ovo nije uvek moguće korišćenjem drugih tehnika gde se razvoj svake verzije korisničkog interfejsa odvija odvojeno.

Posmatrajući kontekstno-osetljive korisničke interfejse, možemo uočiti četiri vrste alata korišćenih za njihov razvoj: alati zasnovani na specifičnim jezicima, okviri aplikacija, interaktivni alati i alati zasnovani na modelima. Prva grupa alata koristi specifične jezike za kreiranje korisničkih interfejsa [Abrams99]. Ovi jezici mogu biti različiti, kao na primer formalne gramatike, dijagrami stanja, deklarativni jezici, jezici za obradu događaja i dr. Okviri aplikacija apstrahuju važne elemente korisničkog interfejsa, kao što su prozori i komande. Programer vrši specijalizaciju ovih klasa radi dobijanja aplikativno specifičnih elemenata, na primer definisanje sadržaja prozora i komandi koje će se koristiti [Korhonen07]. Korišćenje pomenutih vrsta alata zahteva znanje i angažovanje dizajnera u kreiranju konkretnih korisničkih interfejsa sa stanovišta poznavanja same tehnologije. Interaktivni alati donekle rešavaju ovaj problem tako što dizajnerima obezbeđuju predefinisane biblioteke vidžeta. Prilikom njihovog korišćenja u pozadini se generišu opisi komponenti na odgovarajućem jeziku. Upotreba ovih alata ne omogućava dinamičke promene vidžeta. Na primer, promena sadržaja menija ili rasporeda dijaloga na osnovu stanja programa mora biti ručno programirana. Zbog toga su se pojavili alati zasnovani na modelima. Ovi alati polaze od različitih modela korisničkih interfejsa, kao što su modeli zadataka [Mori04] ili modeli uređaja i korisnika [Gajos10].

U poslednjoj deceniji pojavio se veći broj tehnologija koje se koriste u razvoju korisničkih interfejsa. Tehnologije su vezane za pojedine faze razvoja. Kada govorimo o ranim fazama razvoja, tu se pre svega misli na UML jezik i XML tehnologije. U kasnijim fazama razvoja koristi se veliki broj jezika samo za opis korisničkih interfejsa čiji je pregled dat u ovom poglavlju. U poslednje vreme uočljivo je korišćenje hibridnih tehnologija koje kombinuju deklarativne i imperativne jezike [TourDeFlex10]. Pored toga, u poglavlju je dat uporedni pregled jezika za transformacije korišćenih u razvoju korisničkih interfejsa. Na osnovu pregleda mogu se uočiti prednosti ATL jezika, kako u pogledu mogućnosti, tako i u pogledu podrške i integracije sa drugim alatima.

3. Razvoj kontekstno-osetljivih korisničkih interfejsa

Razvoj kontekstno-osetljivih korisničkih interfejsa zateva interdisciplinaran pristup i korišćenje znanja iz različitih oblasti. Kada ljudi komuniciraju međusobno, oni su u mogućnosti da implicitno koriste informaciju o situaciji, drugim rečima kontekst, kako bi povećali sam kapacitet komunikacije. Ovo se, međutim, ne može preslikati na komunikaciju čoveka i računara. Današnji računari nemaju ugrađenu mogućnost korišćenja konteksta u komunikaciji sa čovekom. Omogućujući računarima pristup kontekstu obogaćuje se komunikacija čoveka i računara. Kontekst je definisan kao triplet <*korisnik, platforma, okruženje*> [Calvary03]. Kako bi kontekstno-osetljivi korisnički interfejsi učinili komunikaciju između čoveka i računara prirodnijom i približili je svakodnevnoj međuljudskoj komunikaciji, potrebno je uvažiti niz faktora kao što su čovekova senzorska fiziologija i anatomija, percepcija, saznajni mehanizmi i socijalna interakcija. Takođe, potrebno je uzeti u obzir karakteristike medija i fizičkog okruženja korisnika. Na kraju, razvoj korisničkog interfejsa treba bolje da uvaži karakteristike hardverskih uređaja koji se koriste u komunikaciji sa korisnikom, dostupne softverske resurse, kao i karakteristike programskog sistema koji treba da koristi korisnički interfejs.

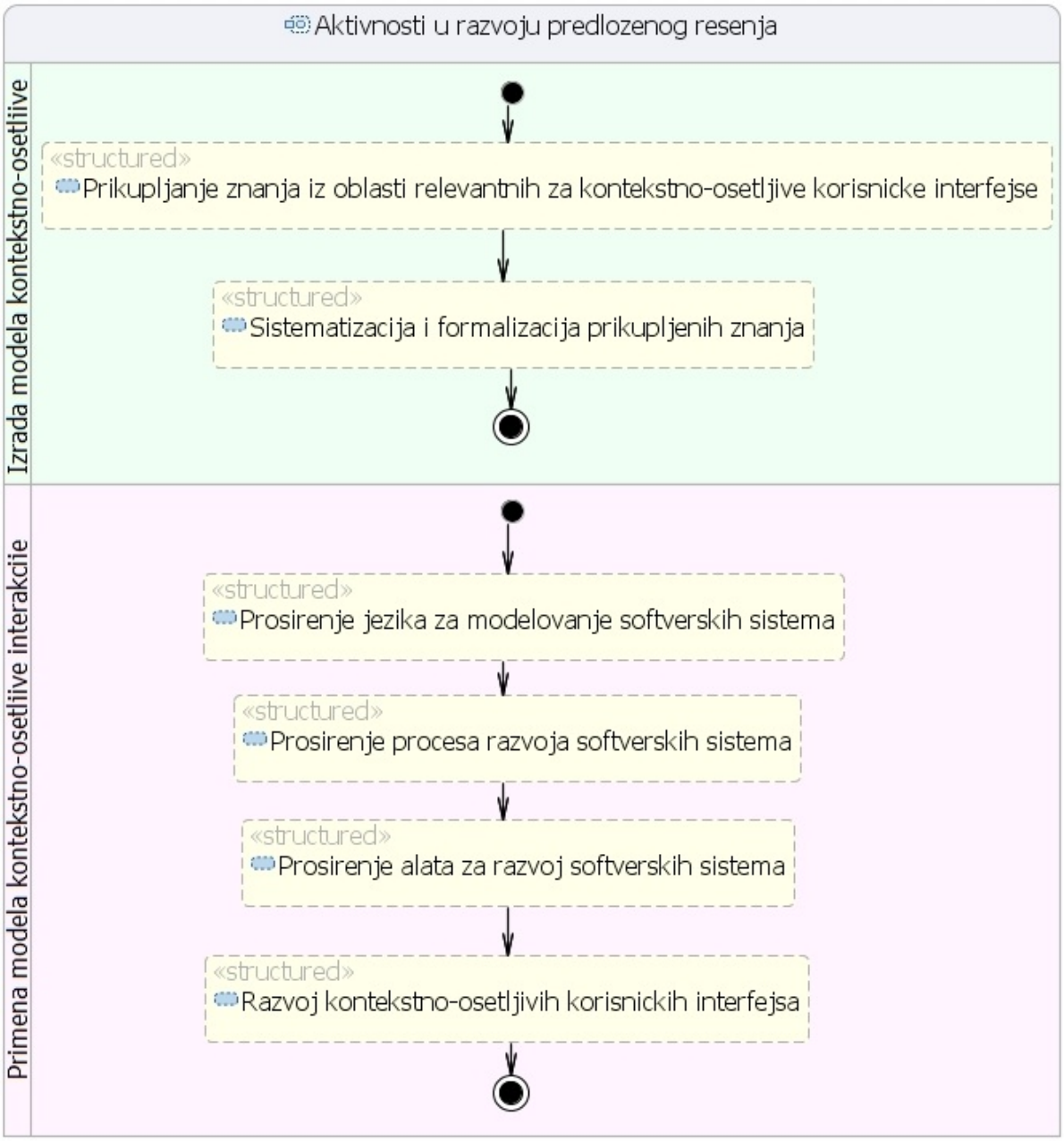
U ovom radu predlažemo pristup razvoju kontekstno-osetljivih korisničkih interfejsa kreiranjem modela kontekstno-osetljive interakcije čoveka i računara. Model kontekstno-osetljive interakcije čoveka i računara predstavlja jedinstven, formalan i standardizovan opis osnovnih koncepata i relacija između faktora od interesa za razvoj kontekstno-osetljivih korisničkog interfejsa. Polazeći od modela kontekstno-osetljive interakcije, razradili smo pristup razvoju kontekstno-osetljvih korisničkih interfejsa oslanjajući se na principe *modelski zasnovanog razvoja softvera* [Selic03].

Korišćenjem modela kontekstno-osetljive interakcije moguće je na različite načine unaprediti razvoj konteksno-osetljivih korisničkih interfejsa. Prvo, model može da služi kao baza znanja ili ontologija koncepata bitnih za kontekstno-osetljivu interakciju između čoveka i računara. Korišćenjem modela moguće je semantički proširiti razvojna okruženja i proces razvoja uvođenjem primitiva specifičnih za modelovanje komunikacije između čoveka i računara na različitim nivoima apstrakcije. Na taj način moguće je potpunije i preciznije opisati poželjna svojstva komunikacije između čoveka i računara za dati programski sistem. Formalni, semantički bogatiji modeli korisničkih interfejsa mogu se koristiti u različite svrhe, počevši od dokumentacije, pa do automatizacije dela razvojnog procesa i transformacija korisničkih interfejsa.

U ovom poglavlju izlažemo osnovnu ideju razvoja kontekstno-osetljivih korisničkih interfejsa pomoću modela kontekstno-osetljive interakcije, kao i primenjena načela i smernice kojima smo se rukovodili u toku naših istraživanja. Prvo opisujemo osnovnu ideju pristupa razvoju kontekstno-osetljivih korisničkih interfejsa pomoću predloženog modela. U okviru ovog dela opisujemo predloženi model kontekstno-osetljive interakcije, kao i mogućnosti njegove primene u procesu izrade korisničkih interfejsa. Zatim opisujemo organizaciju i arhitekturu rešenja. Na kraju dajemo sažetak i diskutujemo o očekivanoj dobiti od predloženog pristupa.

3.1. *Model kontekstno-osetljive interakcije čoveka i računara*

U tezi smo predložili pristup razvoju kontekstno-osetljivih korisničkih interfejsa pomoću modela kontekstno-osetljive interakcije između čoveka i računara. Predloženo rešenje je *rezultat primene interdisciplinarnog pristupa* i objedinjavanja znanja iz više naučnih oblasti od interesa za kontekstno-osetljivu interakciju između čoveka i računara. Slika 3.1 prikazuje aktivnosti u razvoju predloženog rešenja pomoću dijagrama aktivnosti visokog nivoa apstrakcije.



Slika 3.1. *Aktivnosti u razvoju predloženog rešenja.*

Osnovu predloženog rešenja čini model kontekstno-osetljive interakcije između čoveka i računara. Model kontekstno-osetljive interakcije predstavlja jedinstven i formalan opis osnovnih koncepata i

relacija između njih od interesa za razvoj kontekstno-osetljivog korisničkog interfejsa. Pri izradi modela, naš cilj je bio identifikacija osnovnog skupa koncepata iz svakog od relevantnih domena i uspostavljanje relacija između ovih koncepata. Ovaj model može da se posmatra i kao *ontologija* koncepata bitnih za razvoj korisničkih interfejsa uopšte, jer sadrži pregled najznačajnijih rezultata i saznanja iz predmetnih oblasti i može da se koristi kao zajednički i standardizovan jezik za deljenje i korišćenje tih znanja.

Korišćenjem modela kontekstno-osetljive interakcije moguće je semantički *proširiti jezik za modelovanje* korisničkih interfejsa uvođenjem primitiva specifičnih za modelovanje komunikacije između čoveka i računara na različitim nivoima apstrakcije. U ovom slučaju, model kontekstno-osetljive interakcije može da služi kao metamodel za uvedena proširenja. Na taj način moguće je potpunije i preciznije opisati poželjna svojstva komunikacije između čoveka i računara za dati programski sistem. Takođe, model kontekstno-osetljive interakcije može da služi kao zajednički format za opis interakcije i funkcionalnosti različitih alata, što dalje olakšava komunikaciju između njih.

Pored proširenja jezika za modelovanje, u radu smo istraživali i mogućnosti *semantičkog proširenja procesa razvoja* kontekstno-osetljivih korisničkih interfejsa. Razvoj nekog softverskog sistema odvija se kroz više faza, u okviru više aktivnosti, kreiranjem više modela na različitim nivoima apstrakcije. Ovde smo prvenstveno razmatrali mehanizme integracije predloženih koncepata u standardne metodologije razvoja softverskih sistema, kao i prednosti koje takav pristup može doneti projektantima u različitim fazama razvoja.

Formalan opis modela omogućava njihovu integraciju u *razvojne alate korisničkih interfejsa*. Na ovaj način moguće je proširiti postojeće ili razviti nove alate koji mogu da olakšaju i unaprede razvoj kontekstno-osetljivih korisničkih interfejsa.

Sa predloženim elementima rešenja, moguće je na različite načine unaprediti razvoj kontekstno-osetljivih korisničkih interfejsa. Predložena proširenja jezika za modelovanje i procesa razvoja softvera mogu da obezbede standardan opis modela i korišćenje standardne metodologije razvoja korisničkih interfejsa. Razvijeni alati mogu da automatizuju deo procesa razvoja korisničkih interfejsa, kao i da omoguće transformacije korisničkih interfejsa u cilju njihove migracije sa jedne platforme na drugu.

U skladu sa prethodno navedenim, istraživanje u tezi je organizovano u dve celine (Slika 3.1):

- *Izradu modela kontekstno-osetljive interakcije između čoveka i računara*, gde smo na jedinstven način objedinili relevantna znanja i zajedničke koncepte iz različitih domena i
- *Primenu modela kontekstno-osetljive interakcije između čoveka i računara* u razvoju kontekstno-osetljivih korisničkih interfejsa.

U nastavku dajemo detaljniji opis navedenih celina.

3.1.1. Izrada modela kontekstno-osetljive interakcije

Osnovu predloženog pristupa čini model kontekstno-osetljive interakcije između čoveka i računara. Ovaj model predstavlja zajednički i standardizovan jezik za deljenje i korišćenje znanja o fenomenima iz različitih domena relevantnih za razvoj kontekstno-osetljivih korisničkih interfejsa. Predloženi model je objektno-orijentisan, gde se koncepti i relacije između njih opisuju kao skup međusobno povezanih objekata. Važno je napomenuti da je koncept modela i metamodela tesno povezan sa konceptom ontologije iz veštačke inteligencije [Gasevic06]. Zato se model kontekstno-osetljive interakcije između čoveka i računara može posmatrati i kao ontologija koja pruža uniforman pogled na važne relacije između koncepata od interesa za opis kontekstno-osetljive interakcije između čoveka i računara.

Prilikom izrade ovog modela, naš cilj je bio da identifikujemo osnovni skup koncepata iz svakog od relevantnih domena i da uspostavimo relacije između ovih koncepata. U modelu je identifikovan veliki broj koncepata, te je model kontekstno-osetljive interakcije čoveka i računara logički organizovan u više celina u skladu sa strukturom koncepta interakcije. Drugim rečima, koncepti su podeljeni u četiri grupe u zavisnosti od toga da li opisuju čoveka, okruženje, računarsku platform i načine komunikacije između čoveka i računara.

Detaljniji opis modela kontekstno-osetljive interakcije između čoveka i računara dat je u poglavlju 4.

3.1.2. Primena modela kontekstno-osetljive interakcije

Predloženi model kontekstno-osetljive interakcije čini osnovu za razmatranje različitih aspekata razvoja kontekstno-osetljivih korisničkih interfejsa:

- *Proširenje jezika za modelovanje korisničkih interfejsa*, gde smo razmotrili mogućnosti semantičkog proširenja jezika UML sa primitivama iz modela kontekstno-osetljive interakcije čoveka i računara.

- *Proširenje procesa razvoja korisničkih interfejsa*, gde smo razmotrili mehanizme integracije predloženih koncepata u standardne metodologije razvoja softverskih sistema, kao što je Unified proces, kao i prednosti koje takav pristup može doneti projektantima u različitim fazama razvoja.

- *Proširenje alata za razvoj korisničkih interfejsa*, gde smo razmatrali realizaciju prethodno navedenih aspekata u konkretne alate namenjene projektantima i programerima korisničkih interfejsa. Pored toga, posebna pažnja je posvećena sistemu transformacijama između odgovarajućih razvojnih modela, kao i transformaciji modela u skeleton kôda aplikacije.

U narednim odeljcima detaljnije ćemo opisati svaku od predloženih primena modela kontekstno-osetljive interakcije između čoveka i računara.

3.1.2.1. *Proširenje jezika za modelovanje*

Uzimajući u obzir prirodu rešavanog problema i dostupne razvojne alate, u ovom radu opredelili smo se za razmatranje mogućnosti proširenja objektno-orijentisanih jezika za modelovanje sistema. Zbog široke prihvaćenosti, standardizacije i dostupnosti razvojnih alata, kao jezik za modelovanje sistema koristili smo *Unified Modeling Language* (UML) [Booch99]. Bogata semantika i mogućnosti proširenja jezika UML obezbedili su njegovu primenu u raznim domenima. Projektanti jezika UML su predvideli da upotreba UML-a u njegovoj osnovnoj formi neće moći da zadovolji sve situacije, kao i da njegova semantika neće moći da obuhvati koncepte iz svih domena i arhitektura. Zbog toga je kao sastavni deo jezika UML definisan formalan mehanizam proširenja semantike ovog jezika. Ovaj mehanizam je realizovan u formi UML profila [Kleppe03]. Koncept profila odgovara specijalizaciji UML jezika za opis specifičnog domena. Drugim rečima, profil predstavlja prošireni UML metamodel koji definiše jezik za opis određenog domena, pri čemu zadržava značenje i ograničenja postojećih elemanata UML metamodela [UML10]. Osnovni gradivni element profila predstavlja stereotip (*eng. stereotype*). Steretipovi se koriste za dodavanje ključnih reči (*eng. keywords*), ograničenja (*eng. constraints*), vizuelnih elemenata (*eng. images*) i oznaka (*eng. tagged values*) elementima modela. Upotreba profila kao mehanizma proširenja UML za kreiranje domenski specifičnih jezika (*eng. Domain Specific Language - DSL*) se preporučuje [Brucko8]. Većina postojećih standardizovanih profila predstavlja opise specifičnih programskih jezika i tehnologija kao što su Java, EDOC, CORBA. Modeli koji odgovaraju ovakvim profilima su platformski specifični. Pored ove vrste profila, mogu se naći i profili namenjeni razvoju korisničkih interfejsa [da Silva03], [Bergh05].

Korišćenje jezika UML omogućuje da predložena proširenja budu dostupna širem skupu projektanata i programera. Jezik UML je široko prihvaćen, familijaran mnogim projektantima i programerima, uči se u okviru osnovnih studija na velikom broju univerziteta, te je podržan sa velikim brojem knjiga i kurseva. Ova osobina jezika UML je veoma bitna, jer postojeći specijalizovani alati za razvoj korisničkih interfejsa problem interakcije između čoveka i računara rešavaju parcijalno i korišćenjem nestandardnih metodologija, što značajno umanjuje njihovu praktičnu upotrebljivost, a usložnjava integraciju korisničkog interfejsa sa ostatkom softverskog sistema.

Detaljniji opis proširenja jezika za modelovanje dat je u poglavlju 5.

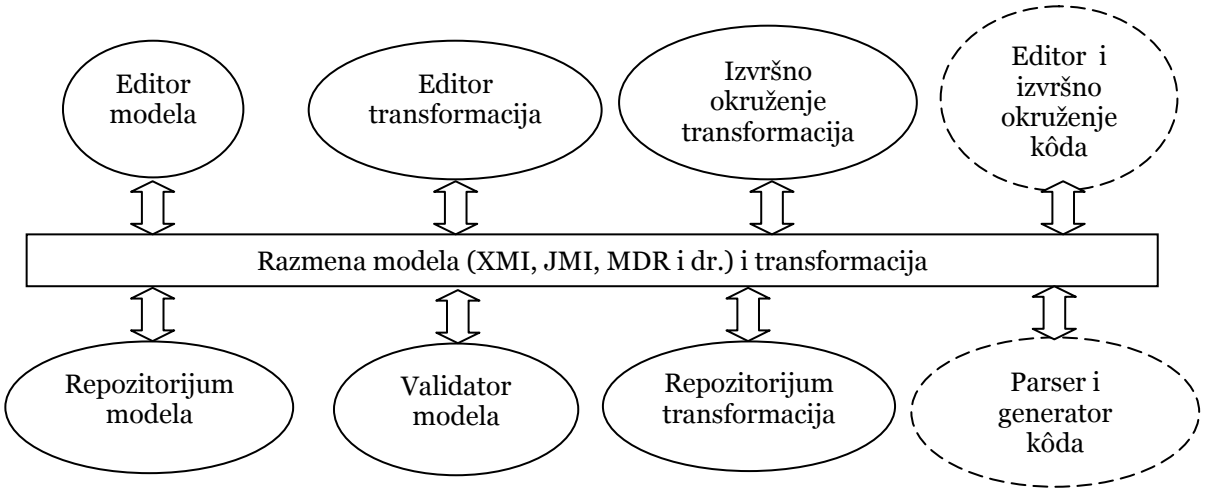
3.1.2.2. *Proširenje procesa razvoja*

Korišćenje standardnih, široko prihvaćenih tehnologija i pristupa može da unapredi razvoj kontekstno-osetljivih korisničkih interfejsa jer je moguće iskoristiti postojeće alate i iskustva iz drugih oblasti. Međutim, aktuelne metodologije nisu prilagođene potrebama detaljnijeg definisanja kontekstno-osetljive interakcije između čoveka i računara. U radu smo istraživali načine *semantičkog proširenja procesa razvoja* softverskih sistema sa konceptima iz oblasti kontekstno-osetljive interakcije između čoveka i računara. U radu su razmatrane mogućnosti proširenja *Unified* procesa korišćenjem UML mehanizama za uvođenje novih elemenata. Prema *Unified* procesu, razvoj nekog softverskog sistema odvija se kroz više faza i aktivnosti, kreiranjem modela na različitim nivoima apstrakcije. Faze kroz koje se obično prolazi u realizaciji softverskih sistema su [Jacobson99]:

specifikacija zahteva, analiza, projektovanje, implementacija i testiranje. Detaljniji opis proširenja procesa u pojedinim fazama razvoja dat je u poglavlju 6.

3.1.2.3. Proširenje alata za razvoj softverskih sistema

Osnovni cilj modelski zasnovanog pristupa razvoju softvera jeste povećanje produktivnosti i skraćenje vremena razvoja. Ovo se postiže tako što se projektanti oslanjaju na koncepte koji su bliži domenu problema, umesto da koriste koncepte iz programskih jezika [Selic03]. Da bi se ovo postiglo potrebno je ugraditi odgovarajuće mehanizme u softverska razvojna okruženja (slika 3.2). Ove mehanizme možemo razmatrati sa aspekta modela i transformacija između njih. Razvoj softverskog proizvoda obuhvata kreiranje modela koji opisuju različite aspekte sistema i koji su definisani na različitim nivoima apstrakcije. Ovde je potrebno obezbediti validnost modela, kao i konzistentnost između modela, kako na istim, tako i na različitim nivoima apstrakcije. Editor model omogućava konstrukciju i modifikciju modela, dok repozitorijum predstavlja skladište modela u odgovarajućem formatu (XMI, JMI, MDR i dr.). Validator proverava ispravnost modela u odnosu na skup definisanih ograničenja kako bi se on mogao koristiti u transformacijama. Jedan od ključnih izazova modelski zasnovanog razvoja softvera jeste transformisanje modela na visokom nivou apstrakcije u platformski specifične modele koji se onda mogu koristiti za automatsko generisanje koda [Sendall03]. U kontekstu razvoja vođenog modelima transformacije možemo klasifikovati kao transformacije između modela (*eng. model-to-model M2M*) i transformacije modela u kôd programskog jezika (*eng. model-to-text M2T*). Editor transformacija omogućava konstrukciju i modifikaciju transformacija, dok repozitorujum predstavlja skladište definisanih transformacija. U opštem slučaju, kôd se može posmatrati kao model, ali se često čuva u tekstualnom obliku koji nije "čitljiv" od strane alata koji rade sa specifičnim formatima modela. Zbog toga postoji parser kôda koji pretvara tekstualni format u odgovarajući format modela, dok generator čita kôd iz repozitorijuma modela u pretvara u tekstualni oblik. Iako modelski zasnovan pristup nastoji da u potpunosti podigne nivo apstrakcije razvoja i potisne aktivnosti vezane za tradicionalno programiranje, u praksi je ovo još uvek delimično realizovano. Aktivnosti vezane za programski kôd jesu stavljene u drugi plan, ali nisu u potpunosti prevaziđene.



Slika 3.2. *Opšti pogled na funkcije i alate modelski zasnovanog razvojnog okruženja.*

Sa obzirom na značaj i složenost transformacija u modelski zasnovanom pristupu, njima je u radu posvećena posebna pažnja. U poglavlju 7 su razmatrane mogućnosti proširenja postojećih razvojnih alata koji pomoću transformacija modela mogu da automatizuju deo razvojnog procesa. Na taj način smo istražili mogućnosti unapređenja razvoja kontekstno-osetljivih korisničkih interfejsa implementacijom odgovarajućih modela i sistema transformacija između njih.

3.2. *Arhitektura predloženog rešenja*

Pre nego što pređemo na detalje realizacije predloženog sistema, opisaćemo primenjena načela i smernice koje su upravljale procesom razvoja rešenja. Drugim rečima, opisaćemo arhitekturu predloženog rešenja. Arhitektura nekog sistema opisuje njegovu strukturu pomoću koncepata aktivnih modula, mehanizama koji omogućuju interakciju između ovih modula i skupa pravila koji upravljaju tom interakcijom [IEEE00]. Arhitektura opisuje najznačajnije elemente i definiše jasnu viziju čitavog sistema, što je neophodno da bi se kontrolisao njegov razvoj.

Prilikom početnog definisanja i izrade arhitekture predloženog rešenja vodili smo se i sledećim principima:

- Arhitektura se mora oslanjati na postojeće modele dizajna korisničkog interfejsa koji su uspešno testirani i našli primenu u praksi.
- Arhitektura mora integrisati koncepte interakcije čoveka i računara u postojeće opšteprihvaćene modele softverskog inženjerstva.
- Arhitektura mora biti fleksibilna u pogledu razdvajanja logike interne funkcionalnosti sistema i logike korisničkog interfejsa.
- Arhitektura mora biti opisana UML rečnikom na sintaksnom i semantičkom nivou, tj. elementi arhitekture moraju biti u skladu sa jezikom UML i njegovim ugrađenim mehanizmima.

U ovom delu ćemo ukratko opisati arhitekturu upravljanu modelima pomoću koje smo organizovali razvoj predloženih elementa rešenja. Takođe, ukratko ćemo opisati faze u razvoju rešenja i korišćene tehnologije.

3.2.1. Arhitektura vođena modelima

Zbog očekivane složenosti, predloženo rešenje je organizovano pomoću arhitekture upravljane modelima koja je u poslednjoj dekadi našla široku primenu u razvoju složenih softverskih sistema [France07]. Sa obzirom da smo koncepte inženjerstva vođenog modelima opisali u prethodnom poglavlju, ovde razmatramo arhitekturu vođenu modelima u kontekstu predloženog pristupa razvoju kontekstno-osetljivih korisničkih interfejsa. Arhitektura upravljana modelima definiše pristup specifikaciji sistema zasnovan na korišćenju hijerarhijski organizovanih formalnih modela [Kleppe03]. Hijerarhijsko organizovanje koncepata i modela u više nivoa je od posebnog značaja imajući u vidu da je razvoj kontekstno-osetljivih korisničkih interfejsa složen proces koji uključuje modelovanje velikog

broja elemenata na različitim nivoima apstrakcije. Sa stanovišta arhitekture upravljane modelima, model kontekstno-osetljive interakcije između čoveka i računara je smešten na nivou metamodela (tabela 3.1).

Tabela 3.1. *Mesto modela kontekstno-osetljive interakcije u OMG MDA.*

OMG MDA Nivo	**Arhitektura modela kontekstno-osetljive interakcije između čoveka i računara**
M3 - Meta-metamodel	*Meta Object Facilities* (MOF)
M2 – Metamodel	**Model kontekstno-osetljive interakcije između čoveka i računara**
M1 - Model	Modeli – instance odgovarajućih metamodela
M0 - Objekti, podaci	Instance modela

Organizovanje modela u više nivoa može da donese niz prednosti. Na primer, koristeći koncepte sa različitih nivoa apstrakcije moguće je definisati jedan platformski nezavisan model (PNM) i jedan ili više platformski-specifičnih modela (PSM). Svaki od PSM modela može da opiše kako se osnovni PNM realizuje na različitim platformama, odnosno da definiše preslikavanja platformski nezavisnih koncepata u primitive dostupne na određenoj implementacionoj platformi. Na ovaj način PNM ne mora da razmatra specifičnosti različitih implementacionih tehnologija, pa nije neophodno da se proces modelovanja aplikacije ponavlja svaki put kada se vrši implementacija rešenja na drugoj platformi. Drugim rečima, u MDA akcenat je na razvoju PNM, dok se potrebni PSM dobijaju transformacijom iz odgovarajućeg PNM. Definisanje transformacija je složen postupak. Sa druge strane, jednom definisana transformacija može biti korišćena u razvoju većeg broja različitih sistema. Na ovaj način se povećava produktivnost razvoja na dva načina. Pre svega, PNM dizajneri ne moraju razmatrati detalje vezane za konkretnu platformu prilikom izrade modela, dok su PSM i delovi kôda automatski generisani izvršavanjem transformacija. Pored toga, akcenat u radu projektanata je na poslovnoj logici, bez detalja implementacije, što dalje rezultuje sistemom koji je bolje prilagođen zahtevima krajnjih korisnika.

Kada govorimo o nivoima modela, odnosno o modelima i metamodelima, bitno je da razlikujemo dve dimenzije instanciranja [Atkinson03]:

- Lingvističko instanciranje i
- Ontološko instanciranje.

U lingvističkom instanciranju, koncepti jednog modela su opisani konceptima nekog metamodela. Drugim rečima, u lingvističkom instanciranju, određeni metamodel definiše jezik kojim će se opisati neki modeli. Ova dimenzija instanciranja je dominantna u OMG MDA standardu. Ovde su jasno definisane granice nivoa apstrakcije tako da koncepti jednog nivoa (npr. M0) predstavljaju instance nivoa iznad (npr. M1).

Sa druge strane, ontološko instanciranje je vezano za definisanje određenog domena. U ovom slučaju oba koncepta se opisuju istim jezikom, odnosno istim metamodelom. To praktično znači da se nalaze

na istom nivou apstrakcije. Na primer, UML objekat je ontološka instanca UML klase, ali su i UML klasa i UML objekat lingvističke instance odgovarajućih MOF klasa.

3.2.2. Razvoj predloženog rešenja i korišćene tehnologije

Razvoj predloženog rešenja odvijao se kroz više faza:

- Prikupljanje i analiza relevantnih rezultata i radova,
- Istraživanje i razvoj modela kontekstno-osetljive interakcije između čoveka i računara,
- Proširenja jezika UML za modelovanje kontekstno-osetljivih korisničkih interfejsa,
- Proširenje *Unified* razvojnog procesa,
- Proširenja alata za razvoj,
- Primena razvijenih proširenja na odabranim primerima.

Iako su navedene faze prikazane hronološki, važno je napomenuti da se razvoj rešenja odvija iterativno i inkrementalno. Na primer, model koji smo naznačili kao prvu fazu u razvoju, se po potrebi dorađivao u toku ostalih faza, kada se uoče nepotpunosti ili nepraktičnost nekih rešenja u modelu. Sa druge strane, na osnovu izmena modela korigovali smo izrađene delove sistema.

Tabela 3.2 daje sažet pregled ključnih tehnologija i alata korišćenih u pojedinim fazama razvoja predloženog rešenja. Sa obzirom na opredeljenje za korišćenje arhitekture upravljane modelima, model kontekstno-osetljive interakcije između čoveka i računara razvijen je korišćenjem ECore jezika. ECore predstavlja verziju MOF jezika, gde postoje male razlike u pogledu imenovanja određenih koncepata. ECore se prvenstveno koristi za opis metamodela i ontologija u okviru EMF platform. Za grafički prikaz modela korišćena je UML notacija i standardan skup UML dijagrama. Za izradu modela i transformacija koristili smo EMT (*eng. Eclipse Modeling Tools)* alat. EMT objedinjuje skup podalata koji pokrivaju čitav proces razvoja upravljanog modelima, od kreiranja i provere validnosti modela, preko tehnologija za transformacije na različitim nivoima apstrakcije, do editora generisanog teksta programa. Iako je ECore podrazumevani jezik za pisanje metamodela, EMT implementira i izvorni MOF standard. Na ovaj način projektanti na raspolaganju imaju oba jezika za kreiranje metamodela. Osim toga, mehanizmi za preslikavanje između metamodela napisanih na ovim jezicima su dobro definisani. Alati za transformacije su klasifikovani u dve grupe u zavisnosti od nivoa apstrakcije modela sa kojima rade. Na taj način postoje M2M (*eng. model-to-model*) alaie i M2T (*eng. model-to-text*) alati. Kod M2M alata se izdvaja ATL jezik sa kojim se može raditi kao sa klasičnim programskim jezikom (na primer, debagovanje i *intellisense* podrška) što ga čini veoma pogodnim za razvoj. Jezik integriše OCL (*eng. Object Constraint Language*) sintaksu što omogućava validaciju modela i kreiranje složenih upita nad izvornim modelima. Od M2T alata izdvajaju se JET (*eng. Java Emitter Templates*), Accelleo i XText. EMT predstavlja modularno, fleksibilno i proširivo okruženje [Leveque09]. Zahvaljujući *plug-in* strukturi EMT može raditi sa izvornim i serijalizovanim (XMI) formatima modela drugih CASE alata kao što su *Poseidon, Rational Rose i Enterprise Architect*. Važno je napomenuti da je XMI podrazumevani format modela u okviru EMT. EMT je razvijen nad Java platformom i može se koristiti i kao Java razvojno okruženje.

Tabela 3.2. *Tehnologije i alati korišćeni u pojedinim fazama razvoja predloženog rešenja.*

Faze	Korišćene tehnologije	Korišćeni alati
Razvoj modela kontekstno-osetljive interakcije između čoveka i računara	ECore (MOF) kao jezik za opis metamodela UML kao notacija za opis modela XMI za skladištenje i razmenu modela	Eclipse Modeling Tools, Protege
Razvoj proširenja jezika UML za modelovanje kontekstno-osetljivih korisničkih interfejsa *Integracija proširenja jezika UML u Unified proces*	UML mehanizmi proširenja	Eclipse Modeling Tools
Razvoj alata za izradu korisničkih interfejsa	UML, ATL, XMI, Java	Eclipse Modeling Tools, M2M (ATL) M2T (ATL, JET, Accelleo)
Primena razvijenih proširenja na odabranim primerima	Predložena UML proširenja, ATL, XMI, Java, Flex (MXML, ActionScript)	Eclipse Modeling Tools, M2M (ATL) M2T (ATL, JET, Accelleo)

3.3. *Rezime poglavlja*

U ovom poglavlju je izložena osnovna ideja predloženog pristupa, kao i primenjena načela i smernice koje su upravljale procesom njegovog razvoja. Predloženi pristup se zasniva na upotrebi modela kontekstno-osetljive interakcije između čoveka i računara i rezultat je primene interdisciplinarnog pristupa i objedinjavanja znanja iz više naučnih oblasti od interesa za kontekstno-osetljivu interakciju između čoveka i računara. Model kontekstno-osetljive interakcije predstavlja jedinstven i formalan opis osnovnih koncepata i relacija između faktora od interesa za razvoj kontekstno-osetljivog korisničkog interfejsa. Korišćenjem modela kontekstno-osetljive interakcije moguće je na različite načine unaprediti razvoj kontekstno-osetljivih korisničkih interfejsa. Model može da služi kao baza znanja (ontologija) ili kao metamodel za proširenja modelovanja i procesa razvoja korisničkih interfejsa uvođenjem primitiva specifičnih za modelovanje kontekstno-osetljive komunikacije čoveka i računara. Formalan, semantički bogatiji modeli korisničkih interfejsa mogu da se koriste u različite svrhe, počevši od dokumentacije pa do automatizacije dela razvojnog procesa i transformacija korisničkih interfejsa. Pored toga, model kontekstno-osetljive interakcije može koristiti i kao osnova za kolaborativni razvoj modela znanja o fenomenima u oblasti interakcije između čoveka i računara. Organizovanje rešenja u skladu sa principima arhitekture upravljane modelima može da obezbedi niz prednosti, jer se ova arhitektura zasniva na korišćenju široko prihvaćenih tehnologija. Korišćenje tehnologija kao što su jezici UML i XML omogućuje da predložena proširenja budu dostupna širem skupu projektanata i programera.

U narednim poglavljima detaljnije će biti opisani pojedini elementi predloženog rešenja: model kontekstno-osetljive interakcije između čoveka i računara; proširenje jezika za modelovanje; proširenje procesa razvoja korisničkih interfejsa; proširenje alata za razvoj softverskih sistema i demonstracija predloženog pristupa na odabranim primerima.

4. Model kontekstno-osetljive interakcije čoveka i računara

Pojam konteksta se može razmatrati sa pozitivističkog i fenomenološkog stanovišta [Dourish04]. Pozitivistički pristup razmatra konteksta sa empirijskog i racionalnog stanovišta, tako što ga opisuje matematičkim modelima kao stabilnu strukturu spoljašnjeg sveta nezavisnu od akcija individua. Fenomenološki pogled je baziran na čoveku koji kreira i menja kontekst u toku interakcije. Na taj način, čovek doživljava kontekst kao svojstvo interakcije, a ne objekata ili drugih ljudi. U opisivanju kontekstno-osetljive komunikacije usvojili smo fenomenološki pristup. Ovakvo početno opredeljenje je važno, jer opisivanje komunikacije ljudi i kontekstno-osetljivih aplikacija zahteva jasnu definiciju konteksta. Model kontekstno-osetljive interakcije čoveka i računara predstavlja sveobuhvatnu i objedinjenu definiciju konteksta. Definisanje ovakvog modela daje formalne opise osnovnih koncepata kontekstno-osetljive komunikacije čoveka i računara. Upotreba ovih koncepata omogućava integraciju konteksta komunikacije čoveka i računara u razvoj korisničkih interfejsa.

U ovom poglavlju detaljnije opisujemo način izrade i strukturu generičkog modela kontekstno-osetljive interakcije između čoveka i računara. Najpre opisujemo razvoj modela kontekstno-osetljive interakcije čoveka i računara sa aspekta metodološkog pristupa kreiranju modela, alata koje smo koristili pri razvoju i korišćene notacije za opis modela. Nakon toga, detaljnije opisujemo model kontekstno-osetljive interakcije. Prvo opisujemo njegovu strukturu, zatim delove modela kao što su okruženje, ljudski faktori, faktori računarskog uređaja i interakcija između čoveka i računara.

4.1. *Razvoj modela kontekstno-osetljive interakcije*

Model kontekstno-osetljive interakcije čoveka i računara je rezultat primene interdisciplinarnog pristupa i formalno opisuje znanja iz više naučnih oblasti od interesa za kontekstno-osetljivu interakciju između čoveka i računara. Na taj način model možemo posmatrati i kao *ontologiju* znanja potrebnih za razvoj kontekstno-osetljivih korisničkih interfejsa, jer sadrži pregled najznačajnijih rezultata i saznanja iz relevantnih predmetnih oblasti.

U razvoju modela kontekstno-osetljive interakcije koristili smo različite pristupe pri izradi različitih delova. Ovi pristupi se mogu klasifikovati u pet kategorija [Holsapple02]:

- *Induktivni pristup*, kod kojeg se model razvija posmatranjem, ispitivanjem i analiziranjem specifičnih slučajeva u domenu od interesa. Rezultat razvoja je generalizacija posmatranih slučajeva. Na taj način se kreirani opis može primeniti i za ostale slučajeve istog tipa.

- *Deduktivni pristup*, kod kojeg se usvaja određeni skup opštih principa, koji se proširuju i primenjuju na specifične slučajeve. Ovaj pristup obično podrazumeva filtriranje i pročišćavanje opštih termina tako da se oni prilagode određenom podskupu domena.

- *Sintetički pristup*, kod kojeg se identifikuje osnovni skup modela i ontologija, a delovi ovih ontologija se onda, zajedno sa dodatnim konceptima identifikovanim u domenu od interesa, integrišu u objedinjenu ontologiju. U ovom slučaju rezultujući model objedinjuje već postojeći model. Na taj način je moguće obezbediti veću prihvatljivost modela jer je samo manji deo modela nov.

- *Inspirativni pristup*, u kome projektant započinje razvoj sa premisom o tome zašto je model potreban. Korišćenjem individualnih razmišljanja, kreativnosti, i ličnih pogleda na domen od interesa, projektant nastavlja razvoj modela do ispunjenja prepoznatih ciljeva. Sa jedne strane, ovakav način izrade može da stvori model koji je ograničenje upotrebljivosti, jer kao rezultat jedne osobe može biti nepraktičan i sa ograničenom teoretskom podlogom. Sa druge strane, on može da donese i mnoge nove, jedinstvene ideje u model.

- *Kolaborativni pristup*, u kome je model rezultat zajedničkog napora više osoba ili organizacija. Rezultujući model obično sadrži poglede sa više različitih strana, te je zbog toga veća šansa da će model biti prihvaćen od šire grupe interesenata. Učešće više zainteresovanih strana u izradi modela može da ukloni i mnoge nedostatke u modelu nastale usled jednostranog razmatranja, te da značajno obogati sadržaj modela. Razvoj ovakvog model uglavnom se obavlja iterativno i inkrementalno, tako što se polazi od manje verzije koja se onda proširuje i nadgrađuje.

Korišćenjem sintetičkog pristupa u radu je izvršena sinteza znanja iz odabranih izvora iz oblasti interakcije između čoveka u računara. Takođe, korišćenjem induktivnog pristupa, izvršili smo generalizaciju znanja, na primer pri kreiranju metamodela postojećih implementacionih platformi. Korišćenjem deduktivnog pristupa smo primenili znanja na nove slučajeve. Pored toga, model kontekstno-osetljive interakcije između čoveka i računara sadrži i neke nove elemente koji nisu preuzeti iz postojećih rešenja već su deo naših istraživanja i rezultat su *inspirativnog pristupa*. Na kraju, pri izradi modela konsultovali smo i saradnike, dok su delovi modela publikovani u časopisima i prezentovani na konferencijama, tako da smo dobili bolju povratnu spregu o upotrebljivosti predloženog modela.

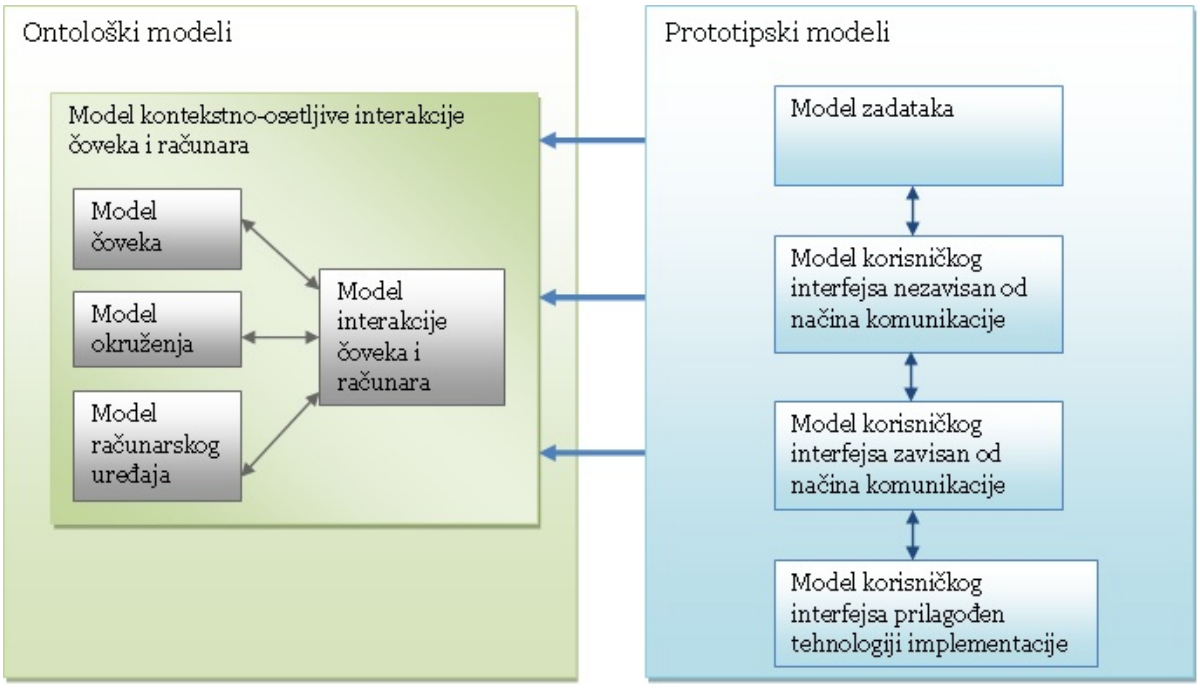
4.2. *Struktura i uloga modela kontekstno-osetljive interakcije*

Model kontekstno-osetljive interakcije između čoveka i računara sadrži formalno opisana znanja o fenomenima iz različitih domena relevantnih za razvoj kontekstno-osetljivih korisničkih interfejsa. Naš cilj je bio da identifikujem osnovni skup koncepata iz svakog od domena i da uspostavimo relacije između ovih koncepata. Drugim rečima, model predstavlja rečnik termina i relacija između njih koji služe kao osnova za razvoj kontekstno-osetljivih korisničkih interfejsa. Praktično govoreći, želeli smo da postignemo sledeće:

- *Precizno definisanje termina i relacija između njih*,

- *Veću izražajnost opisa*, tako što bi definicije termina bile prilagođene oblasti interakcije između čoveka i računara,
- *Koherentnost i interoperabilnost rezultujuće baze znanja*, korišćenjem standardnih i opšte prihvaćenih tehnologija za modelovanje,
- *Skalabilnost modela*, koja treba da omogući definisanje koncepata na različitim nivoima apstrakcije, kao i lako proširivanje i dopunjavanje modela.

Slika 4. 1 daje opšti pogled na strukturu i ulogu modela kontekstno-osetljive interakcije čoveka i računara u razvoju kontekstno-osetljivih korisničkih interfejsa.

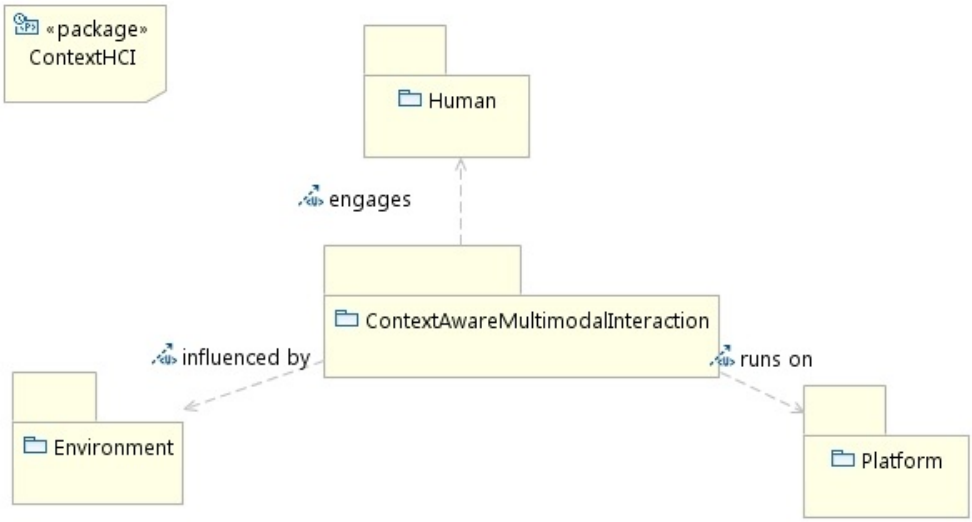


Slika 4.1. *Modeli konteksta interakcije (ontološki modeli) i modeli razvoja korisničkih interfejsa (prototipski modeli).*

Model kontekstno-osetljive interakcije čoveka i računara je struktuiran u skladu sa preporukama CAMELEON referentnog okvira. Kontekst interakcije je dekomponovan u tri dela: čovek, računarska platforma i okruženje u kojem čovek obavlja interaktivne zadatke sa specifiranom platformom. Svaka varijacija ovih delova se može posmatrati kao promena konteksta koja se na određeni način odražava na korisnički interfejs. U skladu sa tim, a radi lakšeg rada i bolje preglednosti, koncepti modela su klasifikovani u četiri podmodela (slika 4.2):

- *Model okruženja* – paket *Environment*, opisuje koncepte okruženja relevatne za interakciju čoveka i računara kao što su prostor, lokacija, atmosferski uslovi okruženja.
- *Model čoveka* – paket *Human*, opisuje koncepte čoveka relevantne za interakciju u obliku struktura i funkcija ljudskog tela.
- *Model računarskog uređaja* – paket *Platform*, opisuje hardversku i softversku platformu sa kojom čovek interaguje.

- *Model interakcije čoveka i računara* – paket *ContextAwareMultimodalInteraction*, opisuje načine komunikacije čoveka sa računarom koja je uslovljena konceptima prethodno navedenih modela. U opisivanju načina komunikacije oslanjamo se na postojeće formalno opisane koncepte višenačinske komunikacije [Obrenovic04].



Slika 4.2. *Osnovne komponente modela kontekstno-osetljive interakcije čoveka i računara.*

Modeli kontekstno-osetljive interakcije čoveka i računara su generičke prirode, drugim rečima nezavisni od domena korišćenja interaktivnog sistema. U tom pogledu predstavljaju ontološku osnovu za kreiranje modela koji će biti korišćeni u razvoju kontekstno-osetljivih korisničkih interfejsa. Ovi modeli su označeni kao prototipski modeli i oni su specifični za interaktivni sistem za dati domen. U procesu razvoja kontekstno-osetljivih korisničkih interfejsa predvideli smo korišćenje modela četiri nivoa apstrakcije. Ovi modeli i njima odgovarajući OMG MDA modeli su prikazani u tabeli 4.1.

Tabela 4.1. *Preslikavanje prototipskih modela na OMG MDA modele.*

Prototipski model	**OMG MDA model**	**Komentar**
Model zadataka	CIM (Computational-Independent Model)	
Model korisničkog interfejsa nezavisan od načina komunikcije (apstraktni korisnički interfejs)	PIM (Platform-Independent Model)	
Model korisničkog interfejsa zavisan od načina komunikcije (konkretni korisnički interfejs)	PSM (Platform-Specific Model) PIM*	*U pogledu načina komunikacije može se smatrati PSM, dok se u pogledu platforme implemenentacije može posmatrati kao PIM.
Model korisničkog interfejsa prilagođen tehnologiji implementacije (konačni korisnički interfejs)	PSM	

Ovde je važno napomenuti da je identifikacija četiri nivoa apstrakcije i uspostavljanje hijerarhije između njih izvršena na osnovu njihove nezavisnosti u odnosu na kontekst u kojem čovek koristi

korisnički intefejs. Drugim rečima, modeli koncepata i zadataka su računski nezavisni, apstraktni korisnički interfejs je nezavisan od načina komunikacije, dok je konkretni korisnički interfejs nezavisan od tehnologije implementacije. Model konačnog korisničkog interfejsa je prilagođen tehnologiji implementacije. Modeli razvoja su ovde prikazani radi boljeg razumevanja mesta i uloge modela kontekstno-osetljive interakcije. Detaljniji opis ovih modela dat je u narednom poglavlju.

Pri izradi modela kontekstno-osetljive interakcije čoveka i računara, pored relevantnih radova iz oblasti, korišćena je i klasifikacija zdravstvenih faktora čoveka (*eng. International Classification of Functioning, Disability and Health - ICF*) [ICF10] propisana od strane Međunarodne Zdravstvene Organizacije. Klasifikacija predstavlja standardizovan jezik i okvir za opis zdravstvenih stanja čoveka. Zdravstvena stanja se opisuju preko zdravstvenih domena koji predstavljaju promene u strukturi i funkcionisanju ljudskog tela u toku različitih aktivnosti čoveka u različitim okruženjima. Tabela 4.2 daje pregled osnovnih kategorija klasifikacije. Sa obzirom da je klasifikacija veoma opsežna, prikazane su samo one kategorije koje smo smatrali relevenatnim za kontekstno-osetljivu interakciju čoveka i računara. Pored toga, nabrojane kategorije imaju dodatnu hijerarhijsku strukturu koja zbog složenosti nije prikazana u tabeli, ali je obuhvaćena odgovarajućim modelima.

Tabela 4.2. *Sažet pregled kategorija ICF klasifikacije.*

Funkcije ljudskog tela	
Funkcije	**Opis**
Mentalne funkcije	Globalne (svesnost, orijentacija i dr.) i specifične (pažnja, percepcija i dr.) mentalne funkcije.
Senzorske funkcije	Funkcije odgovarajućih čula čoveka.
Glasovne i govorne funkcije	Funkcije vezane za kreiranje zvuka i govora.
Neuromuskulatorne funkcije i funkcije pokreta	Funkcije pokreta i pokretljivosti.
Strukture ljudskog tela	
Strukture	**Opis**
Strukture nervnog sistema	
Strukture oka, uha i povezane strukture	
Strukture vezane za glas i govor	
Strukture vezane za pokrete	
Aktivnosti i učešća čoveka	
Aktivnosti i učešća	**Opis**
Učenje i primena znanja	Učenje, primena naučenog, razmišljanje, rešavanje problema, odlučivanje i dr.
Opšti zadaci i zahtevi	Opšti aspekti izvršenja jednog ili više zadataka, na primer alokacija vremenskih, prostornih i materijalnih resursa.
Komunikacija	Opšte i specifične karakteristike komunikacije putem jezika, znakova i simbola.
Mobilnost	Načini promene lokacije i položaja ljudskog tela.
Međuljudske interakcije i relacije	Aktivnosti vezane za različite oblike društveno dozvoljenih međuljudskih interakcija.
Faktori okruženja	
Faktori okruženja	**Opis**
Tehnologija i proizvodi	Prirodni i proizvodi koje je kreirao čovek, oprema i tehnologija u neposrednom okruženju ljudi.
Prirodno okruženje i okruženje koje je kreirao čovek	Elementi i delovi okruženja, kako prirodni, tako i oni koje je modifikovao čovek.

Klasifikacija je organizovana u dve celine. Prva celina se odnosi funkcije i ograničenja funkcija čoveka i korišćena je za formiranje modela čoveka. Sastavljena je iz dva dela:

- Funkcije tela (*eng. Body Functions*) i strukture tela (*eng. Body Structures*) koje opisuju funkcije čoveka (npr. mentalne funkcije, senzorske funkcije) i strukture čoveka koje ove funkcije realizuju (npr. nervni sistem, uho, oko i povezane stukture).

- Aktivnosti i učešća čoveka (*eng. Activities and Participation*) pokrivaju funkcionisanje čoveka sa individualnog (npr. komunikacija, Mobilnost) i društvenog stanovišta (npr. međuljudske interakcije i relacije).

Druga celina pokriva faktore konteksta (*eng. Contextual Factors*). Koncepti ove celine su korišćeni u opisivanju modela okruženja i računarskog uređaka. Celina se sastoji iz dva dela:

- Faktori okruženja (*eng. Environmental Factors*) su organizovani od neposrednog okruženja čoveka (npr. proizvodi i tehnologija) do opštijih oblika okruženja (npr. servisi i sistemi).

- Personalni faktori (*eng. Personal Factors*) u koje spadaju pol, starost, rasa, stil života, navike i dr.

Svaka od komponenti se sastoji od različitih domena (npr. domen senzorskih funkcija pripada komponenti funkcija i struktura ljudskog tela). Unutar svakog domena, kategorije su jedinice klasifikacije i uređene su hijerarhijski (tako npr. domen senzorskih funkcija ima ugnježdene kategorije kao što su funkcije vida, kvalitet vida i osetljivost na svetlo). Klasifikacija je uređena tako da omogućava određivanje zdravstvenog stanja čoveka. Ovo je realizovano izborom odgovarajućeg kôda kategorije i dodavanjem kvalifikatora koji predstavljaju numeričke oznake i označavaju stepen normalnog funkcionisanja ili ograničenja u funkcionisanju date kategorije. Pored toga, kvalifikatori mogu određivati i stepen u kojem faktor okruženja predstavlja stimulans ili ograničenje. Na primer, oznaka b210.4 označava potpuno ograničenje funkcije vida: prefiks b identifikuje komponentu funkcija tela, kôd "210" označava kategoriju funkcije vida domena senzorskih funkcija, kôd ".4" označava vrednost potpunog ograničenja funkcija vida. Kako ICF sadrži nešto vise od 1400 kategorija, predložen je i skraćeni pregled taksonomije (*eng. ICF Checklist*) kako bi ona bila efikasnije korišćena u kliničkoj praksi. Ovaj pregled predstavlja izbor 125 ICF kategorija koje se najčešće javljaju u kliničkoj praksi i realizovan je u formi upitnika čijim se popunjavanjem dobija zdravstveni profil čoveka.

U pogledu formalizacije, ICF klasifikacija u pojedinim segmentima ispoljava nedostatke poput nedovoljno jasnih relacija između vrsta određenih aktivnosti i njihovih svojstava, kao i preteranog uprošćavanja ili usložnjavanja pojedinih delova. Ovo dovodi do pojave suviše pojednostavljenih ili nekompletnih klasifikacija. I pored toga, ona je našla široku primenu u medicinskoj praksi definisanjem opsežnog rečnika za opise zdravstvenih profila čoveka. Uzimajući ovo u obzir, opise nekih koncepata smo proširivali komplementarnim znanjima iz drugih izvora i istovremeno prilagođavali mehanizmima UML jezika kako bi oni bili upotrebljivi u razvoju softvera. Formalizovan je deo klasifikacije u obliku UML profila. U medicini klasifikacija je našla širu primenu i u oblasti rehabilitacione psihologije [Bruyère05]. U oblasti interakcije čoveka i računara korišćena je u

disciplinama koje se bave dizajnom sofverskih sistema za različite kategorije ljudi [Wobbrock11], kao i za osobe sa posebnim potrebama [Chittaro11].

U odeljcima koji slede detaljnije će biti opisani elementi modela kontekstno-osetljive interakcije čoveka i računara. Poseban naglasak je dat na modelima čoveka koji su opisani u narednom odeljku. Nakon toga ćemo govoriti o modelima koji opisuju okruženje i računarske uređaje. Na kraju dajemo opis modela interakcije čoveka i računara. Zbog izuzetno velikog koncepata u modelu, detaljnije su opisani samo neki od ključnih delova modela na koje ćemo se pozivati u kasnijim poglavljima.

4.2.1. Modelovanje čoveka

Posmatrajući istorijat metodologije razvoja softverskih sistema možemo uočiti da je razvoj prvenstveno vođen tehnologijom, sa pretpostavkom da se korisnik može prilagoditi svakom proizvodu. Posledica ovakvog pristupa je prilagođavanje korisnika sistemu kroz obuku i praksu u korišćenju. Dizajn orijentisan ka čoveku (*eng. Human-centered design*) promoviše pristup u kojem se u razvoju softverskih sistema polazi od modelovanja prirodnog ponašanja čoveka, uključujući i ograničenja u sposobnostima učenja i vršenja radnji, kako bi se došlo do sistema koji su intuitivniji za korišćenje, jednostavniji za učenje i manje skloni greškama [Oviatt06].

Kada govorimo o modelovanju korisnika-čoveka, možemo uočiti dva pristupa:

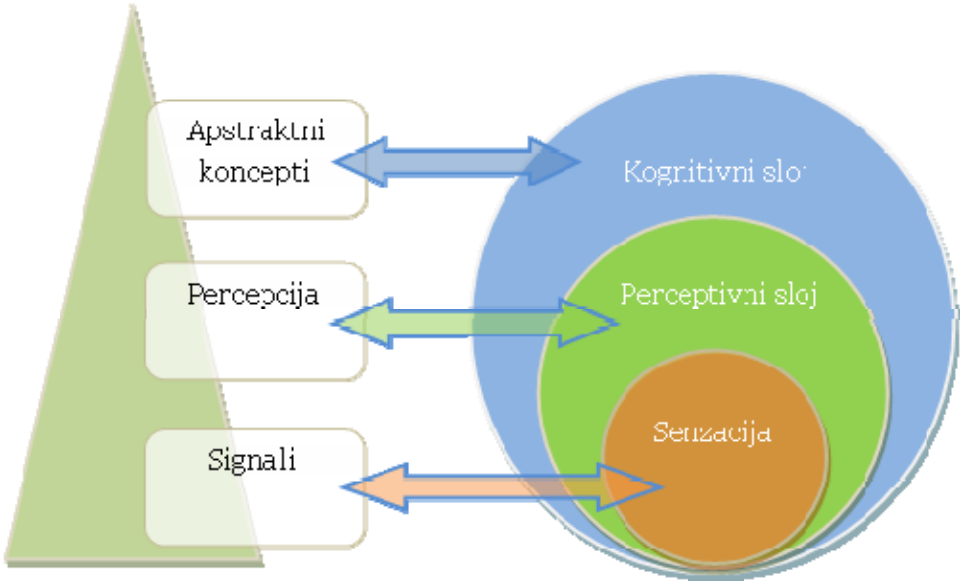
- Modelovanje korisnika u edukativnim sistemima i sistemima adaptivne hipermedije kod kojih se polazi od znanja, stečenog iskustva i interesovanja korisnika vezanih za određenu oblast.
- Modelovanje korisnika u oblasti interakcije čoveka i računara gde se u obzir uzimaju unutrašnji (anatomija i fiziologija čoveka) i spoljašnji (zadaci, potrebe, mogućnosti) faktori čoveka kao biološkog bića.

Modeli korisnika predstavljaju sastavni deo adaptivnih softverskih sistema [Brusilovsky07]. U kontekstu ovih sistema, pod modelom korisnika se podrazumeva reprezentacija informacija o korisniku na osnovu koje adaptivni sistem vrši adaptaciju, tj. obezbeđuje različito ponašanje za različite korisnike. Na primer, kada korisnik vrši pretragu relevantnih informacija, sistem adaptivno selektuje i određuje prioritet informacija. Prilikom otvaranja određene stranice, sistem vrši adaptaciju prikaza sadržaja. Da bi kreirao i održavao ažurne modele korisnika sistem prikuplja podatke iz različitih izvora koji mogu obuhvatiti implicitno praćenje interakcije korisnika i eksplicitan zahtev korisniku za unosom podataka. Skup ovakvih aktivnosti čini proces modelovanja korisnika. Model korisnika predstavlja osnovu adaptacije sistema. Količina i priroda informacija sadržanih u modelu korisnika određeni su vrstom adaptacije koju sistem treba da obezbedi. Na ovaj način modeli korisnika se mogu klasifikovati na osnovu sledećih kriterijuma:

- Priroda modelovanih informacija,
- Struktura i reprezentacija modela i
- Mehanizmi održavanja modela.

U pogledu prethodno navedenih kriterijuma, modeli korisnika predstavljaju sastavni deo sistema za pribavljanje informacija (*eng. Information Retrieval - IR*) i inteligentnih tutorskih sistema (*eng. Intelligent Tutoring Systems - ITS*) [Brusilovsky07]. IR sistemi i sistemi filtriranja se generalno bave problemom pronalaženja dokumenata relevantnih sa stanovišta potreba korisnika. Model korisnika koji se koristi u ovakvim sistemima poznat je pod nazivom *profil korisnika* (*eng. user profile*) i predstavlja interesovanja korisnika izražena u terminima ključnih reči ili koncepata. ITS sistemi se bave aktivnostima i sadržajima vezanim za edukaciju. Oni vrše prilagođavanje edukativnog sadržaja nivou znanja korisnika. Zbog toga je model korisnika u ovim sistemima poznat pod nazivom model studenta (*eng. student model*) i predstavlja znanje korisnika iz određenog domena koje je u relaciji sa ekspertnim znanjem domena. Ovo znanje je struktuirano na različite način. Najzastupljeniji oblik striktuiranog znanja predstavlja tzv. *overlay model* kod kojeg je znanje pojedinačnog korisnika predstavljeno kao podskup domenskog znanja koje predstavlja znanje eksperta iz domena koji opisuje. Samo znanje može biti dato u konceptualnom obliku (činjenice i relacije između njih) i proceduralnom obliku (veštine rešavanja problema). Za svaki fragment domenskog znanja kreira se i čuva procena nivoa znanja pojedinačnog korisnika za dati fragment. Ova procena može biti kvalitativne prirode (npr. dobar, prosečan, ispod proseka) ili kvantitativne prirode (stepen poznavanja određene teme izražen u procentima). Osim znanja, kao osnovnog predmeta modelovanja, sistemi adaptivne hipermedije modeluju i interese čoveka, ciljeve i zadatke čoveka, iskustvo čoveka i osobine čovekove ličnosti. Uopšteno govoreći, predmet modelovanja je određen domenom i načinom korišćenja konkretnog sistema.

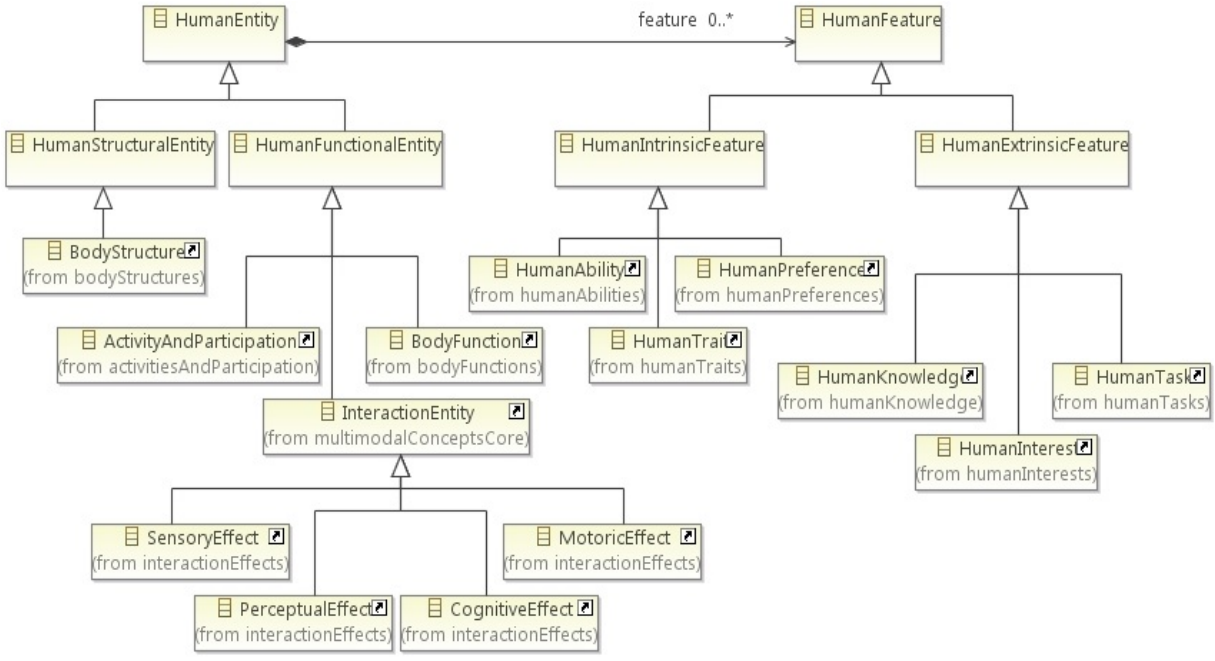
U oblasti interakcije čoveka i računara, čovek je modelovan sa aspekta njegovih karakteristika kao biološkog bića. Istraživači su proučavali tokove informacija čoveka izvodeći eksperimente u kojima je čovek izvršavao specijalizovane, kontrolisane zadatke. Cilj ovih eksperimenata je bio da se ustanove mehanizmi koji obezbeđuju tokove informacija kod čoveka. Široko prihvaćen model percepcije čoveka predstavlja model koji opisuje čoveka pomoću skupa motoričkih, senzorskih, percepcijskih i kognitivnih kapaciteta [Card83] (slika 4.3).



Slika 4.3. *Senzorsko-percepcijski-kognitivni model interakcije čoveka sa okruženjem.*

Na osnovnom nivou se nalazi senzacija koja predstavlja kombinaciju biohemijskih i neuroloških događaja koje generišu draži registrovane od strane odgovarajućih čula čoveka. Ovi događaji mogu biti generisani i kao posledica radnji mehaničkih aktuatora čoveka. Na perceptivnom nivou dobijeni signali se klasifikuju, analiziraju i interpretiraju. Kognitivni nivo vrši prijem i semantičku obradu percepcijskih signala. Na ovom nivou se nalaze kognitivne funkcije visokog nivoa kao što su mišljenje, korišćenje stečenog iskustva, rešavanje problema, donošenje odluka i dr. Kao rezultat ovih funkcija mogu nastati komande koje se šalju aktuatorima čoveka.

Prilikom kreiranja modela čoveka nastojali smo da objedinimo oba prethodno opisana pristupa. Drugim rečima, formirali smo model čoveka koji objedinjuje osobine korisnika sa stanovišta interakcije čoveka i računara sa jedne strane, i korišćenja računara u određenoj oblasti (inteligentni tutorski sistemi, adaptivna hipermedija i dr.) sa druge strane. Osnovni koncepti modela čoveka i njihove međusobne relacije prikazani su na slici 4.4. Koncepti koji opisuju anatomiju čoveka (*BodyStructure*) i fiziologiju čoveka (*BodyStructure*), kao i aktivnosti vezane za primenu anatomskih i fizioloških mehanizama (*ActivityAndParticipation*) predstavljeni su kao entiteti čoveka (*HumanEntity*). Entiteti su klasifikovani kao strukturni i funkcionalni. U okviru hijerarhije funkcionalnoh entiteta uvedeni su i koncepti koji opisuju senzorske, percepcijske, kognitivne i motoričke osobine interakcije čoveka sa računarom. Svojstva čoveka (*HumanFeature*) opisuju stečene osobine, tj. čovekove psihološke osobine i karakteristike vezane za specifičan domen korišćenja računara. U zavisnosti od toga da li su vezana za psihologiju čoveka ili određenu oblast, svojstva čoveka su klasifikovana kao unutrašnja (*HumanIntrinsicFeature*) ili spoljašnja (*HumanExtrinsicFeature*). Unutrašnja svojstva čoveka su dalje klasifikovana kao mogućnosti (*HumanAbility*), naklonjenosti (*HumanPreference*) i crte čoveka (*HumanTrait*). Sa druge strane, spoljašnja svojstva čoveka su klasifikovana kao znanje (*HumanKnowledge*), interesovanja (*HumanInterest*) i zadaci (*HumanTask*).



Slika 4.4. *Osnovni koncepti modela čoveka i relacije između njih.*

4.2.1.1. Entiteti čoveka

Ljudsko telo, odnosno njegov motorni i senzorski sistemi predstavljaju osnovni ljudski interfejs ka okruženju. Senzorski sistem obezbeđuje čoveku prozor u svet preko kojeg on može da održava svoju sliku realnog sveta. Sa druge strane, motorni sistem obezbeđuje aktuatore koji omogućuju čoveku delovanje i fizičku interakciju sa okruženjem. Anatomije ljudskih senzorskih, percepcijskih, kognitivnih i motornih sistema su usko povezane. Sa stanovišta korisničkih interfejsa, poznavanje ljudske anatomije i fiziologije može da omogući bolje prilagođavanje korisničkih interfejsa i uređaja koji se koriste u interakciji između čoveka i računara ograničenjima čoveka. Još od ranih dana je istraživanje interakcije između čoveka i računara uključivalo istraživanje osobina ljudske motorike i percepcije, kao što to, na primer, čini Fittsov zakon [MacKenzie92]. Biomedicinske aplikacije istražuju mnoge elemente ljudske anatomije i fiziologije u cilju unapređenja medicinskih procedura. Pored njih, specifične vrste korisničkih interfejsa, poput elektrofizioloških i neuralnih interfejsa, zasnivaju se na znanjima o ljudskoj anatomiji i fiziologiji.

Opisi ljudskih faktora predstavljaju najsloženiji deo modela, sa najvećim brojem koncepata i relacija. Koncepte koji opisuju anatomiju, fiziologiju i aktivnosti čoveka su objedinjeni modelom entiteta čoveka koji je struktuiran po osnovnim paketima:

- Strukture ljudskog tela (*BodyStructures*),
- Funkcije ljudskog tela (*BodyFunctions*),
- Aktivnosti čoveka (*ActivitiesAndParticipation*).

Model entiteta je kreiran na osnovu ICF klasifikacije. Pri tome su koncepti klasifikacije dopunjeni i prilagođeni mehanizmima UML jezika kako bi bili upotrebljivi u razvoju kontekstno-osetljivih korisničkih interfejsa. Funkcije tela (*BodyFunctions*) predstavljaju fiziološke funkcije sistema organa čoveka. Strukture tela (*BodyStructures*) predstavljaju anatomske delove tela kao što su organi i njihove komponente. Aktivnost (*Activity*) predstavlja izvršavanje zadatka ili akcije pojedinca. Koncepti vezani za anatomiju ljudskog tela biće korišćeni za opis čovekovih strukturnih mehanizama u razvoju korisničkih interfejsa. Koncepti koji opisuju fiziologiju čoveka će omogućiti opise čovekovih mehanizama ponašanja (neki od njih imaju i strukturnu prirodu). Koncepti anatomije i fiziologije su međusobno usko povezani i uslovljeni. Koncepti aktivnosti vezani za anatomiju i fiziologiju čoveka će biti korišćeni za opise ponašanja čoveka na specifičan način, tj. korišćenjem UML mehanizama modelovanja stanja i aktivnosti.

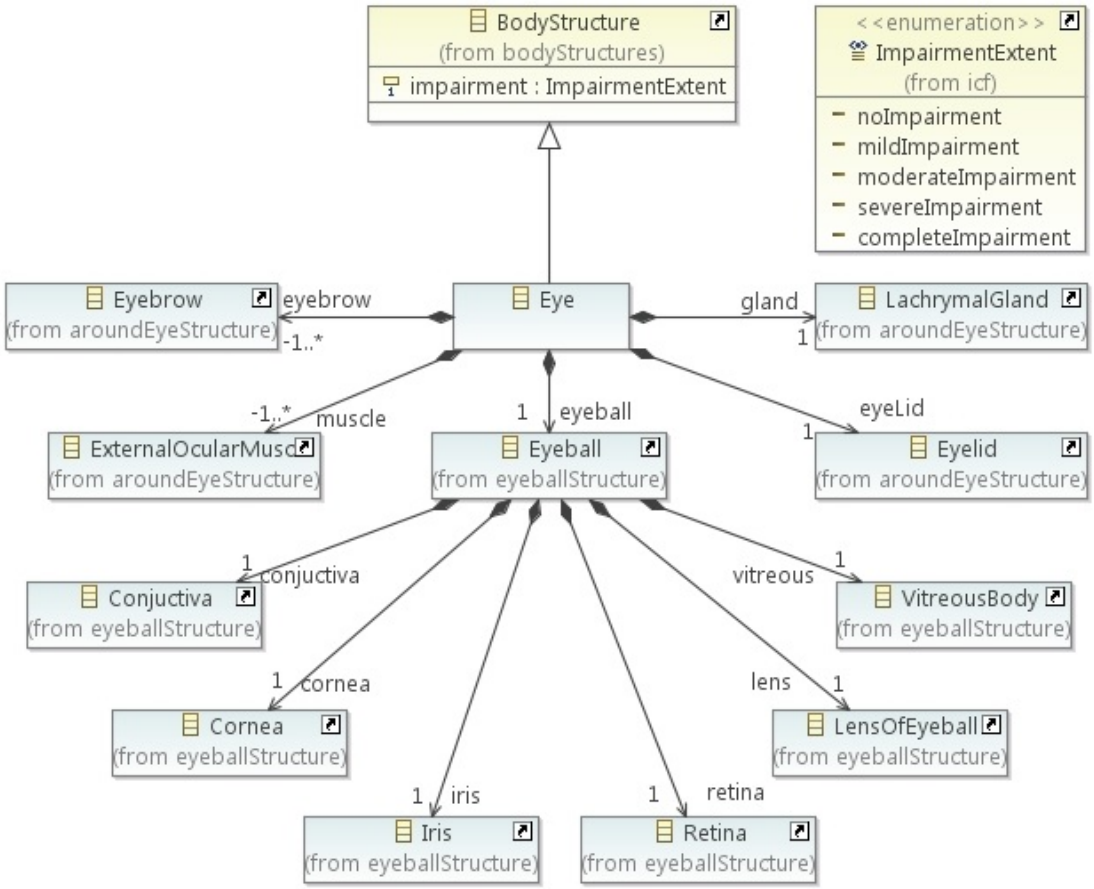
4.2.1.1.1. Strukture čoveka

Model struktura čoveka (*BodyStructure*) opisuje ljudsku anatomiju i sastoji se od sledećih paketa:

- Strukture nervnog sistema (*StructuresOfTheNervousSystem*),
- Strukture oka, uha i povezane strukture (*EyeEarAndRelatedStructures*),

- Strukture kardiovaskularnog i respiratornog sistema (*StructuresOfTheCardiovascularAndRespiratorySystems*),
- Strukture glasa i govora (*StructuresInvolvedInVoiceAndSpeech*),
- Strukture pokreta (*StructuresRelatedToMovement*).

Svaki od paketa je dalje struktuiran u podpakete, na primer paket struktura pokreta sastoji od struktura glave i vrata, strukture ramena, strukture karlice i struktura gornjih i donjih ekstremiteta čoveka. Modeli anatomije čoveka predstavljaju osnovu za formiranje ostalih modela. Strukturna specifikacija čoveka čini osnovu mehanizama ponašanja i funkcija višega nivoa, kao što su oblici interakcije u kontekstu obavljanja specifičnih aktivnosti. Slika 4.5 daje prikaz strukture čovekovog oka. Model predviđa i definisanje generičkog kvalifikatora stepena devijacije određenog organa čoveka (*ImpairmentExtent*). Uvođenje ovog svojstva omogućava preciznije određivanje mogućnosti i kvaliteta interakcije čoveka sa računarom, naročito u oblasti dizajna za osobe sa posebnim potrebama.



Slika 4.5. *Struktura čula vida čoveka.*

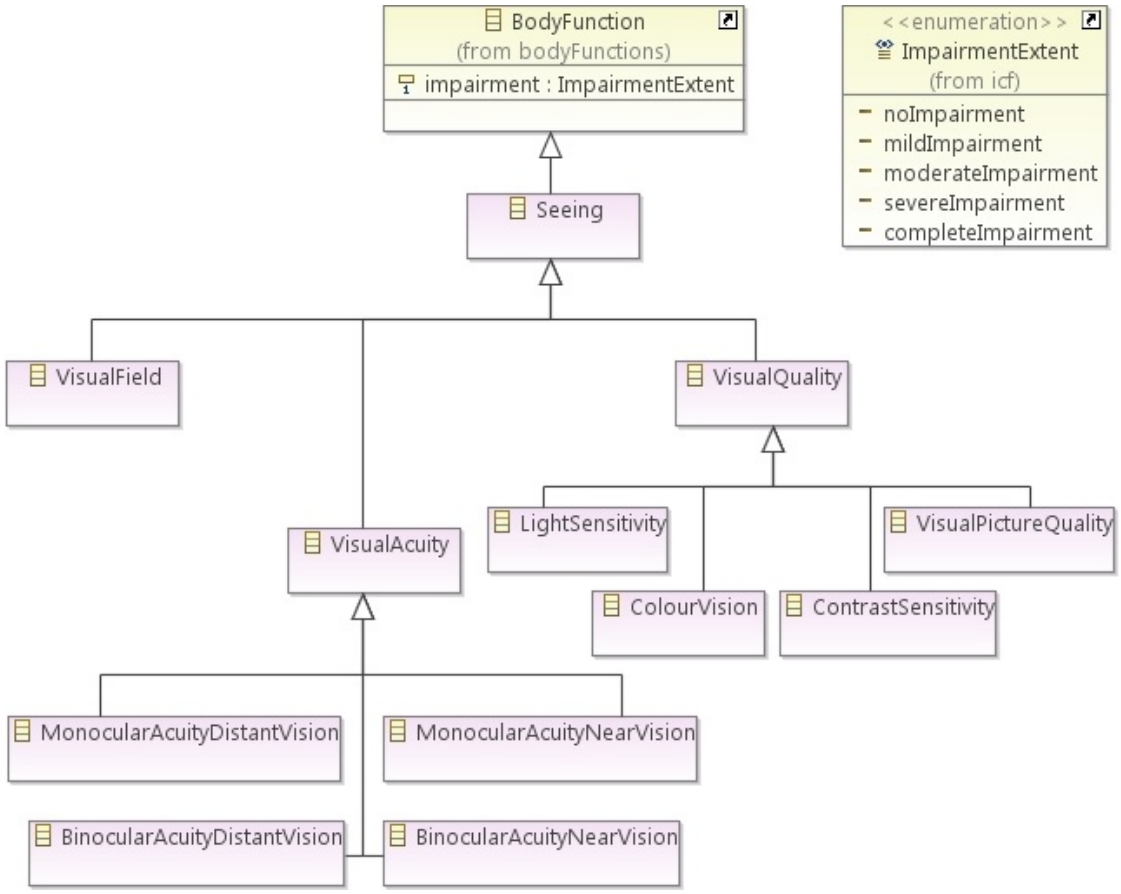
4.2.1.1.2. Funkcije čoveka

Model funkcija čoveka opisuje ljudske fiziološke mehanizme i sastoji se od sledećih paketa:

- Mentalne funkcije (*MentalFunctions*) - opisuje funkcije mozga čoveka koje mogu biti globalne (svest, orijentacija i dr.) i specifične (percepcija, kognitivne funkcije višeg nivoa i dr.).

- Senzorske funkcije (*SensoryFunctions*) - predstavljaju aktivnosti odgovarajućih čula čoveka kao što su vid, sluh, dodir i dr.
- Funkcije kardiovaskularnog i respiratirnog sistema - (*FunctionsOfTheCardiovascularAndRespiratorySystem*s) su vezane za rad srca i pluća.
- Glasovne i govorne funkcije (*VoiceAndSpeechFunctions*) - opisuju produkciju zvukova i govora,
- Neuromuskulatorne funkcije i funkcije pokreta (*NeuromusculoskeletalAndMovementRelatedFunction*s) - obuhvataju mobilnost i kretanje, tako što definišu funkcije odgovarajućih struktura, tj. zglobova, kostiju, mišića i refleksa.

Svaki od paketa se sastoji od odgovarajućih podpaketa. Funkcije čoveka vrše odgovarajuće anatomske strukture. Promena određene strukture usloviće promene u njoj pridruženim fiziološkim funkcijama i obrnuto. Analogno strukturama čoveka, i kod funkcija je uveden generički kvalifikator stepena devijacije fiziološke funkcije čoveka. Na slici 4.6 je prikazana klasifikacija funkcija vida čoveka.



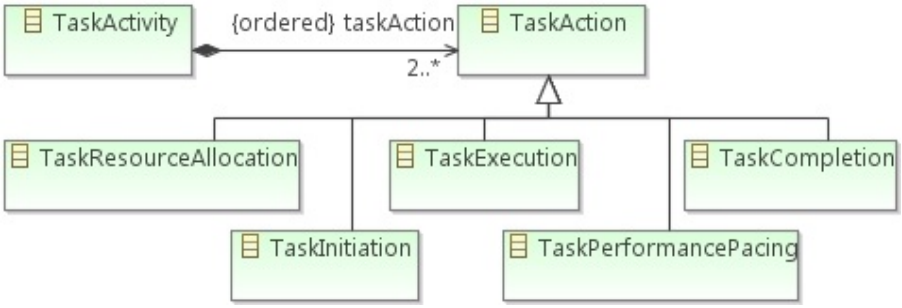
Slika 4.6. *Funkcije vida čoveka.*

4.2.1.1.3. Aktivnosti čoveka

Model aktivnosti čoveka opisuje radnje i aktivnosti čoveka u kontekstu njegovog okruženja. Model se sastoji od sledećih paketa:

- Učenje i primena znanja (*LearningAndApplyingKnowledge*) - obuhvata aktivnosti kao što su učenje, primena naučenog, razmišljanje, rešavanje problema, odlučivanje i dr.
- Opšti zadaci i zahtevi (*GeneralTasksAndDemands*) - opisuje aspekte izvršenja jednog ili više zadataka, na primer alokacija vremenskih, prostornih i materijalnih resursa i dr.
- Komunikacija (*Communication*) - sadrži opšte i specifične karakteristike komunikacije putem jezika, znakova i simbola, poput prijema i produkcije poruka, korišćenja tehnika komunikacije i dr.
- Mobilnost (*Mobility*) – integriše različite načine i mehanizme promene lokacije i položaja ljudskog tela.
- Međuljudske interakcije i relacije (*InterpersonalInteractionsAndRelationships*) – sastavljen od aktivnosti vezanih za različite oblike društveno dozvoljenih međuljudskih interakcija.

Model aktivnosti čoveka predstavlja nadgradnju modela struktura i funkcija u kontekstu njegovog okruženja. Koncepti aktivnosti vezani za anatomiju i fiziologiju čoveka će biti korišćeni za definisanje generičkih termina koji opisuju radnje čoveka u relaciji sa njegovim okruženjem. Drugim rečima, ovi termini će predstavljati stereotipove UML koncepata stanja i aktivnosti. Na slici 4.7 dat je generički opis koncepta zadatka korišćenjem UML mehanizma aktivnosti. Zadatak je modelovan kao UML aktivnost (*TaskActivity*) koju čine elementarne akcije (*TaskAction*). Drugim rečima, svaki zadatak ima faze izvršavanja u koje spadaju alokacija potrebnih resursa, iniciranje zadatka, izvršavanje zadatka, određivanje ritma izvršavanja zadatka i završetak zadatka. Modelom je definisan i redosled faza.



Slika 4.7. *Generički tipovi akcija u okviru aktivnosti čoveka.*

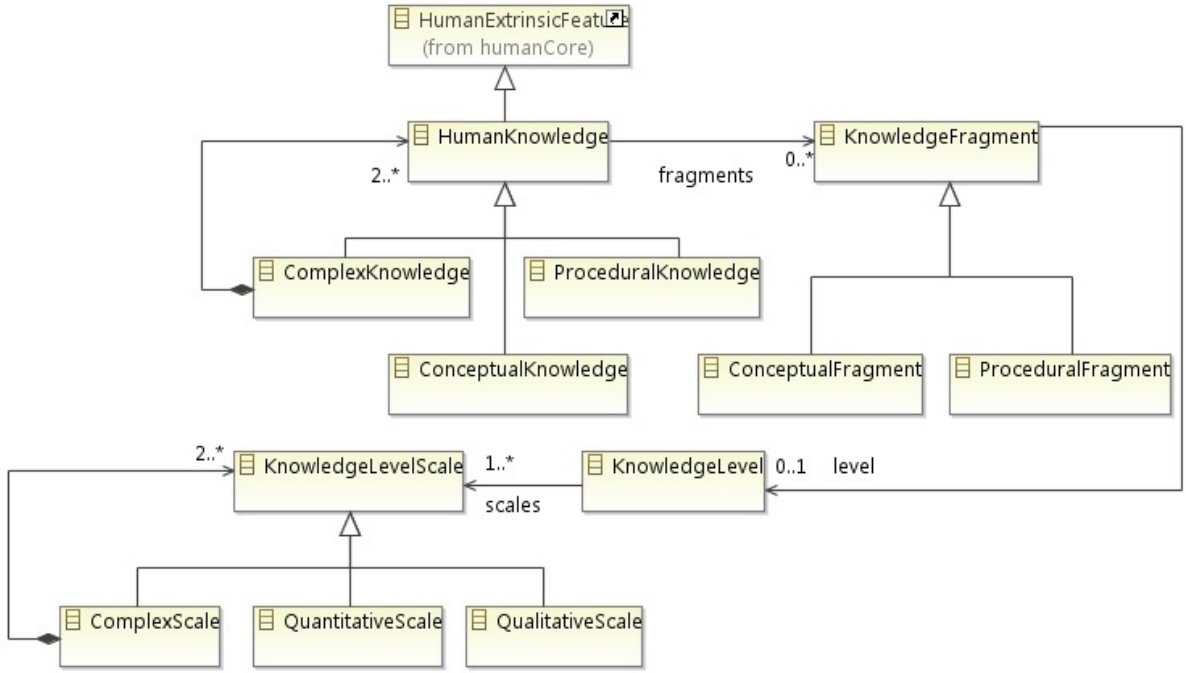
4.2.1.2. Svojstva čoveka

Model svojstava predstavlja osobine koje čovek stiče i razvija u toku života. Na formiranje ovih osobina utiču urođeni entiteti (anatomija, fiziologija i povezane aktivnosti) i celokupno čovekovo okruženje. U zavisnosti od toga da li su ove osobine subjektivne prirode (vezane za psihologiju čoveka) ili objektivne prirode (vezane za specifičnu oblast angažovanja čoveka), klasifikovane su kao unutrašnja svojstva (*HumanIntrinsicFeature*) i spoljašnja svojstva (*HumanExtrinsicFeature*).

U narednim odeljcima detaljnije će biti opisani značajniji podmodeli modela svojstava čoveka.

4.2.1.2.1. Znanje čoveka

Na slici 4.8 prikazana su osnovni elementi strukture modela znanja čoveka.



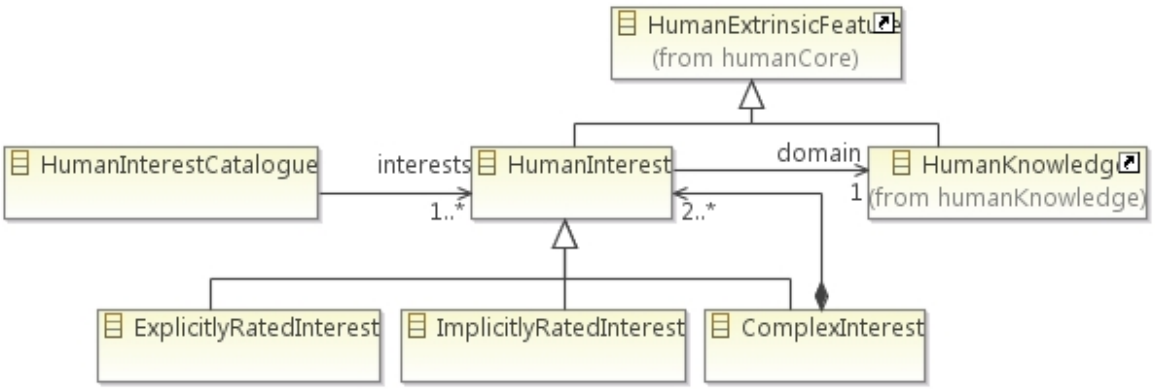
Slika 4.8. *Osnovni koncepti modela znanja čoveka.*

Kao osnovu za formiranje modela znanja iskoristili smo pregled istraživanja u oblasti adaptivne hipermedije i inteligentnih tutorskih sistema [Brusilovsky07]. Pomenute oblasti modeluju korisnika sistema sa stanovišta domenskog znanja. U opštem slučaju, znanje može biti predstavljeno u proceduralnom ili konceptualnom obliku. Proceduralno znanje (*ProceduralKnowledge*) opisuje veštine rešavanja problema i najčešće je dato u obliku pravila. Konceptualno znanje predstavljaju činjenice i relacije između njih i najčešće je dato u obliku mreže koncepata. U praksi se najčešće javlja kombinacija dva tipa (*ComplexKnowledge*). Konkretno znanje je struktuirano u skup elemenata koji predstavljaju fragmente (*KnowledgeFragment*) i koji u skladu sa načinom predstavljanja znanja mogu biti konceptualni ili proceduralni. Suština struktuiranja znanja u skup povezanih i relativno nezavisnih fragmenata je procena nivoa znanja čoveka (*KnowledgeLevel*). Nivo znanja se izražava korišćenjem skale (*KnowledgeLevelScale*). U praksi se ovo najčešće realizuje poređenjem ekspertnog znanja koje je unapred uneto u sistem sa znanjem korisnika. Skala može biti kvalitativna (*QualitativeScale*) (skup predefinisanih vrednosti, na primer dobar-prosečan-loš), kvantitativna (*QuantitativeScale*) (kao što je verovatnoća poznavanja pojma) ili kombinacija (*ComplexScale*).

4.2.1.2.2. Interesovanja čoveka

Modeli interesovanja čoveka se sreću u adaptivnim sistemima za pribavljanje i filtriranje velikih količina informacija kao što su enciklopedije, vesti, elektronska prodaja, muzejski vodiči i dr. U tom pogledu postoji uska povezanost modela znanja i modela interesovanja čoveka. Sa jedne strane, u modelima znanja interesovanja se mogu posmatrati kao ciljevi učenja (*eng. learning goals*). Sa druge strane, struktura interesovanja je identična strukturi znanja. U tom pogledu interesovanja mogu biti

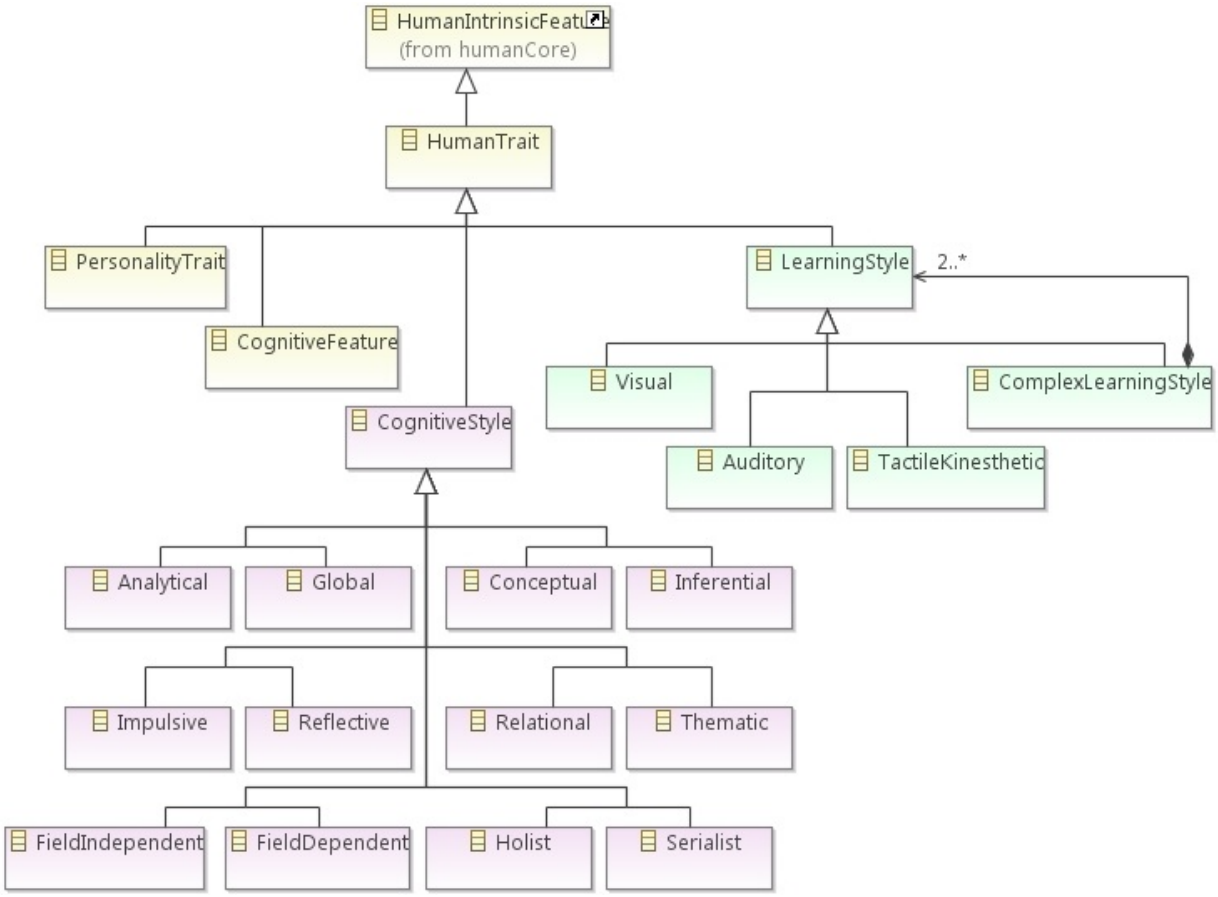
predstavljena u obliku vektora težine ključnih reči (odgovara jednostavnoj formi konceptualnog znanja sa nekom od skala) ili složenijeg modela znanja oblasti interesovanja. Primer korišćenja složenijeg modela može biti sistem za personalizaciju vesti koji na ovaj način može modelovati interesovanja u vezi sa različitim temama na osnovu kriterijuma kao što je geografska lokacija čoveka. Osnovni elementi modela interesovanja su prikazani na slici 4.9. U zavisnosti od načina formiranja, interesovanja mogu biti eksplicitno data od strane čoveka (na primer ankete, upitnici, narudžbine, potrošačke korpe) (*ExplicitlyRatedInterest*), implicitno kreirana (na primer praćenjem navigacije i akcija korisnika) (*ImplicitlyRatedInterest*) ili kombinacija (*ComplexInterest*). Uveden je i koncept kataloga kao mehanizam za grupisanje interesovanja.



Slika 4.9. *Osnovni koncepti modela interesovanja čoveka.*

4.2.1.2.3. Osobine čoveka

Model osobina čoveka (*eng. HumanTrait*) opisuje relativno stabilne karakteristike čoveka koje se ne mogu promeniti ili se mogu postepeno menjati u toku dužeg intervala vremena. Ove karakteristike se kod pojedinca utvrđuju primenom posebno dizajniranih psiholoških testova. Dok je psihologija klasifikovala i izučava veliki broj crta ličnosti čoveka, u oblasti personalizacije softverskih sistema naglasak je na izučavanju dva tipa crta ličnosti – kognitivni stil (*eng. cognitive style*) i stil učenja *(eng. learning style*). Pod kognitivnim stilom se podrazumeva tipičan način organizacije, reprezentacije i procesuiranja informacija kod pojedinca [Riding98]. Stil opisuje uobičajen način razmišljanja, pamćenja i rešavanja problema. Po svojoj prirodi predstavlja bipolarnu dimenziju za razliku od mogućnosti čoveka koje su unipolarne (imaju rang vrednosti od nulte do maksimalne vrednosti). Na ovaj način se posedovanje više od jedne mogućnosti smatra prednošću, dok posedovanje kognitivnog stila označava tendenciju ponašanja po određenom obrascu. U stručnoj literaturi se može naći veliki broj dimenzija kroz koje se može posmatrati kognitivni stil [Chen02]. Postojeće dimenzije kognitivnog stila prikazane su na slici 4.10.

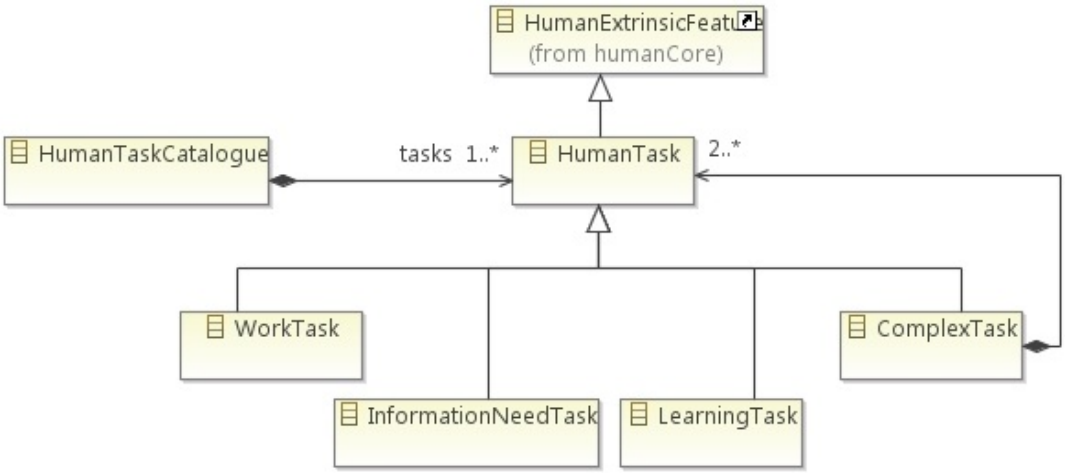


Slika 4.10 *Osnovni elementi modela osobina čoveka.*

Dve dimenzije kognitivnog stila koje se našle širu primenu jesu holistički/serialistički (*eng. holist/serialist*) [Pask72] i domenski zavisan/nezavisan (*eng. field-dependent/independent*) [Witkin77]. Holistički tip ima hijerarhijski pristup rešavanju problema u smislu posmatranja u širem okviru celine, dok serijalistički tip ima sekvencijalni pristup i rešava problem korak po korak bez sagledavanja šire celine. Drugi par tipova se odnosi na tendenciju percepcije okruženja na analitički ili globalan način. Domenski nezavisan tip doživljava pojave kao diskretne u odnosu na okruženje, dok domenski zavisan tip pojave i okruženje doživljava kao nerazdvojnu celinu. Postoji i veza između ova dva tipa i učenja. Domenski nezavisan tip uči efikasnije pod uticajem unutrašnje motivacije, dok se domenski zavisan tip podstiče socijalnim motivacionim faktorima i ima naglašenu socijalnu orijentaciju. Pojam stila učenja je usko povezan sa kognitivnim stilom, ali ima uži fokus i orijentisan je na proces učenja. Ne postoji jedinstvena i opšteprihvaćena klasifikacija stilova učenja. Sa obzirom da interakciju čoveka i računara opisujemo putem komunikacionih kanala, uveli smo klasifikaciju kao na slici 4.10. Svaki pojedinac može posedovati sklonost ka jednom stilu učenja (*Visual, Auditory, TactileKinesthetic*) ili kombinaciji različitih stilova (*ComplexLearningStyle*). Pored kognitivnog stila i stila učenja, modelom osobina obuhvaćene su i crte ličnosti (*PersonalityTrait*) (na primer introvertan/ekstravertan) i kognitivna svojstva (*CognitiveFeature*) (na primer kapacitet radne memorije).

4.2.1.2.4. Zadaci čoveka

Model zadataka korisnika opisuje zadatke korisnika vezane za specifičan domen korišćenja (slika 4.11). Termin zadatka predstavlja neposrednu svrhu rada korisnika u kontekstu konkretnog sistema. U tom pogledu se on razlikuje od koncepta zadatka kao entita čoveka. U nekim sistemima umesto pojma zadatka upotrebljava se termin cilj (*eng. goal*). U zavisnosti od vrste sistema to može biti neposredni cilj u korišćenju sistema (*WorkTask*) u slučaju adaptivnih interfejsa; potreba za informacijama (*InformationNeedTask*) u slučaju adaptivnih sistema za pribavljanje informacija; cilj učenja (*LearningGoal*) u edukativnim sistemima. Složen zadatak (*ComplexType*) predstavlja kombinaciju elementarnih tipova. Modelom je predviđan i mehanizam grupisanja zadataka (*HumanTaskCatalogue*).



Slika 4.11. *Osnovni tipovi zadataka čoveka.*

4.2.2. Modelovanje okruženja

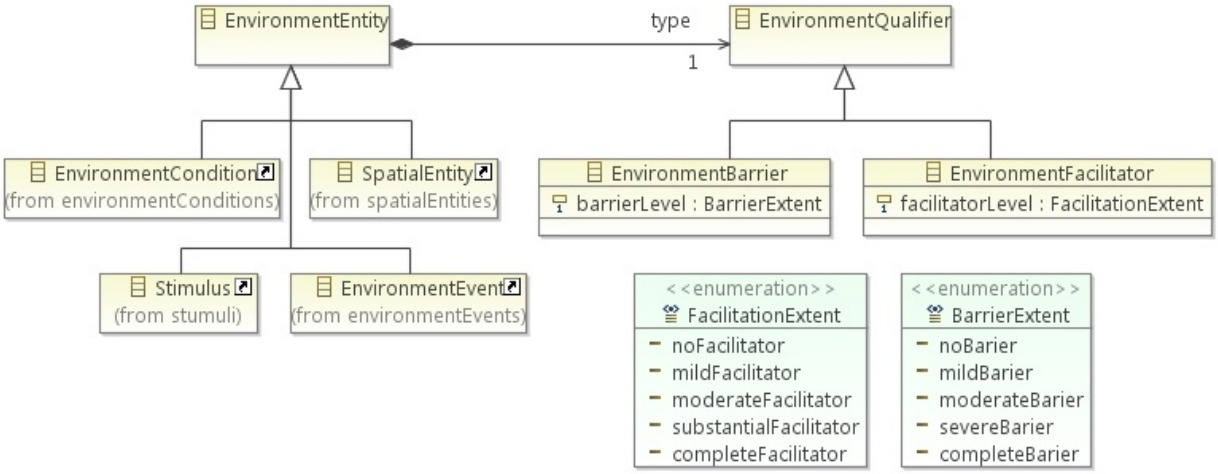
Interakcija između čoveka i računara se odvija u okruženju koje na različite načine utiče na interakciju. Model okruženja definiše osnovne koncepte okruženja od interesa za interakciju između čoveka i računara. Pri kreiranju modela okruženja koristili smo znanja iz SOUPA (*eng. Standard Ontology for Ubiquitous and Pervasive Applications*) ontologije [Chen04] i ICF klasifikacije. Model okruženja je struktuiran u potpakete:

- Prostorni entiteti (*SpatialEntities*), u kojem definišemo koncepte vezane za fizički prostor, fizičke objekte koji se u njemu nalaze i relacije između njih.
- Uslovi u okruženju (*EnvironmentConditions*), u kojem su uvedeni opisi fizičkih uslova u okruženju, tj svetlosne i zvučne karakteristike, kao i stanje vazduha.
- Dimenzije i merne jedinice (*DimensionsAndMeasurementUnits*), u okviru kojeg su identifikovane standardne fizičke dimenzije i jedinice mere.

- Događaji u okruženju (*EnvironmentEvents*), gde smo uveli koncept događaja u okruženju koji generiše stimulanse (draži) koje čovek ili računar registruje.

- Stimulansi (*Stimulus*), gde smo opisali vrste stimulansa (draži) koje čovek ili računar može osetiti ili generisati.

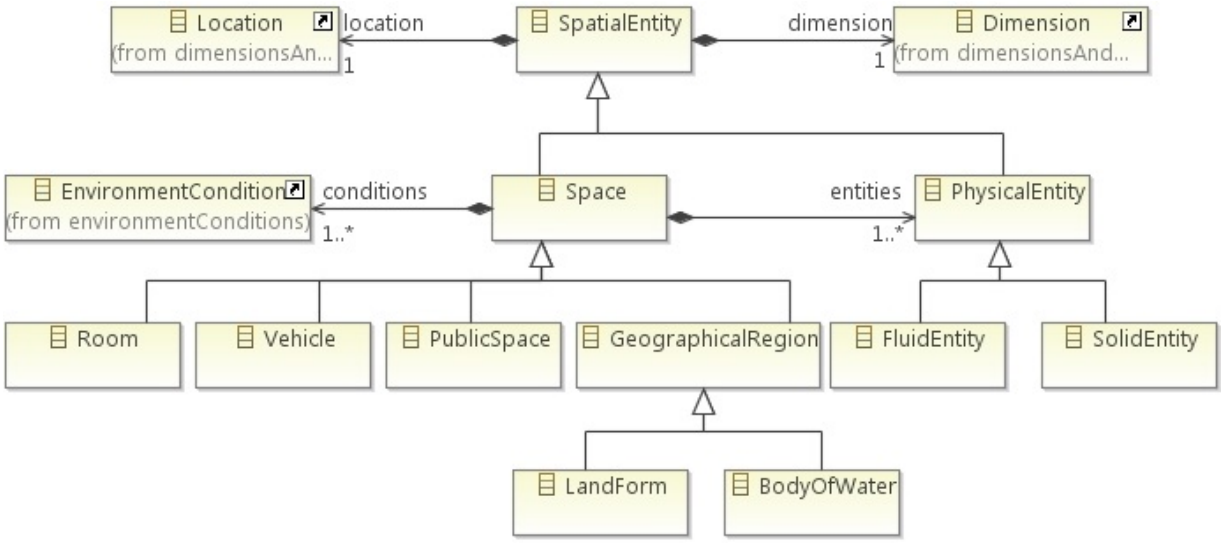
Na slici 4.12 je prikazana klasifikacija osnovnih entiteta okruženja. Svakom od entiteta je pridružen generički kvalifikator koji govori o tome da li entitet vrši pozitivan ili negativan uticaj na interakciju čoveka i računara u datom kontekstu. Ovo svojstvo postoji u ICF klasifikaciji.



Slika 4.12. *Klasifikacija entiteta okruženja.*

4.2.2.1. Prostorni entiteti

Interakcija između čoveka i računara uvek se odvija u odgovarajućem okruženju. Pojavom prenosivih računara manjih dimenzija, komunikacija između čoveka i računara počinje da se odvija u veoma raznolikim okruženjima. Zbog toga je prilikom projektovanja korisničkih interfejsa potrebno uzeti u obzir i karakteristike okruženja u kome će se odvijati interakcija, kako bi se interakcija prilagodila tom okruženju. Na slici 4.13 prikazani su osnovni prostorni entiteti modela okruženja. Fizičko okruženje interakcije prvenstveno definišemo kao fizički prostor (*Space*) u kojem su smešteni fizički objekti (*PhysicalEntity*). Oblik i razvoj interakcije u fizičkom okruženju određuju lokacija (*Location*), dimenzije (*Dimension*), atmosferski uslovi (*EnvironmentCondition*), kao i specifičan tip prostora u kojem se interakcija odvija.



Slika 4.13. *Koncepti za opis prostornih entiteta.*

Lokacija precizno određuje mesto na kojem se odvija interakcija. Sa stanovišta prostora i objekata u njemu, lokacija može biti data apsolutno (geografske koordinate) i relativno (u odnosu na druge prostorne entitete). Informacija o lokaciji povezuje prostorne entitete kako međusobno, tako i sa događajima i stimulansima koji se javljaju u okruženju. Na ovaj način je i sveukupan kontekst interakcije bliže određen.

Specifičan tip prostora određuje pre svega geometriju prostora u kojem se odvija interakcija. Zajedno sa dimenzijom prostora definiše osnovne ergonomske karakteristike okruženje, kao što je stepen slobode pokreta, uticaja atmosferskih uslova (otvoren ili zatvoren prostor) i socijalna komponenta interakcije (prisustvo drugih ljudi). Stoga smo u modelu identifikovali osnovne tipove prostora:

- Soba (*Room*), predstavlja namenski prilagođen prostor koji predviđa interakciju u kontekstu unapred određenog skupa aktivnosti, bilo da se radi o poslu (kancelarija) ili privatnom životu (soba u stanu).
- Vozilo (*Vehicle*), definiše prostor u kojem se odvija interakcija koji poseduje svojstvo mobilnosti i pre svega se odnosi na različite vrste prevoznog sredstva.
- Javni prostor (*PublicSpace*), određuje prostor u kojem se odvija i interakcija sa drugim ljudima, kao što su muzeji, šetališta, čekaonice, tržni centri i drugi.
- Geografski region (*GeographicalRegion*), opisuje neurbano, prirodno okruženje u kojem prisustvo drugih ljudi nije izraženo. Ova okruženja mogu biti kopneni ili vodeni oblici.

4.2.2.2. Uslovi u okruženju

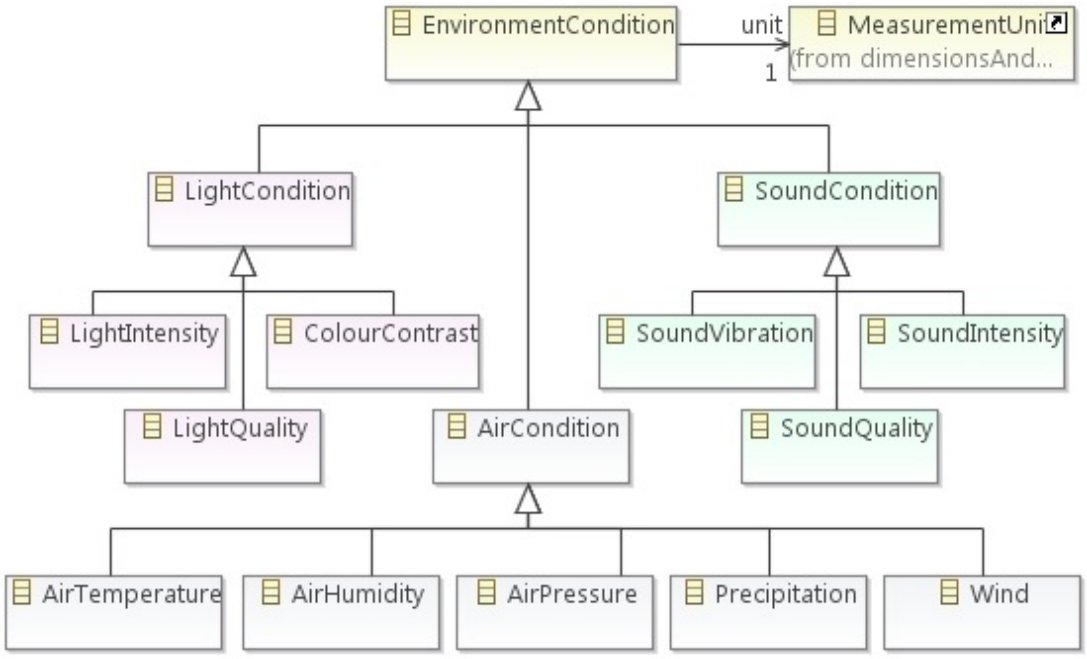
Fizički uslovi u okruženju direktno utiču na kanale komunikacije između čoveka i računara. U modelu smo identifikovali tri osnovne grupe fizičkih uslova u okruženju (slika 4.14):

- Svetlosni uslovi okruženja (*LightCondition*), određuju svetlosne karakteristike okruženja kao što su intenzitet osvetljenja, kvalitet svetla i kontrast. Svetlosne karakteristike prvenstveno utiču na vizuelnu komunikaciju čoveka i računara.

- Vazdušni uslovi u okruženju (*AirCondition*), opisuju stanje vazduha u atmosferi. Stanje se može opisivati preko temperature vazduha, vlažnosti vazduha, vazdušnog pritiska, strujanja vazduha i padavina. Ovi uslovi utiču na funkcionisanje računarskih uređaja i na psihofizičke sposobnosti čoveka.

- Zvučni uslovi u okruženju (*SoundCondition*), deifnišu karakteristike prostiranja zvučnih talasa u prostoru. Ovde spadaju intenzitet zvuka, kvalitet zvuka i zvučne vibracije. Ova grupa uslova direktno utiče na auditornu komunikaciju između čoveka i računara.

Tipičan scenario komunikacije čoveka i računara podrazumeva istovremeni uticaj većeg broja fizičkih uslova u okruženju. U zavisnosti od specifičnosti okruženja (na primer, otvoren ili zatvoren prostor, kontrolisani ili nekontrolisani uslovi) oni mogu ispoljavati manji ili veći uticaj. U opštem slučaju, fizički uslovi predstavljaju dominantan faktor okruženja koji utiče na komunikaciju čoveka i računara.



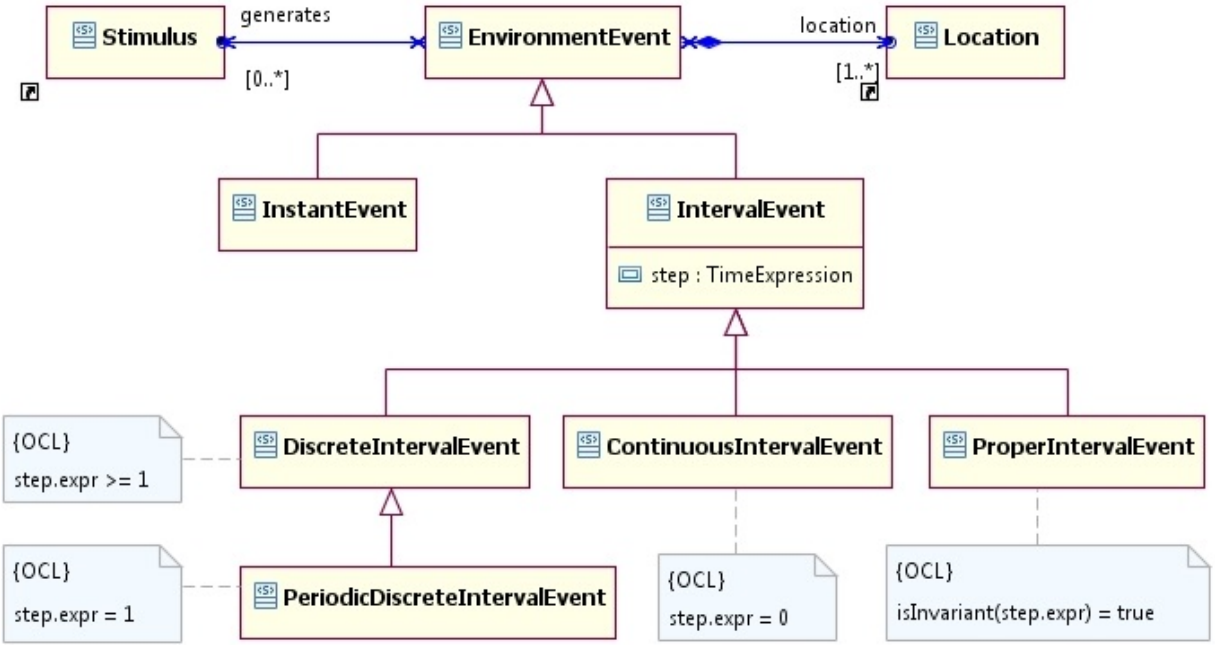
Slika 4.14. *Klasifikacija fizičkih uslova u okruženju.*

4.2.2.3. Događaji u okruženju

Koncept događaja predstavlja apstrakciju promene u okruženju koja će generisati stimulans koji čovek ili računar može registrovati. Događaji predstavljaju akcije koje imaju prostornu i vremensku dimenziju. Vremenska dimenzija govori da li se događaj odigrava u trenutku vremena ili u vremenskom intervalu. Prostorna dimenzija je data lokacijom događaja. Lokacija može biti zadata apsolutno ili relativno (u odnosu na druge događaje ili prostorne entitete). Na slici 4.15 je data

generička klasifikacija tipova događaja u modelu. Događaji su razvrstani u dva osnovna tipa (*InstantEvent*, *IntervalEvent*) u zavisnosti od toga da li se događaj odigrava u trenutku ili intervalu vremena. Događaji intervali su u zavisnosti od koraka intervala dalje razvrstani kao:

- Diskretni intervalni događaj (*DiscreteIntervalEvent*), kod kojeg je vrednost koraka veća ili jednaka jediničnoj vrednosti. U okviru ovog tipa postoji periodični diskretni interval kod kojeg vrednost koraka uvek ima jediničnu vrednost.
- Kontinualni intervalni događaj (*ContinuousIntervalEvent*), kod kojeg vrednost koraka teži nultoj vrednosti.
- Pravilni intervalni događaj (*ProperIntervalEvent*), kod kojeg se vrednost koraka ne menja u toku vremena.



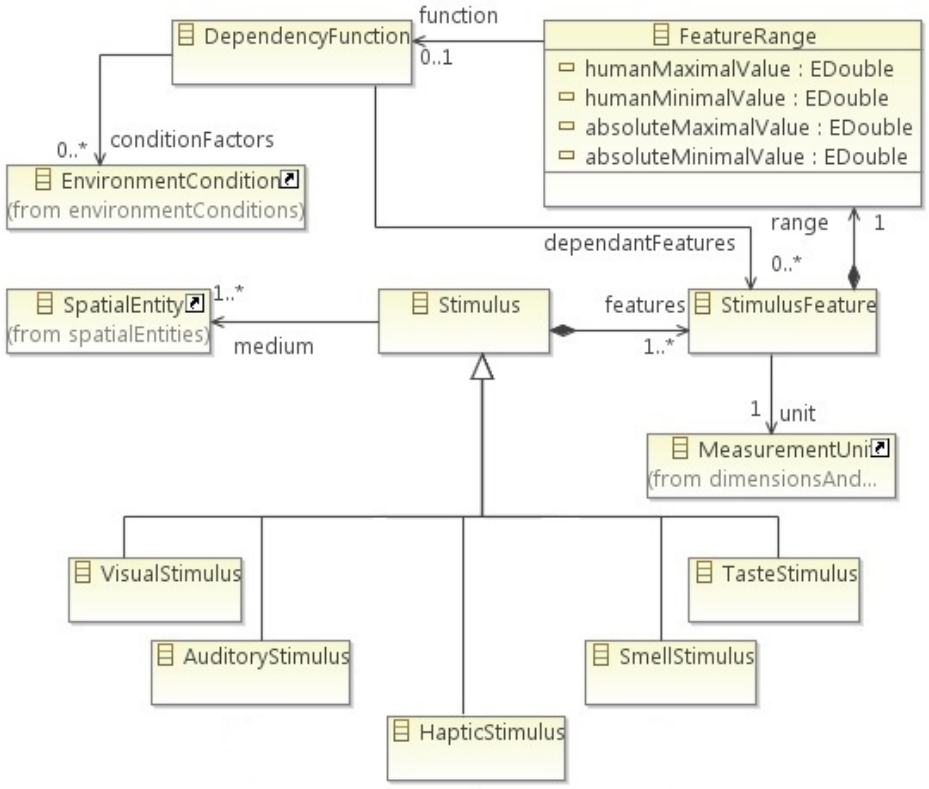
Slika 4.15. *Generički tipovi događaja u okruženju.*

4.2.2.4. Stimulansi

U fizičkom smislu komunikacija između čoveka i računara svodi se na razmenu signala odgovarajućeg oblika. Zbog toga smo u modelu identifikovali koncept stimulansa (*Stimulus*), koji predstavlja fizički signal koji čovek ili računar mogu da detektuju ili generišu u toku komunikacije. Na slici 4.16 je dat opis koncepta stimulansa.

Stimulans može da koristi različite prostorne entitete kao medij za prenos signala. Takođe, svaki stimulans ima jednu ili vise osetnih osobina (*StimulusFeature*), tj. svojstava koje ljudski ili računarski senzorski sistem mogu prepoznati u signalu. Na primer, čovek u zvuku opaža intenzitet i ton (frekvenciju), dok u svetlosti opaža frekvenciju (boju) i intenzitet (osvetljenje). Svako od svojstava oseta poseduje ograničenja (*FeatureRange*) koja su specifična za čoveka i računarski uređaj. Tako na

primer, čovek u proseku detektuje zvučne signale frekvencije od 20 Hz do 20 kHz. Takođe, često su senzorska ograničenja međusobno povezana, tj. nisu nezavisna. Ova zavisnost se može izraziti odgovarajućom funkcijom (*DependencyFunction*) koja u razmatranje može uzeti i uslove u okruženju. Na primer, prag čujnosti, najniži intenzitet zvuka koji čovek može da čuje, zavisi od frekvencije zvuka. Definisanje ovih ograničenja omogućava određivanje minimalne vrednosti nekog signala koji može da se detektuje, odnosno maksimalnu vrednost nekog signala pri kojoj ne dolazi do oštećenja senzorskog sistema.



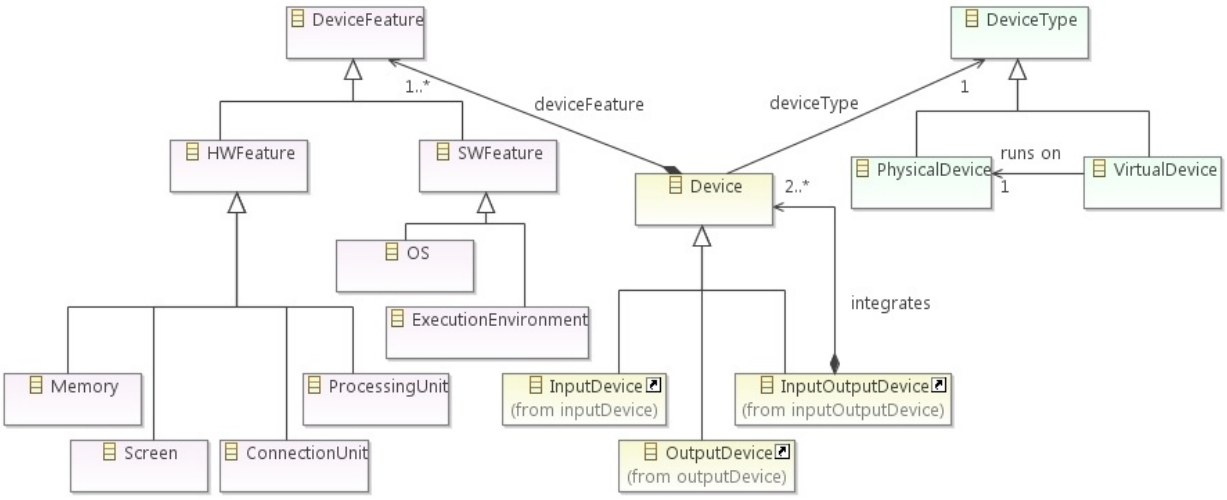
Slika 4.16. *Koncepti stimulansa kao mehanizma razmene informacija između čoveka i okruženja.*

Oblici stimulansa su klasifikovani kao:

- Vizuelni stimulans (*VisualStimulus*), predstavlja generički opis vizuelne draži i svetlosti kao fizičkog medija za prenos vizuelnih draži.
- Zvučni stimulans (*AuditoryStimulus*), predstavlja generički opis zvučne draži koja se prenosi putem vibracija fluida.
- Dodirni stmulans (*HapticStimulus*), predstavlja generički opis draži koja stimuliše čulo dodira tj. spoljašnju površinu tela čoveka (*eng. tactile sensing*) ili se odnosi na mehaničke stimulacije, tj. svest o poziciji udova, pokreta ili tonusa mišića (*eng. kinesthetic sensing*).
- Mirisni stimulans (*SmellStimulus*), opisuje prenos podataka putem mirisnih svojstava fluida.
- Stimulans ukusa (*TasteStimulus*), predstavlja generički opis draži koja stimuliše čulo ukusa.

4.2.3. Modelovanje računarskog uređaja

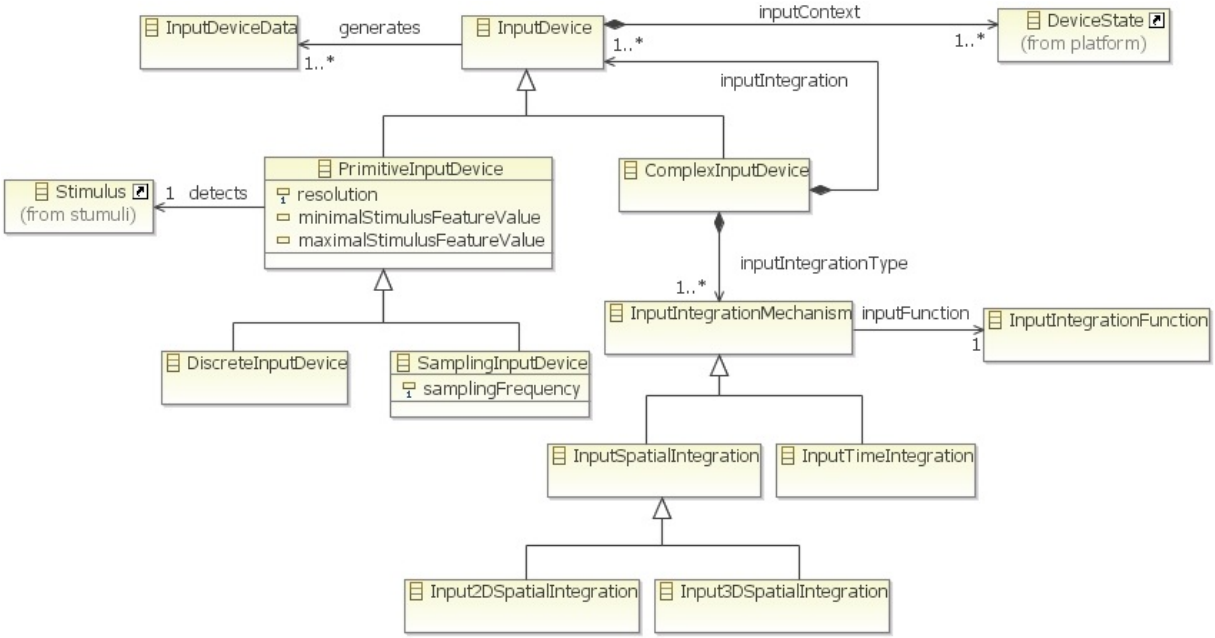
Prilikom izrade modela računarskih uređaja korstili smo znanja poznate taksonomije uređaja za interakciju [Bernsen94]. U nastavku će biti opisani koncepti uređaja relevantni za interakciju računara sa čovekom. U skladu sa tokovima informacija između čoveka i računara, uređaji korisničkog interfejsa mogu biti ulazni, izlazni i ulazno-izlazni (slika 4.17). Ulazni uređaji (*InputDevice*) prevode stimulanse iz okruženja u računarski čitljiv oblik. Izlazni uređaji (*OutputDevice*) prevode računarski čitljive podatke u oblik pogodan za obradu od strane čoveka. Pored toga, kao posebnu klasu izdvojili ulazno-izlazni uređaj (*InputOutputDevice*) koji integriše prethodne tipove. Primer ovakvog uređaja predstavlja ekran osetljiv na dodir (*eng. touchscreen*). Sa stanovišta tehnologije realizacije, svaki uređaj može biti opisan skupom hardverskih i softverskih karakteristika. Pored toga, uređaj se sa aspekta fizičke realizacije može posmatrati kao fizički (*PhysicalDevice*) ili virtuelni uređaj (*VirtualDevice*). Virtuelni uređaj predstavlja softversku realizaciju fizičkog uređaja. Korisniku se stvara virtuelna predstava fizičkog uređaja. Najčešće se javljaju u obliku simulatora [Jovanović10], gde je osnovni motiv njihovog uvođenja cena izrade i korišćenja u odnosu na fizički uređaj.



Slika 4.17. *Osnovni koncepti modela računarskog uređaja.*

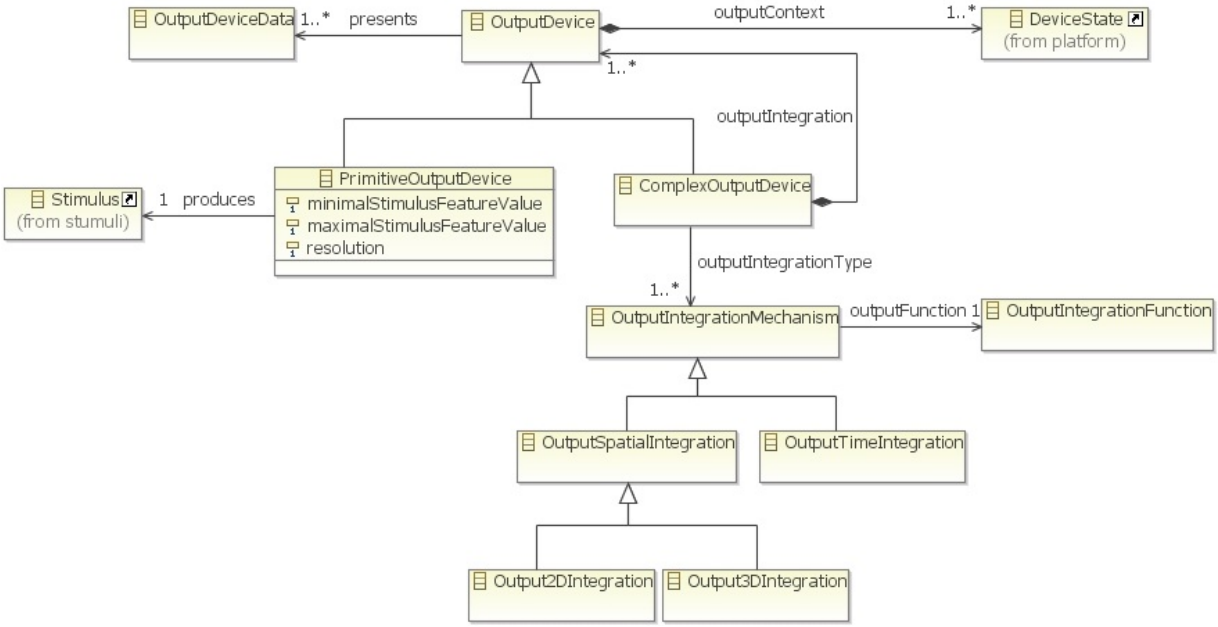
Ulazni uređaj (slika 4.18) detektuje različite oblike fizičkih signala (*Stimulus*) u okruženju i transformiše ih u oblik pogodan za korišćenje od strane računara, tj. podatke ulaznog uređaja (*InputDeviceData*). Ulazni uređaj je bliže opisan i odgovarajućim stanjem (*DeviceState*) koje određuje kontekst generisanja i interpretacije ulaznih podataka. Ulazni uređaj može biti jednostavan (*PrimitiveInputDevice*) ili složen (*ComplexInputDevice*). Jednostavan ulazni uređaj detektuje specifičnu vrstu stimulansa u određenom rasponu vrednosti i generiše računarski čitljive podatke. U pogledu načina rada, jednostavan uređaj može biti diskretan (*DiscreteInputDevice*) ili uzorkujući (*SamplingInputDevice*). Diskretan ulazni uređaj detektuje stimulans i generiše ulazne podatke kada osobina oseta dostigne vrednost koja se nalazi u definisanom opsegu vrednosti uređaja (na primer, svetlosni senzor). Uzorkujući ulazni uređaj generiše podatke stimulansa u intervalu vremena odgovarajućom brzinom (*samplingFrequency*), nezavisno od promene fizičkog signala (na primer, elektro-fiziološki uređaji). Složen ulazni uređaj integriše veći broj primitivnih ili složenih uređaja.

Integracija uređaja je opisana ulaznim mehanizmom integracije (*InputIntegrationMechanism*). Sam mehanizam može biti formalno opisan odgovarajućom funkcijom (*InputIntegrationFunction*). Mehanizmi integracije ulaznih uređaja mogu biti vremenski ili prostorni. Vremenski mehanizmi su opisani parametrima vremena. Ova vrsta mehanizama se može sresti kod uređaja koji kombinuju različite oblike medija kao što je kamera. Prostorni mehanizmi integracije se zasnivaju na prostornim relacijama između različitih uređaja. Primeri ove vrste mehanizama se mogu naći kod uređaja kao što su tasteri miša i tastature (dvodimenzionalna integracija) ili integrisani skup senzora pokreta kod platformi video igara kao što je *Microsoft XBox* (trodimenzionalna integracija).



Slika 4.18. *Model ulaznog računarskog uređaja.*

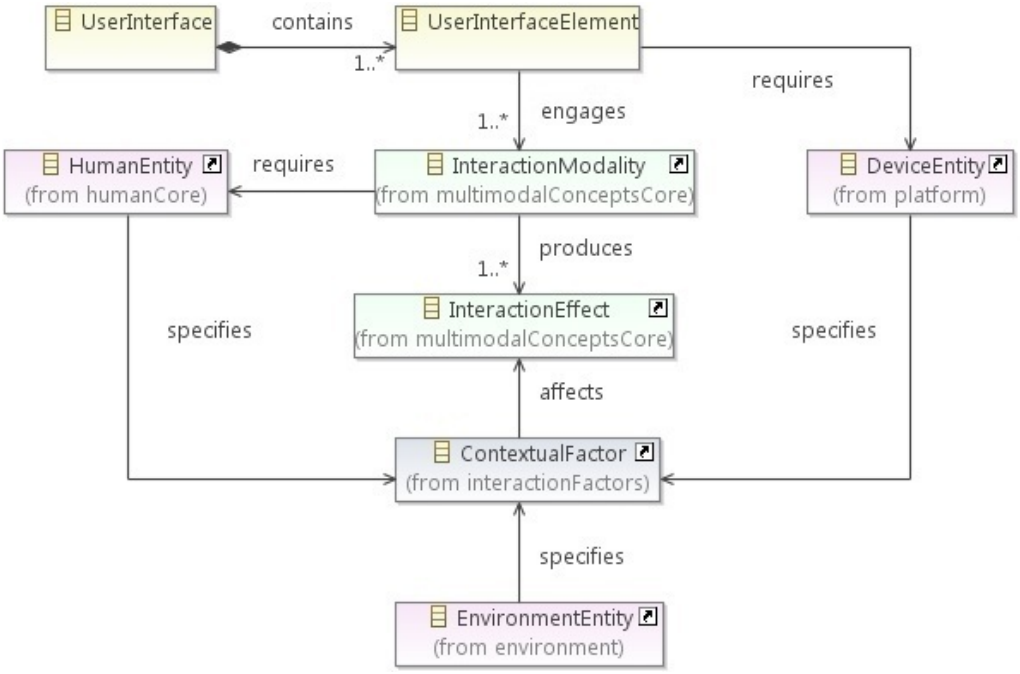
Izlazni uređaji (slika 4.19) na sebi svojstven način prezentuju podatke i kreiraju fizičke signale koje čovek može registrovati. Izlazni uređaj može biti primitivan (*PrimitiveOutputDevice*) ili složen (*ComplexOutputDevice*). Primitivan izlazni uređaj prezentuje određenu strukturu podataka i formira fizički signal u definisanom opsegu vrednosti. Složen izlazni uređaj integriše veći broj primitivnih ili složenih ulaznih uređaja korišćenjem izlaznih mehanizama integracije (*OutputIntegrationMechanism*) koji su bliže određeni izlaznom funkcijom integracije (*OutputIntegrationFunction*). Ulazni mehanizmi integracije mogu biti prostorni i vremenski. Vremenski mehanizmi integracije se mogu naći kod različitih sistema zvučnika, dok primer prostornog mehanizma integracije predstavlja displej.



Slika 4.19. *Model izlaznog računarskog uređaja.*

4.2.4. Modelovanje konteksta interakcije između čoveka i računara

U ovoj sekciji biće opisani koncepti kontekstno-osetljive interakcije čoveka računara. Modelovanje kontekstno-osetljive interakcije objedinjuje i uspostavlja relacije između koncepata modela čoveka, okruženja i platforme sa jedne strane (ontološki modeli), i koncepata specifičnih za korisnički interfejs (prototipski modeli) sa druge strane. Na slici 4.20 su prikazane relacije između koncepata korisničkog interfejsa i koncepata ontoloških modela koji se javljaju kao faktori konteksta.



Slika 4.20. *Osnovne komponente kontekstno-osetljive interakcije čoveka i računara.*

U opisivanju same komunikacije čoveka i računara oslanjamo se na postojeće koncepte načina komunikacije (*InteractionModality*) i efekata interakcije (*InteractionEffect*) [Obrenovic04]. Korisnički interfejs (*UserInterface*) se sastoji od skupa elemenata korisničkog interfejsa (*UserInterfaceElement*), gde svaki od elemenata u interakciji sa čovekom koristi specifične komunikacione kanale koji angažuju određene ljudske aparate. Kao posledica angažovanja sposobnosti čoveka javljaju se odgovarajući efekti. Sa druge strane, koncepti čoveka, platforme i okruženja ispoljavaju odgovarajući uticaj na generisane efekte. U tom pogledu oni određuju faktore konteksta interakcije (*ContextualFactor*).

U nastavku se daje osvrt na postojeće koncepte načina komunikacije i efekata na koje se oslanjamo, kao i detaljniji opis faktora konteksta interakcije čoveka i računara.

4.2.4.1. Načini komunikacije

Način komunikacije (*eng. modality*) je definisan kao oblik interakcije između čoveka i računara koji ima za cilj angažovanje nekih od ljudskih sposobnosti. Kao posledica angažovanja sposobnosti čoveka javljaju se efekti ili poruke. Na ovaj način se korisnički interfejs može opisati preko skupa poruka koje dizajner šalje ka korisniku. Radi boljeg razumevanja, ovde će ukratko biti opisan koncept načina komunikacije, dok je detaljniji pregled dat u [Obrenovic04].

Način komunikacije može biti jednostavan ili složen. Složen način komunikacije integriše druge načine komunikacije u cilju kreiranja njihove simultane upotrebe. Sa druge strane, jednostavan način komunikacije predstavlja primitivni oblik interakcije. Jednostavan način komunikacije može biti ulazni i izlazni. Kao referentni učesnik za definisanje smera komunikacije uzet je računar, tako da ulazni način predstavlja ulaz računara, a izlazni način prezentaciju od strane računara. Ulazni način komunikacije pretvara neku od ljudskih komunikacionih akcija (kao što je pokret ili govor) u oblik pogodan za obradu od strane računara. Izlazni način komunikacije predstavlja prikaz podataka od strane računara u cilju prenosa podataka ka čoveku. Izlazni način komunikacije može biti statički, poput slike ili teksta, i dinamički, kao što je animacija.

4.2.4.2. Efekti interakcije

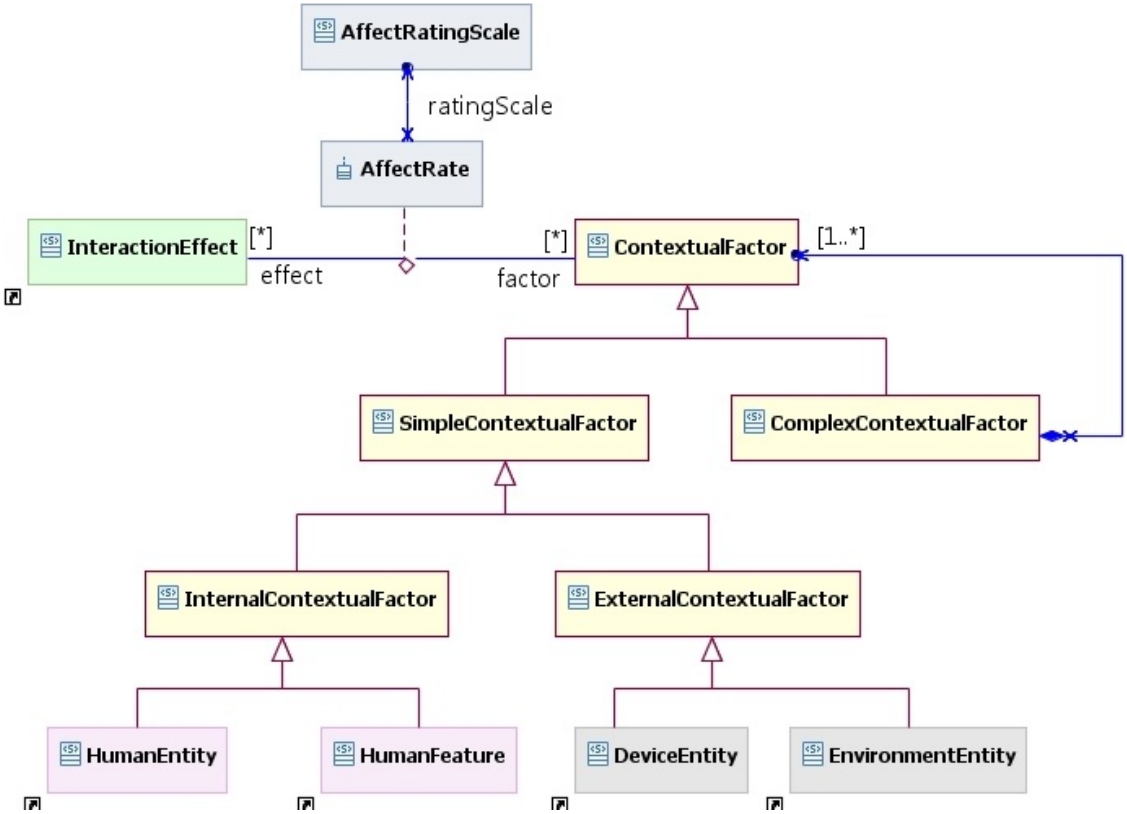
Pojam efekta ili poruke predstavlja generički koncept koji opisuje rezultat angažovanja funkcionalnog aparata čoveka u komunikaciji sa računarom. U skladu sa široko prihvaćenim pristupom modelovanja čoveka preko senzorskih, percepcijskih, motoričkih i kognitivnih kapaciteta, izvršena je sledeća klasifikacija efekata [Obrenovic04] :

- Senzorski efekti
- Percepcijski efekti
- Motorički efekti
- Kognitivni efekti
- Lingvistički efekti

U našem modelu, efekti su našli mesto u hijerarhiji funkcionalnih entiteta čoveka (Slika 4.4). Konkretni oblici efekata su izvedeni iz ICF funkcionalnih koncepata čoveka. Pored toga, neki oblici su prilagođeni komunikaciji čoveka i računara (na primer, različiti oblici grupisanja i isticanja). Senzorski efekti opisuju obradu stimulansa od strane ljudskih senzorskih aparata. Percepcijski efekti nastaju kao posledica obrade podataka dobijenih od senzora koju vrše percepcijski aparati čoveka. Primeri ovih efekata predstavljaju, na primer, prepoznavanje oblika, grupisanje ili isticanje. Ovi efekti mogu imati vizuelnu, akustičnu i druge forme u zavisnosti od vrste detektovanog stimulansa (Slika 4.19). Motorički efekti nastaju kao posledica mehaničkih radnji čoveka, poput pokreta ruke ili pritiska. Lingvistički efekti se odnose na govor, slušanje, čitanje i pisanje. Kognitivni efekti se javljaju u kontekstu mentalnih funkcija čoveka, kao što su pažnja ili pamćenje. Efekti su međusobno uslovljeni. Percepcijski efekti nastaju kao posledica senzorskih. Oni dalje utiču na kognitivne funkcije čoveka koje određuju mehaničke radnje koje će čovek preduzeti u datom kontekstu.

4.2.4.3. Faktori konteksta interakcije

Na slici 4.21 prikazana je osnovna klasifikacija faktora koji određuju kontekst komunikacije čoveka i računara.



Slika 4.21. *Faktori konteksta interakcije.*

Uticaj činioca konteksta na komunikaciju čoveka i računara određen je relacijom sa efektima interakcije. Uticaj je kvantifikovan proširenjem semantike relacije (*AffecRate*) i može biti izražen odgovarajućom skalom (*AffectRating*). Faktori konteksta interakcije mogu biti jednostavni (*SimpleContextualFactor*) i složeni (*ComplexContextualFactor*). Složen faktor konteksta interakcije integriše veći broj jednostavnih ili složenih faktora. Jednostavni faktori su sa stanovišta čoveka kao

učesnika u interakciji klasifikovani kao unutrašnji (*InternalContextualFactor*) i spoljašnji (*ExternalContextualFactor*). Unutrašnji faktori konteksta su entiteti i svojstva čoveka. Čovek predstavlja faktor interakcije koji može biti subjektivne ili objektivne prirode. Kada govorimo o objektivnom faktoru čoveka, najčešće se misli na trajna ili privremena ograničenja entiteta čoveka. Ova ograničenja se mogu specificirati u modelima entiteta (Slika 4.6, Slika 4.7, Slika 4.8). Subjektivni faktor čoveka je određen svojstvima čoveka i ogleda se u stečenim navikama i naklonjenostima ka određenim načinima komunikacije sa računarom (na primer, *HumanPreference*; stil učenja, kognitivni stil Slika 4.10). Spoljašnje faktore konteksta određuju karakteristike okruženja i uređaja za interakciju.

U konkretnom slučaju u praksi, rezultujući činioci konteksta inerakcije će predstavljati složenu kombinaciju različitih faktora čoveka, okruženja i uređaja. Predloženi način modelovanja omogućava sveobuhvatno i fleksibilno definisanje faktora konteksta interakcije čoveka i računara različitih nivoa složenosti.

4.3. *Rezime poglavlja*

U ovom poglavlju opisan je generički model kontekstno-osetljive interakcije čoveka i računara. Opisani su pristupi kreiranju modela, sastavni delovi modela i njihova uloga u definisanju interakcije čoveka i računara i sveukupnog konteksta u kojem se ona odvija. Faktori konteksta interakcije definisani su u okviru tri grupe modela, tj. modela čoveka, modela okruženja i modela računarskih uređaja. Pri opisivanju konteksta interakcije polazimo od fenomenološkog pristupa koji je baziran na čoveku koji kreira i menja kontekst u toku interakcije. Zbog toga je naglasak na modelu čoveka koji u našem slučaju objedinjuje koncepte korisnika sa stanovišta interakcije čoveka i računara sa jedne strane, i korišćenja računara u određenoj oblasti (edukativni sistemi, inteligentni tutorski sistemi, adaptivna hipermedija i dr.) sa druge strane. Pri opisivanju koncepata okruženja i računarskog uređaja akcenat je dat na čoveku, drugim rečima, konceptu fizičkog stimulansa ili draži koje čovek kreira ili detektuje. Na kraju smo, koristeći postojeće principe višenačinske komunikacije, definisali relacije između faktora konteksta interakcije sa jedne strane, i komponenti korisničkog interfejsa sa druge strane.

U narednom poglavlju opisaćemo upotrebu modela kontekstno-osetljive interakcije čoveka i računara u definisanju proširenja jezika za modelovanje softverskih sistema. Drugim rečima, uvešćemo proširenja za opis ontoloških modela. Nakon toga, definisaćemo proširenja specifična za modelovanje korisničkih interfejsa, tj. opise prototipskih modela. Ovde će biti opisani model zadataka, model korisničkog interfejsa nezavisan od načina komunikacije, model korisničkog interfejsa zavisan od načina komunikacije i model korisničkog interfejsa prilagođen tehnologiji implementacije.

5. Modelovanje kontekstno-osetljivih korisničkih interfejsa

U ovom poglavlju definišemo proširenja UML jezika u skladu sa prethodno opisanim modelom kontekstno-osetljive interakcije čoveka i računara. Uvedena proširenja omogućavaju korišćenje modela u razvoju kontekstno-osetljivih korisničkih interfejsa. Prilikom definisanja proširenja jezika za modelovanje rukovodili smo se sledećim principima:

- Predloženi skup proširenja mora očuvati standardni UML u smislu zadržavanja uloge i semantike postojećih elemenata jezika.
- Predloženi skup proširenja treba da bude familijaran UML projektantima u smislu usvajanja i primene u delu razvoju vezanom za korisnički interfejs.
- Predloženi skup proširenja treba da bude intuitivan za korišćenje dizajnerima korisničkog interfejsa koji imaju iskustvo korišćenja postojećih tehnika za modelovanje korisničkog interfejsa. U tom pogledu predložena proširenja ne bi trebalo da predstavljaju ograničenja u dizajnu korisničkog interfejsa.
- Predloženi skup proširenja treba da obezbedi integraciju korisničkog interfejsa sa ostatkom softverskog sistema. Ovde se misli na jasno definisane veze sa UML modelima koji opisuju funkcionalne i druge aspekte softverskih sistema.

U skladu sa prethodno definisanim principima pristupili smo kreiranju proširenja UML jezika. Kako postoji veći broj pristupa proširenju jezika UML, najpre ćemo ukratko opisati pristup za koji smo se opredelili, tj. mehanizam UML profila. Nakon toga opisujemo osnovne elemente i načine korišćenja definisanih profila UML jezika, tj. profil okruženja, profil računarskog uređaja, profil čoveka i profil kontekstno osetljive interakcije čoveka i računara, respektivno. Radi boljeg razumevanja, u prethodnom poglavlju su neki od modela u izvesnoj meri razmatrani sa aspekta proširenja elemenata UML jezika. Ovim je obuhvaćen prvi deo poglavlja u kojem su opisani opšti koncepti kontekstno-osetljive interakcije čoveka i računara (ontološki modeli, videti Slika 4.1) koji će biti korišćeni u dizajnu kontekstno-osetljivih korisničkih interfejsa. Drugi deo poglavlja je posvećen opisu modela vezanih za razvoj kontekstno-osetljivih korisničkih interfejsa (prototipski modeli, videti Slika 4.1). Drugim rečima, biće opisani model zadataka, model korisničkog interfejsa nezavisan od načina komunikacije, model korisničkog interfejsa zavisan od načina komunikacije i model korisničkog interfejsa prilagođen tehnologiji implementacije. Svaki od modela je realizovan u obliku UML profila.

5.1. *Modelovanje konteksta interakcije između čoveka i računara*

Jezik UML je projektovan tako da u svojoj osnovnoj ne zadovoljava specifične slučajeve korišćenja softvera. Drugim rečima, njegova semantika ne može da obuhvati koncepte iz svih domena i arhitektura. Zbog toga postoje formalni mehanizmi proširenja izvorne semantike UML jezika. Izbor

načina proširenja (ili prilagođavanja) jezika zavisi od prirode konkretnog domena i predviđenog načina korišćenja proširenog modela. Na taj način, sa stanovišta domena upotrebe, govorimo o kreiranju domenski specifičnog jezika *(eng. Domain Specific Language- DSL).* DSL obezbeđuje rečnik termina koji omogućava opisivanje problema iz datog domena i čini ih razumljivim od strane domenskih eksperata. Na taj način se podstiče kvalitet i produktivnost razvoja, održavanje sistema i njegova ponovna upotrebljivost. Sa druge strane, kreiranje DSL ima svoju cenu u pogledu dizajna, implementacije i održavanja. Pored toga, mora se voditi računa o balansu između domenski-specifičnih i opštih konstrukcija jezika. Sa obzirom da je naše opredeljenje proširenje UML jezika sa osloncem na MDA arhitekturu, definisanje DSL za razvoj kontekstno-osetljivih korisničkih interfejsa se svodi na proširenje semantike koncepata smeštenih u odgovarajućim modelima jezika visokog nivoa asptrakcije.

U kontekstu UML jezika i MOF standarda, možemo razlikovati dva načina proširenja jezika UML [Brucko8] :

- "*Lako proširenje*" (*eng. Lightweight extension*) koje se svodi na mehanizam UML profila kod kojeg se izvorni, referentni UML metamodel koji se proširuje ne može modifikovati (ne mogu se dodavati nove metaklase u postojeće hijerarhije ili menjati standardne definicije metaklasa). Drugim rečima, postojeći metakoncepti iz metamodela se mogu proširivati na specifičan način kako bi se prilagodili specifičnom domenu.

- Mehanizmi proširenja prvog reda (*eng. first-class extension mechanism*) predstavljaju mehanizme proširenja na MOF nivou. Kako se MOF koncepti koriste za kreiranje metamodela, ovi mehanizmi omogućavaju kreiranje novih metamodela. Kod ovih mehanizama ne postoje ograničenja u pogledu modifikacije postojećih metamodela (mogu se menjati definicije i relacije metakoncepata po potrebi). Ovde se mogu uočiti dve varijante proširenja:

    - *Middleweight* varijanta podrazumeva proširivanje specijalizacijom UML metatipova.

    - *Heavyweight* varijanta podrazumeva kreiranje sopstvenih domenski specifičnih tipova (koncepata) bez specijalizacije tipova referentnog metamodela.

Sa jedne strane, menanizmi proširenja prvog reda nude veću fleksibilnost u pogledu kreiranja i korišćenja DSL. Sa druge strane, oni su skuplji za razvoj i održavanje. Pored toga, *Middleweight* proširenje zavisi od specifične verzije UML jezika, dok se kod *Heavyweight* varijante gubi interoperabilnost sa drugim UML alatima jer imaju sopstveni metamodel. Zbog prethodno navedenih razloga, a u skladu sa principima navedenim na početku poglavlja opredelili smo se za proširenje jezika za modelovanje korišćenjem mehanizma UML profila.

Mehanizam UML profila je uveden sa ciljem proširenja standardnog UML-a. UML 1.1 standard koristi stereotipove i pridodane vrednosti kao tekstualna proširenja koja mogu biti pridodata UML elementima na proizvoljan način. U narednim verzijama jezika, mehanizam profila je unapređivan preciznijim definisanjem strukture i semantike stereotipa i pridodanih vrednosti. Na taj način je u verziji 2.0 profil definisan kao specifična tehnika metamodelovanja, gde su steretipovi specifični

metaklase, pridodane vrednosti standardni metaatributi, dok su profili specifične vrste paketa [UML10]. Mehanizam profila je integrisan u UML metamodel i specificira tehnike proširenja postojećih UML metakoncepata u kontekstu specifičnih implementaciotnih platformi ili domena. Mehanizam je konsistentan sa OMG MOF standardom. Dakle, osnovna namena profila je adaptacija postojećeg metamodela specifičnom domenu, platformi ili metodi. Svaka ovakva adaptacija je grupisana u odgovarajući profil. Pri tome je očuvana izvorna semantika UML metamodela. Postojećim UML metakonceptima se mogu dodavati ograničenja specifična za dati domen. Primena određenog profila u modelovanju konkretnog domena se svodi na primenu stereotipa definisanih u profilu na pojedinačne koncepte domena. Po primeni stereotipa, za dati element će važiti sva pravila definisan odgovarajućim stereotipom. Ova pravila se mogu definisati korišćenjem OCL jezika.

Sa stanovišta modelski zasnovanog razvoja na osnovu profila spomenućemo dva koncepta o kojima ćemo detaljnije govoriti u narednom poglavlju. Kao podrška zvaničnom standardu, a u cilju unapređenja modelski zasnovanog razvoja korišćenjem profila, EMF okruženje integriše dva mehanizma:

- Statička definicija profila - statički definisani profili omogućavaju generisanje programskog kôda modela profila.
- OCL integracija - podrazumeva korišćenje OCL jezika za definisanje ograničenja ili skeletona tela metoda koja se mogu preslikati u kôd. Validacija ograničenja i generisanje kôda je moguće posle primene stereotipa.

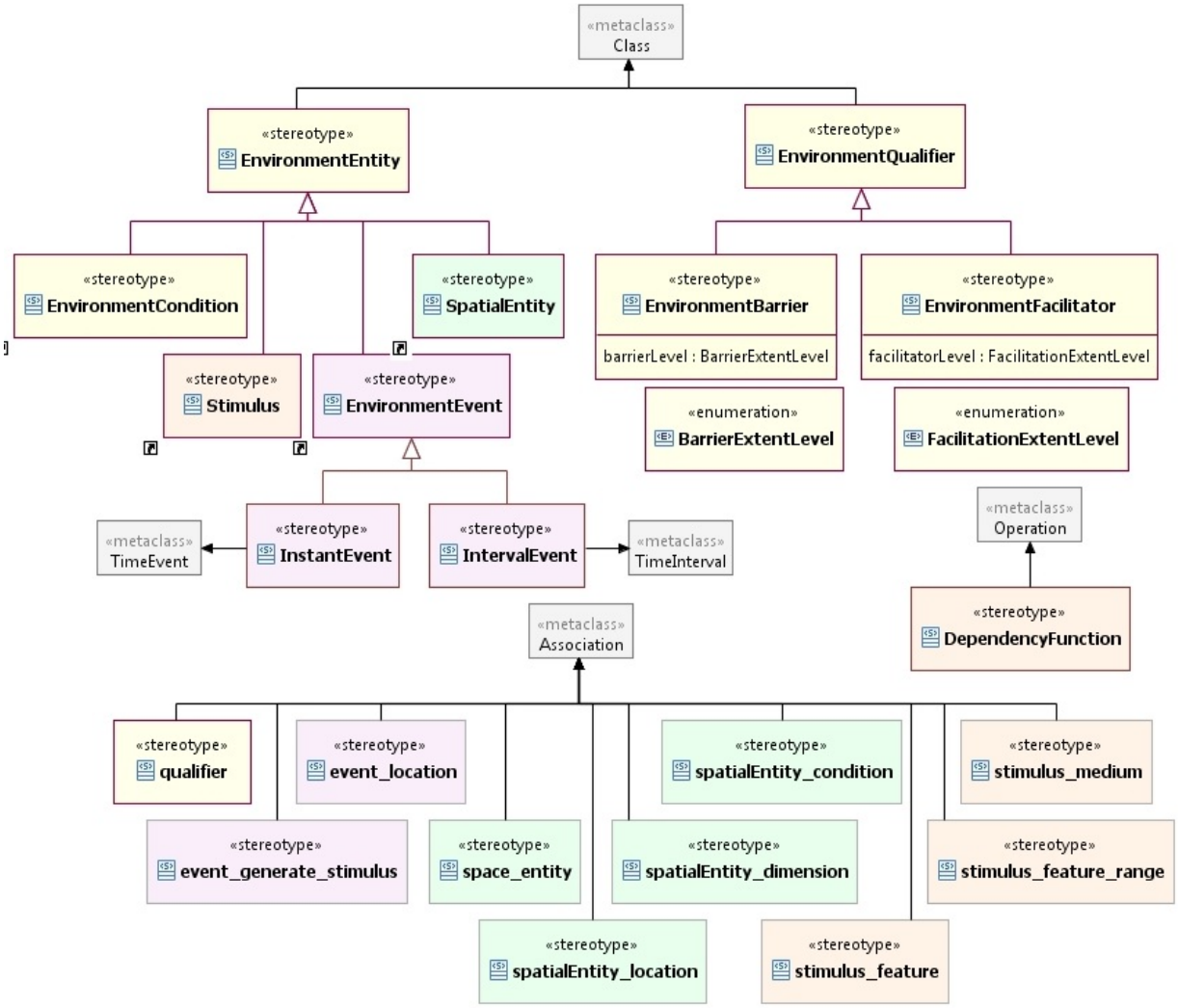
U nastavku dajemo opise ključnih elemenata i primere korišćenja profila kontekstno-osetljive interakcije čoveka i računara. Radi boljeg razumevanja i preglednosti, u primerima su prikazani važniji elementi modela i primenjenih profila.

5.1.1. Modeli okruženja

Na osnovu generičkog modela okruženja (poglavlje 4.2.1) definisali smo proširenja UML metakoncepata. Osnovni koncepti modelovanja okruženja predstavljaju stereotipe klasa. Veze između koncepata okruženja mogi biti izražene korišćenjem relacija nasleđivanja i asocijacije. Korišćenjem relacije nasleđivanja iz osnovnih koncepata su izvedene odogavarajuće specijalizacije. Asocijacija omogućava uspostavljanje semantičkih relacija između koncepata (na primer, svaki prostorni entitet ima određene dimenzije i lokacije). Agregacija, specifičan slučaj relacije asocijacije, omogućava definisanje strukturnih relacija. Tako je, na primer, moguće definisati da je prostorni entitet sastavni deo nekog šireg prostornog entiteta. Metaatributi koncepata okruženja će se pojaviti kao pridodane vrednosti u modelima u kojima se odgovarajući profil primenjuje.

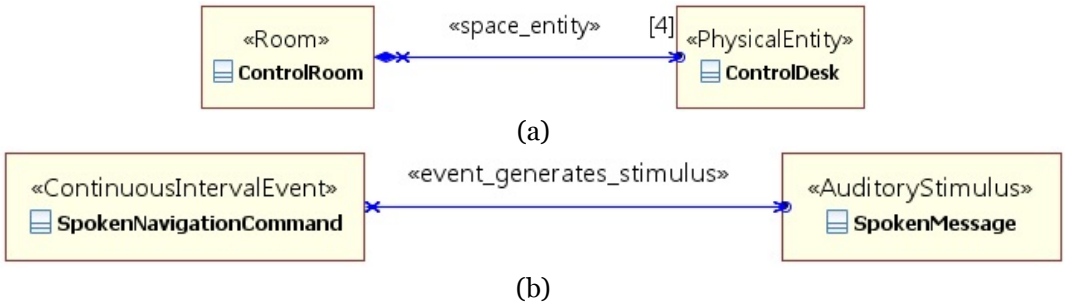
Na slici 5.1 dat je pojednostavljen pregled osnovnih koncepata modela okruženja. Koncepti entiteta prostora, uslova u okruženju, stimulansa i događaja u okruženju modelovani su kao strukturna proširenja. Vremenska dimenzija osnovnih tipova događaja je modelovana proširenjem UML apstrakcija vremena. Izvedeni tipovi događaja i pravila vezana za njih su opisana generički modelom (poglavlje 4.2.2.3). Relacije između koncepata su modelovane kao proširenja relacije asocijacije. Tako

je na primer relacija između događaja sa jedne strane, i lokacije i stimulansa koji generiše sa druge strane, modelovana kao proširenje asocijacije, tj. *event_location* i *event_generate_stimulus*, respektivno. Krajevi definisanih proširenja asocijacija određuju metaatribute koncepata koje povezuju. Ovi metatributi će se prilikom primene profila pojaviti kao pridodane vrednosti. Pored ovih metaatributa, postoje i metaatributi koji su isključivo svojstvo odgovarajućeg entiteta (kao što su atributi lokacije). Zbog preglednosti, metaatributi koncepata nisu prikazani na slici.



Slika 5.1. *Pojednostavljen prikaz osnovnih koncepata okruženja kao proširenja UML metakoncepata*.

Jednostavni primeri na slici 5.2 ilustruju upotrebu predloženih proširenja.



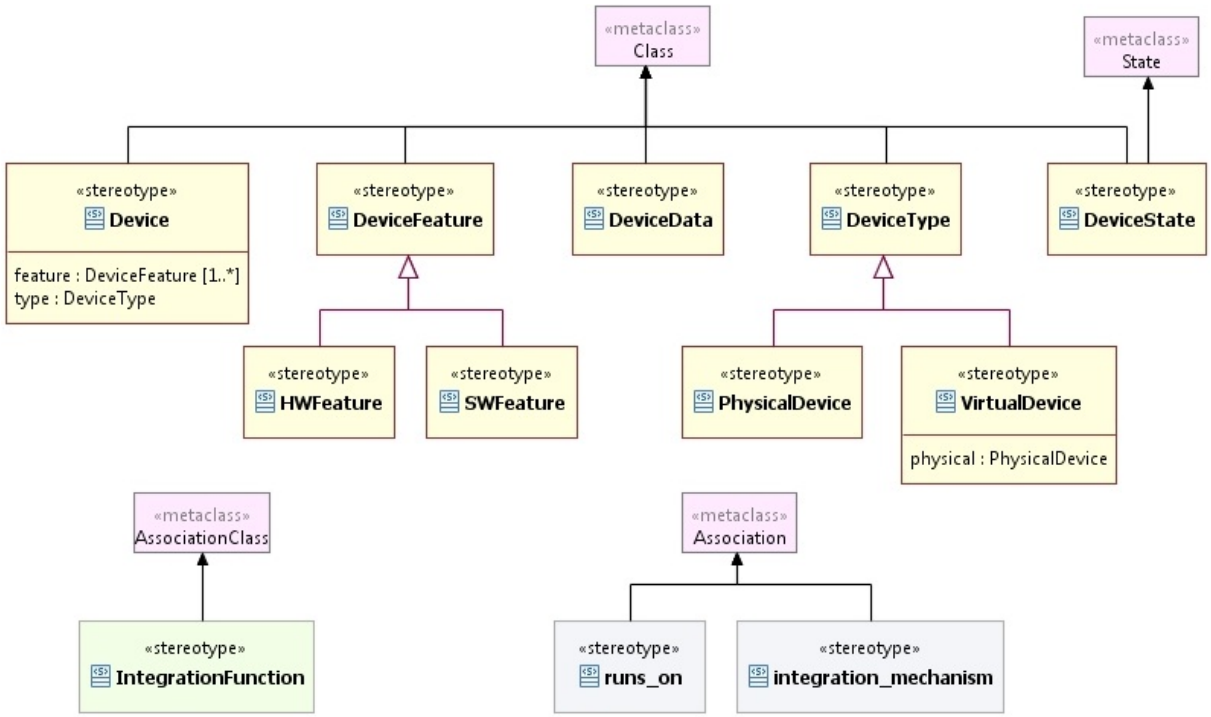
Slika 5.2. *Primena predloženih proširenja (a) opis prostornih entiteta, (b) opis događaja u okruženju*

Na slici 5.2a dat je primer modelovanja strukture kontrolne prostorije (kao što je kontrolna prostorija za upravljanje misijama bespilotne letelice), dok slika 5.2b modeluje glasovnu komandu kao kontinualni intervalni događaj koji generiše govornu poruku kao zvučni stimulans.

5.1.2. Modeli računarske platforme

Profili računarskih uređaja su kreirani na osnovu generičkih modela uređaja (poglavlje 4.2.3). Kreirana su dva UML profila, profil ulaznog uređaja i profil izlaznog uređaja. Ulazno-izlazni uređaji predstavlja kombinaciju dva tipa te se ovi uređaji modeluju korišćenjem koncepata iz definisanih profila. Važno je napomenuti da se korišćenjem predloženih profila računarski uređaji modeluju na aplikativno generičkom nivou, tj. kao klase uređaja. Na osnovu njih mogu biti izvedeni modeli specifičnih uređaja.

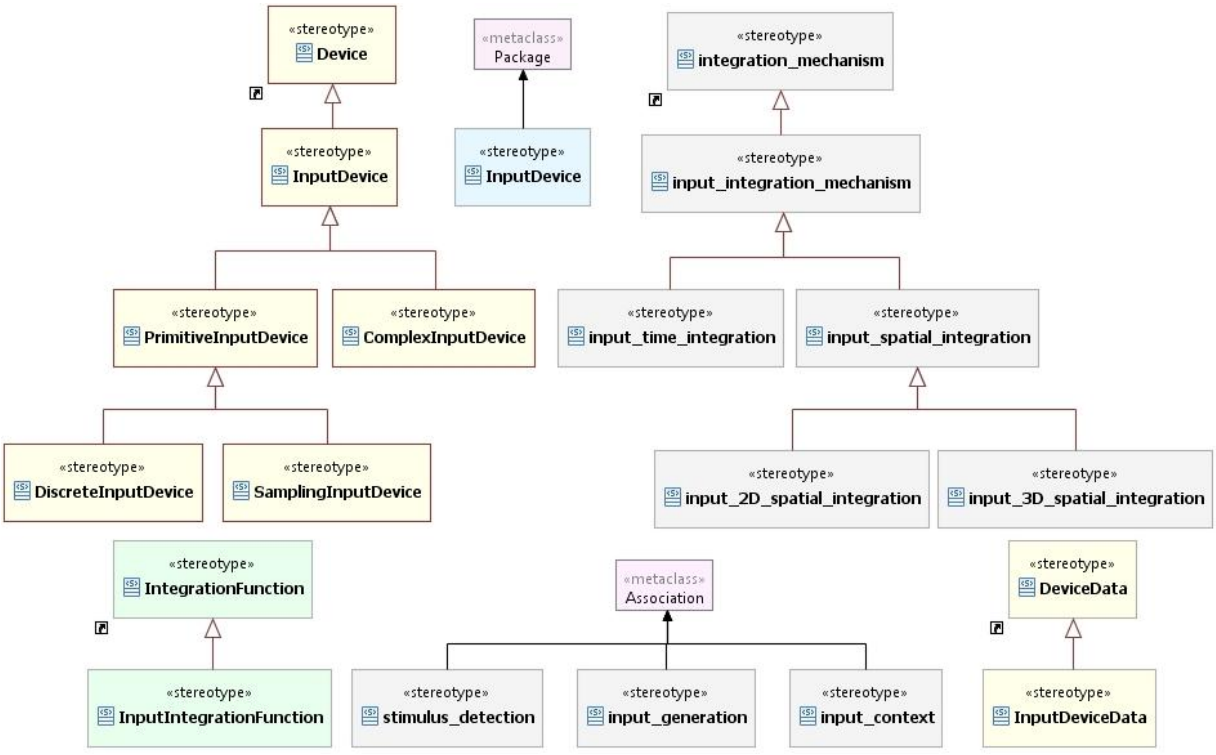
Na slici 5.3 dat je pojednostavljen pregled osnovnih koncepata zajedničkih za sve tipove uređaja. Sam koncept uređaja, osobina uređaja, struktura podataka sa kojima uređaj radi, tip uređaja i unutrašnji kontekst rada uređaja (stanje) definisani su kao steretipovi klasa. Korišćenjem relacije generalizacije izvršena je njihova specijalizacija, pa tako osobine uređaja mogu biti biti hardverske i softverske, dok je tip uređaja sa aspekta virtuelizacije klasifikovan kao fizički ili virtuelni. Koncept stanja je dodatno proširen i predstavljen i kao stereotip UML koncepta stanja. Na ovaj način je objedinjen opis strukturnih i dinamičkih aspekata unutrašnjeg konteksta računarskog uređaja. Semantika relacije između virtuelnog i fizičkog uređaja definisana je stereotipom asocijacije *runs_on*. Računarski uređaj u opštem slučaju ima složenu strukturu, te je semantika integracije jednostavnih uređaja u složenu celinu modelovana kao stereotip ascocijacije *integration_mechanism*. Semantika ove asocijacije može biti dodatno proširena funkcijom integracije, tj. stereotipom asocijacione klase *IntegrationFunction*.



Slika 5.3. *Pojednostavljen prikaz osnovnih koncepata uređaja kao proširenja UML metakoncepata.*

5.1.2.1. Model ulaznog uređaja

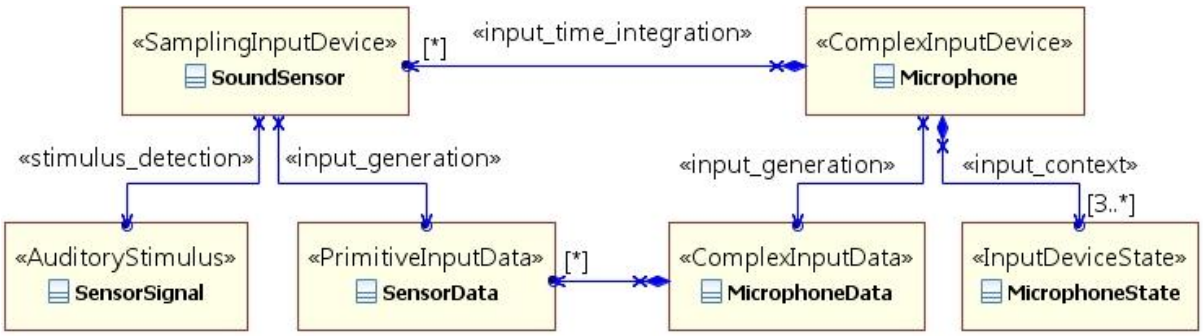
Ulazni uređaj detektuje različite oblike fizičkih signala - stimulansa u okruženju i pretvara ih u računarski čitljiv oblik. Koncept stimulansa je opisan u modelu okruženja. Za modelovanje ulaznih uređaja koriste se proširenja paketa, klasa, asocijacija, asocijacionih klasa i atributa (slika 5.4). Zbog preglednosti, proširenja atributa modela ulaznog uređaja nisu prikazana. Proširenje koncepta paketa omogućava logičko grupisanje koncepata koji definišu strukturu ulaznog uređaja kao jedinstvene celine. Proširenja klase opisuju osnovne koncepte ulaznih uređaja (kao što su tipovi i podaci ulaznog uređaja). Proširenja asocijacija se koriste za opise semantike strukturnih relacija (*input _integration_mechanism*) između ulaznih uređaja, kao i za opise semantike relacija sa drugim konceptima. Na taj način stereotip *stimulus_detection* opisuje relaciju između uređaja i stimulansa koji detektuje, stereotip *input_generation* povezuje ulazni uređaj sa podacima koje generiše, dok je stereotip *input_context* deo modela stanja ulaznog uređaja. Relacije specijalizacije između tipova ulaznih uređaja uspostavljene su korišćenjem generalizacije. U nekim slučajevima, kao što je detaljnije opisivanje semantike integracije ulaznih uređaja, definisana proširenja strukturnih relacija mogu biti preciznije opisana proširenjima asocijacionih klasa (*InputIntegrationFunction*).



Slika 5.4. *Osnovni oblici UML proširenja za opisivanje ulaznih uređaja.*

Na slici 5.5 je prikazan model mikrofona kreiran korišćenjem predloženih proširenja. Mikrofon je modelovan kao složen ulazni uređaj koji integriše skup zvučnih senzora. Integracija ima vremensku semantiku. Senzor zvuka je modelovan kao uzorkujući ulazni uređaj koji sa određenom frekvencijom neprekidno uzorkuje zvučni signal u svom okruženju i konvertuje ga u računarski čitljive primitivne strukture podataka. Mikrofon integriše generisane primitivne podatke u složenu strukturu koja se

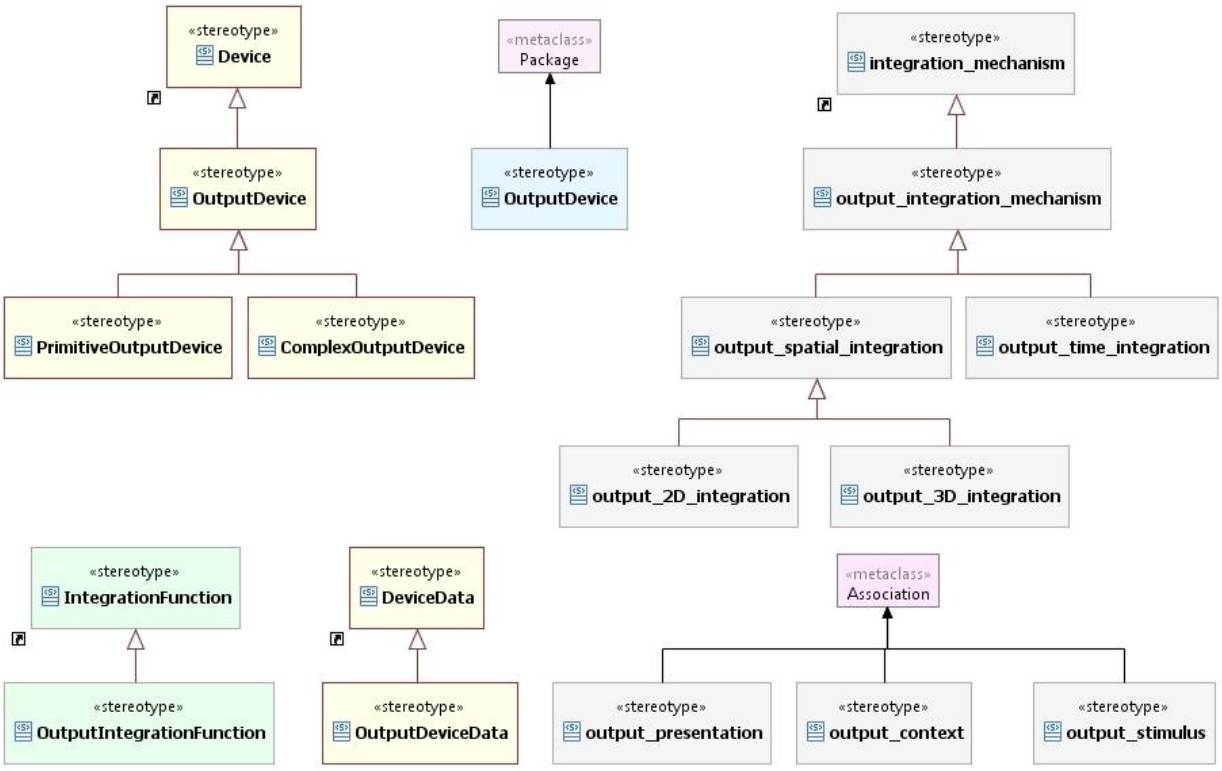
prosleđuje računaru na dalju obradu. Unutrašnji kontekst mikrofona kao uređaja je modelovan skupom stanja u kojima se može naći.



Slika 5.5. *Modelovanje mikrofona korišćenjem predloženih proširenja.*

5.1.2.2. Model izlaznog uređaja

Izlazni uređaj prezentuje određenu strukturu podataka i formira fizički signal koji čovek ili drugi uređaj može registrovati. Za modelovanje izlaznih uređaja koriste se proširenja paketa, klasa, asocijacija, asocijacionih klasa i atributa (slika 5.6).

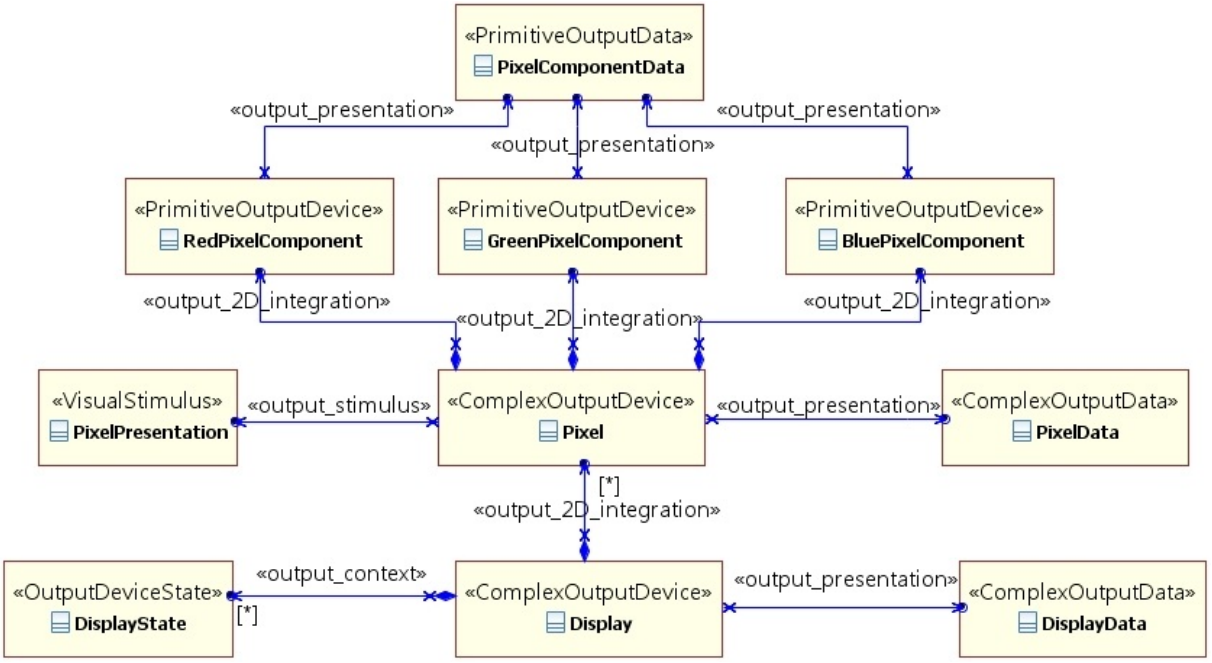


Slika 5.6. *Osnovni oblici UML proširenja za opisivanje izlaznog uređaja.*

Zbog preglednosti, proširenja atributa modela izlaznog uređaja nisu prikazana. Proširenje koncepta paketa omogućava logičko grupisanje koncepata koji definišu strukturu izlaznog uređaja kao jedinstvene celine. Proširenja klase opisuju osnovne koncepte izlaznih uređaja (kao što su tipovi i podaci izlaznog uređaja). Proširenja asocijacija se koriste za opise semantike strukturnih relacija

(*output _integration_mechanism*) između izlaznih uređaja, kao i za opise semantike relacija sa drugim konceptima. Na taj način stereotip *output_stimulus* opisuje relaciju između izlaznog uređaja i stimulansa koji generiše, stereotip *output_presentation* povezuje uređaj sa podacima koje prezentuje, dok je stereotip *output_context* deo modela stanja izlaznog uređaja. U nekim slučajevima, kao što je detaljnije opisivanje semantike integracije izlaznih uređaja, definisana proširenja strukturnih relacija mogu biti preciznije opisana proširenjima asocijacionih klasa (*OutputIntegrationFunction*).

Na slici 5.7 dat je primer modelovanja fizičkog displeja pomoću predloženih proširenja. Displej se modeluje kao dvodimenzionalna matrica piksela. Piksel predstavlja složen izlazni uređaj koji integriše tri primitivna uređaja: crvenu komponentu piksela, zelenu komponentu piksela i plavu komponentu piksela. Svaki od primitivnih uređaja ima pridruženu strukturu podataka koja opisuje frekvenciju i intenzitet svetlosti koju emituje. Piksel kao složen izlazni uređaj spaja svetlosne kanale koje generišu komponente i emituje svetlosni stimulans koji čovek može detektovati. Samom pikselu je pridružena struktura podataka koja predstavlja agregaciju podataka pridruženih komponentama. Displej kao složen izlazni uređaj integriše matricu piksela kao složenu strukturu podataka. Pored toga, displej je opisan je skupom stanja u kojima se može naći i koja određuju unutrašnji kontekst računarskog uređaja.



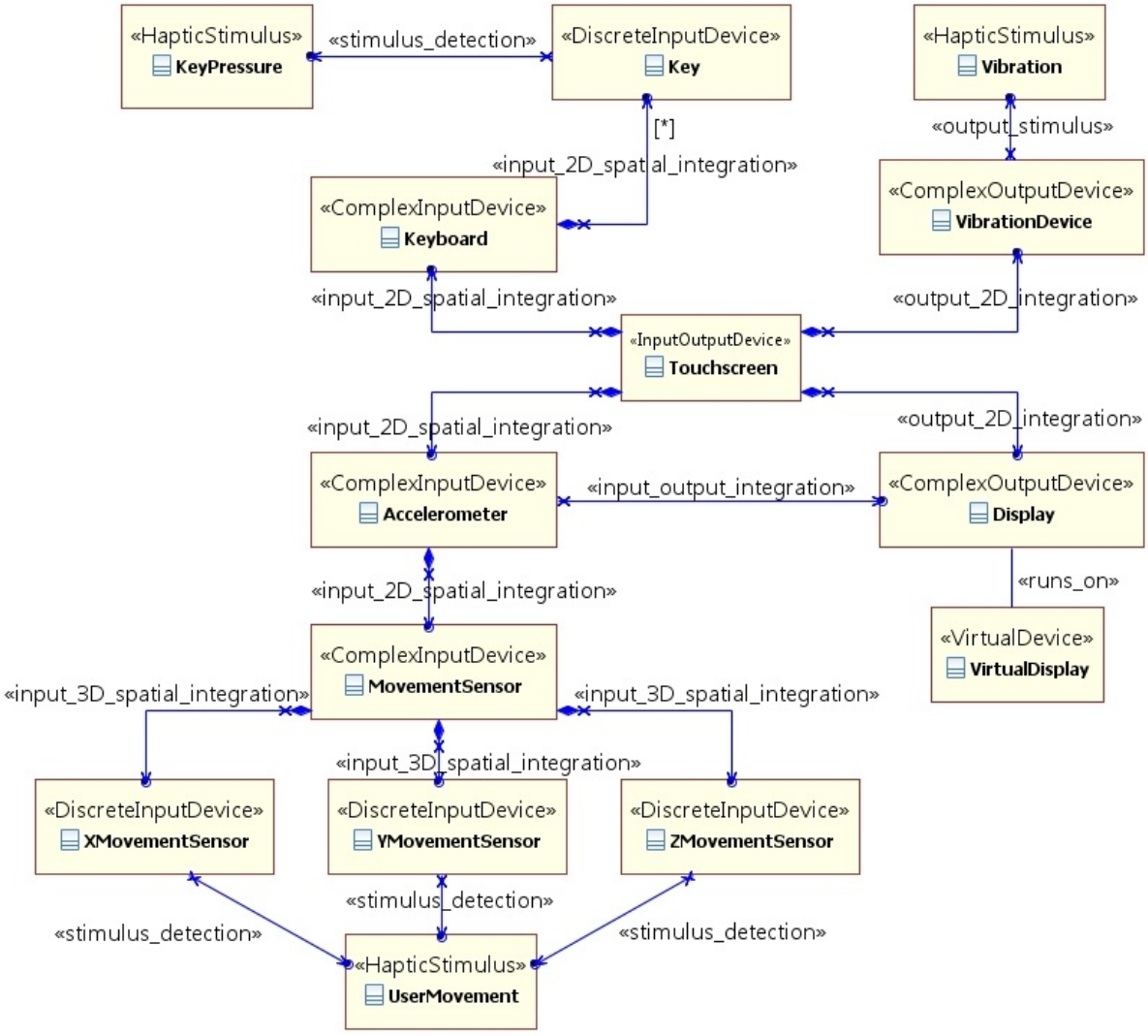
Slika 5.7. *Pojednostavljen generički model displeja kao izlaznog uređaja.*

5.1.2.3. Model ulazno-izlaznog uređaja

Kao primer modela ulazno-izlaznog uređaja izabrali smo ekran osetljiv na dodir (*eng. touchscreen*) kao tehnologiju koja nalazi široku primenu u različitim komercijalnim proizvodima. Ova vrsta displeja detektuje dodirni stimulans u granicama samog displeja i prvenstveno generiše vizuelni prikaz kao rezultat akcije korisnika. Sa stanovišta interakcije sa korisnikom, ekran osetljiv na dodir možemo posmatrati kao integraciju ulaznog uređaja osetljivog na dodir i displeja kao izlaznog uređaja. Izlazni

stimulans je primarno vizuelan, ali može biti i višenačinski uključivanjem zvučnih signala (kao što su glasovna obaveštenja) ili dodirnih signala (na primer vibracija).

Na slici 5.8 je predstavljen generički model displeja osetljivog na dodir. Zbog preglednosti na dijagramu klasa su prikazani samo najvažniji koncepti uređaja. Osnovni ulazni uređaj predstavlja tastatura koja fizički preklapa displej i sastoji se od prostorno raspoređenih tastera koji detektuju pritisak kao kinestetički dodirni stimulans. Osnovni izlazni uređaj predstavlja displej čiji je model opisan u prethodnom odeljku. Neki od uređaja koji koriste ekran osetljiv na dodir, kao što su na primer *BlackBerry* pametni telefoni, mogu integrisati i vibrator koji generiše dodirni stimulans kao rezultat akcije korisnika, kao i akcelerometar (*eng. accelerometer*). Akcelerometar je uređaj koji detektuje položaj i orijentaciju pametnog telefona. Kada korisnik menja položaj i orijentaciju uređaja, akcelerometar detektuje pokret u trodimenzionalnom prostoru duž x, y i z ose pomoću senzora pokreta. Senzor pokreta je modelovan kao složen ulazni uređaj koji integriše tri diskretna senzora za svaku osu trodimenzionalnog koordinatnog sistema. Na ovaj način promena orijentacije uređaja utiče na oblik prikaza displeja koji se može menjati između uspravnog (*eng. portrait*) i položenog (*eng. landscape*).



Slika 5.8. *Pojednostavljen generički model displeja osetljivog na dodir.*

5.1.3. Modeli čoveka

Na osnovu predloženog modela čoveka opisanog u prethodnom poglavlju definisali smo proširenja jezika UML kako bi se uvedeni koncepti mogli koristiti u procesu razvoja korisničkih interfejsa. Predložena proširenja omogućavaju kreiranje modela čoveka na različitim nivoima apstrakcije gde možemo razlikovati:

- *Kontekstno generičke modele čoveka*. Ovo su modeli visokog nivoa asptrakcije kreirani korišćenjem koncepata ICF klasifikacije koji su zajednički za sve ljude i nisu vezani za kontekst ili domen korišćenja softverskog sistema.

- *Kontekstno specifične modele čoveka*. Ovo su modeli specifični za određeni kontekst ili domen korišćenja. Oni su izvedeni iz generičkih modela uvođenjem kriterijuma za klasifikaciju korisnika koji je relevantan za određeni domen, kao što je na primer tip inteligencije u edukativnim igrama ili tip učenja (kognitivni stil) u slučaju operatora bespilotne letelice. Kriterijum će odrediti upotrebu relevantnih koncepata iz generičkih modela.

Ovde je važno napomenuti da se reč kontekst u ovom slučaju odnosi prvenstveno na domen korišćenja softverskog sistema. Kontekst u širem smislu određuje kombinacija modela čoveka, okruženja, uređaja i interakcije.

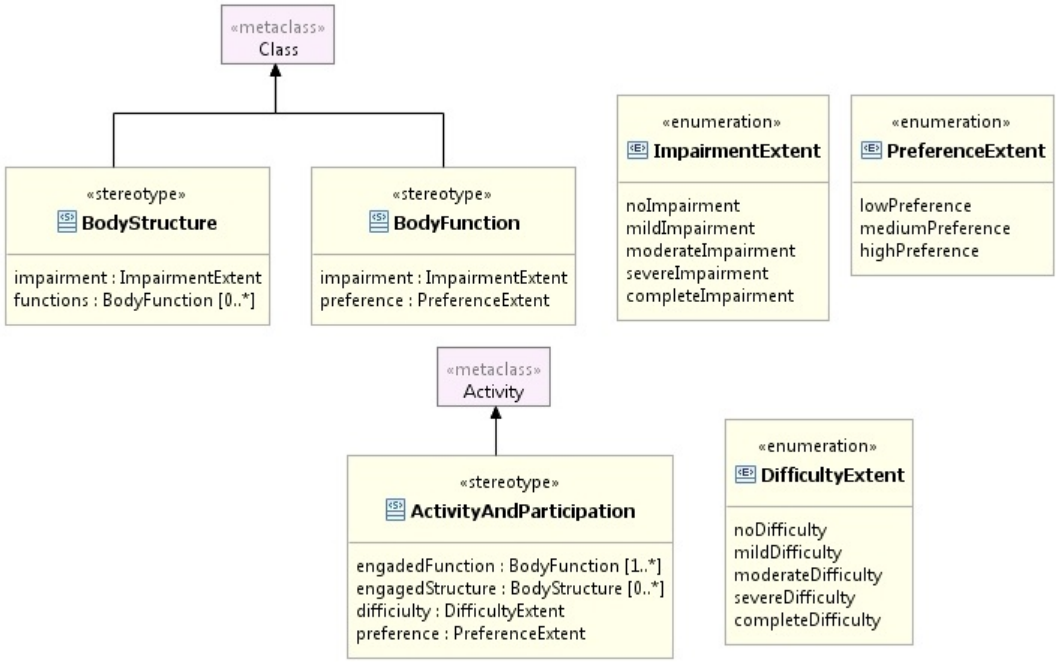
Kada govorimo o samom postupku modelovanja korisnika, možemo uočiti dva principa [Brusilovsky07]:

- Modelovanje korisnika zasnovano na osobinama (*eng. feature-based user modeling*) predstavlja dominantan način modelovanja korisnika u adaptivnim softverskim sistemima. Ova vrsta modela modeluje specifične osobine korisnika kao što su znanje, interesovanja, ciljevi i dr. U toku interakcije korisnika sa sistemom ove osobine se menjaju, pa je cilj modela da prati i odražava ažurno stanje korisnika, tj. skupa karateristika koje se modeluju.

- Modelovanje korisnika pomoću stereotipa (*eng. stereotype user modeling*) predstavlja jedan od najranijih pristupa modelovanju korisnika [Rich79]. Stereotipski modeli korisnika predstavljaju predefinisane šablonske modele. Postupak modelovanja korisnika se svodi na učlanjivanje korisnika koji se modeluje u odgovarajuću grupu, tj. stereotip ili šablon. Na ovaj način se korisnici koji pripadaju jednom stereotipu tretiraju na identičan način u mehanizmima adaptacije. U ovim sistemima korisnik je predstavljen kao tekući stereotip (tj. kao grupa kojoj trenutno pripada).

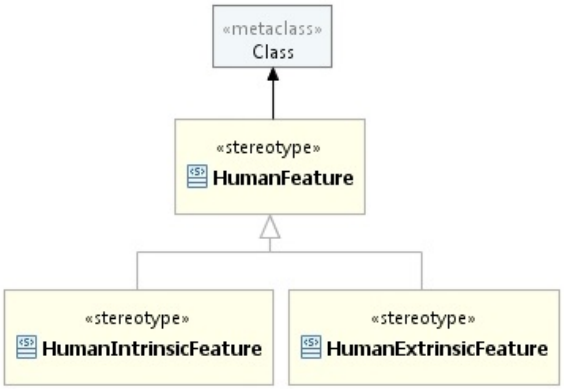
Predloženi pristup modelovanja korisnika omogućava kombinaciju dve opisane tehnike. Stereotipski modeli vezani za određeni domen ili anatomiju i fiziologiju čoveka se mogu kreirati na osnovu odgovarajućih testova. Formirani steretipovi se dalje mogu koristiti za inicijalizaciju modela zasnovanih na karakteristikama. Osnovna prednost ovakvog pristupa se ogleda u mogućnosti efikasnog uvođeja novog korisnika u sistem. Pored toga, integracija jasno definisanih šablonskih modela sa promenjivim modelima karakteristika čoveka može unaprediti proces adaptacije softverskog sistema konkretnom korisniku.

Profil za modelovanje čoveka kreiran je na osnovu modela čoveka opisanog u odeljku 4.2.1. Zbog preglednosti, na slici 5.9 su prikazana proširenja entiteta čoveka vezana za osnovne koncepte ICF klasifikacije. Elementi modelovanja anatomije (*BodyStructure*) i fiziologije (*BodyFunction*) čoveka proširuju UML metakoncept klase. Stepen ograničenja u strukturama i funkcijama čoveka opisan je pridodanim vrednostima *impairment*. Vrednosti su generičke prirode i nabrojivog tipa u skladu sa ICF klasifikacijom. Aktivnosti vezane za angažovanje anatomskih i fizioloških mehanizama (*ActivityAndParticipation*) su modelovane kao proširenja UML koncepta aktivnosti. Svaka aktivnost zahteva angažovanje specifičnih struktura (*engagedStructure*) i funkcija (*engagedFunction*) čoveka. Stepen ograničenja u aktivnostima (*difficulty*) može biti određen ograničenjima struktura i funkcija čoveka, kao i spoljnim faktorima. Ovo svojstvo je definisano u ICF klasifikaciji. Radi preciznijeg opisa funkcija i aktivnosti čoveka sa stanovišta individualnih sklonosti ka njihovom korišćenju predlažemo uvođenje dodatnog atributa *preference* čiji je skup vrednosti određen nabrojivim tipom *PreferenceExtent*.



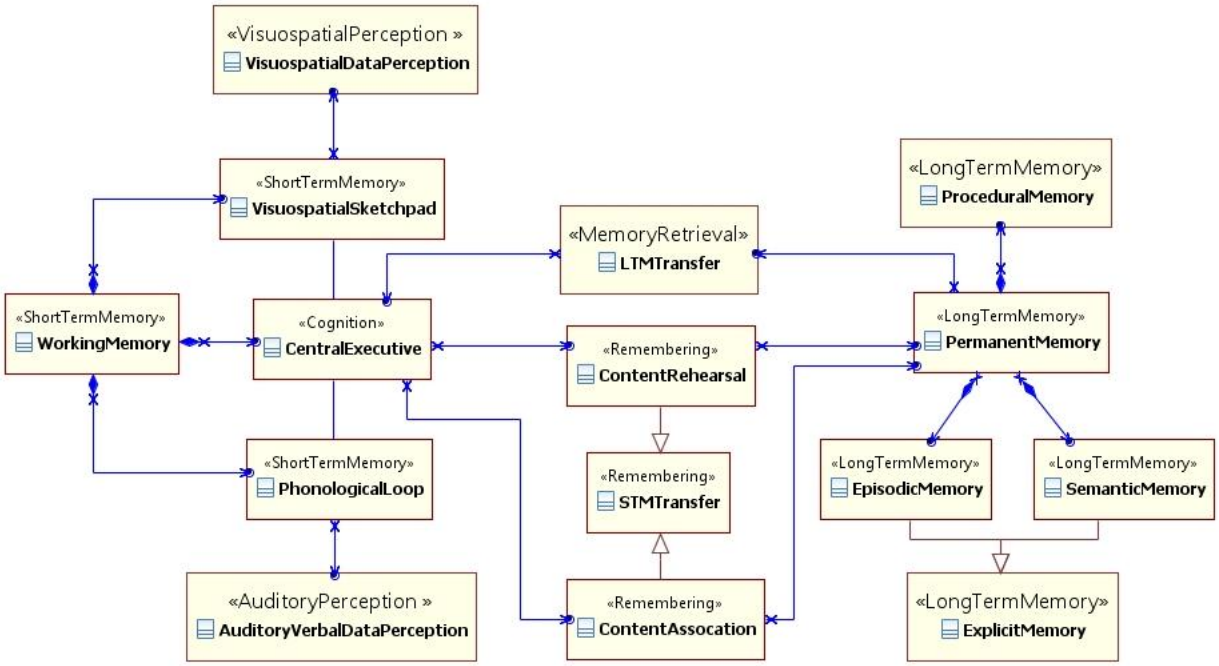
Slika 5.9. *ICF entiteti čoveka kao proširenja odgovarajućih UML apstrakcija.*

Elementi modelovanja svojstava čoveka predstavljaju strukturna UML proširenja (slika 5.10).



Slika 5.10. *Osnovna UML proširenja za opis svojstava čoveka.*

Kao primer korišćenja predloženih proširenja opisujemo model radne memorije čoveka [Baddeley03]. Na slici 5.11 opisana je struktura modela radne memorije čoveka korišćenjem UML proširenja za opis ljudske memorije iz ICF klasifikacije [Jovanović12]. Definisana proširenja opisuju opšte koncepte memorije čoveka.



Slika 5.11. *Model radne memorije čoveka [Baddeley03] korišćenjem proširenja modela entiteta čoveka.*

Koncept radne memorije predstavlja glavnu teorijsku osnovu teorije kognitivnog naprezanja (*eng. Cognitive Load Theory*) u oblasti interakcije čoveka i računara [Oviatt06]. Kognitivno naprezanje u širem smislu podrazumeva raspoložive mentalne resurse čoveka za rešavanje problema ili izvršavanje zadataka u datom vremenu. U oblasti interakcije čoveka i računara kao kritični faktori kognitivnog naprezanja izdvojeni su ograničena pažnja i radna memorija.

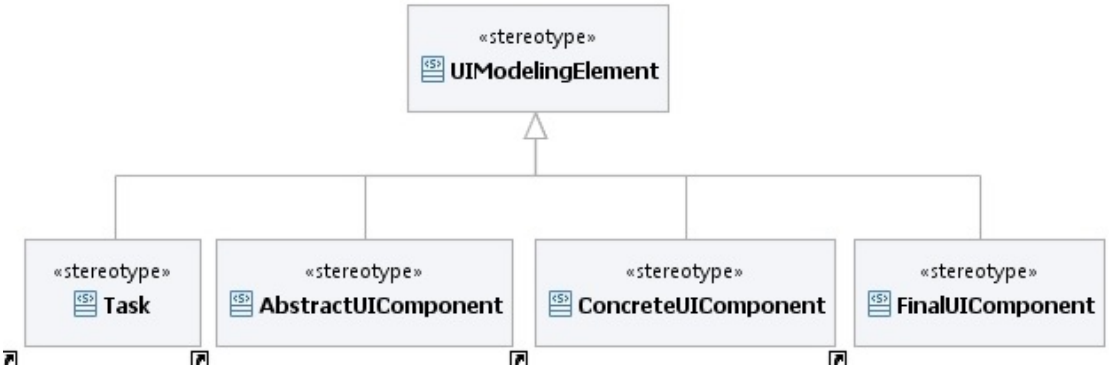
Teorija radne memorije posmatra kratkoročnu memoriju (*eng. short-term memory - STM*) kao sintezu nezavisnih celina povezanih sa specifičnim načinima komunikacije. Vizuelnoprostorna memorija (*eng. visuospatial sketchpad*) predstavlja oblast radne memorije koja skladišti vizuelne podatke kao što su slike i dijagrami. Fonološka petlja (*eng. phonological loop*) je oblast koja čuva audio-verbalne podatke. Ove dve celine povezuje centralna obradna jedinica (*eng. central executive*). Iako koordinaciju dve celine vrši obradna jedinica, u pogledu obrade modaliteta niskog nivoa one u velikoj meri funkcionišu nezavisno. Na ovaj način se efektivni kapacitet radne memorije povećava u slučaju angažovanja različitih načina komunikacije čoveka i računara u vršenju zadataka. Obradna jedinica vrši kognitivne funkcije kao što su multimodalna integracija, planiranje budućih akcija, proces donošenja odluka, iniciranje dobavljanja informacija iz dugoročne memorije i dr.

Dugoročna memorija (*eng. long-term memory - LTM*) predstavlja trajno skladište informacija. Čine je dve celine – deklarativna (eksplicitna) memorija (*eng. explicit memory*) i proceduralna (implicitna) memorija (*eng. procedural memory*). Deklarativna memorija može biti epizodna (*eng. episodic*) ili

semantička (*eng. semantic*). Epizodna memorija čuva sećanja o specifičnim događajima iz života čoveka kao što su autobiografski događaji. Semantička memorija predstavlja znanje o spoljašnjem svetu i pojavama u njemu. Proceduralna memorija čuva podatke vezane za pokrete ljudskog tela i korišćenje objekata u okruženju kao što je skup radnji vezan za upravljanje automobilom. Pomoću procesa ponavljanja (*eng. rehearsal*) i asocijacije (*eng. association*) sadržaj kratkoročne memorije prelazi u dugoročnu memoriju. Ove dve grupe procesa su modelovane kao stereotip pamćenja (*eng. remembering*). Posebna grupa mentalnih funkcija je zadužena za prebacivanje sadržaja iz dugoročne u kratkoročnu memoriju i one su modelovane stereotipom dobavljanja iz dugoročne memorije (*memory retrieval*).

5.2. *Modelovanje korisničkih interfejsa*

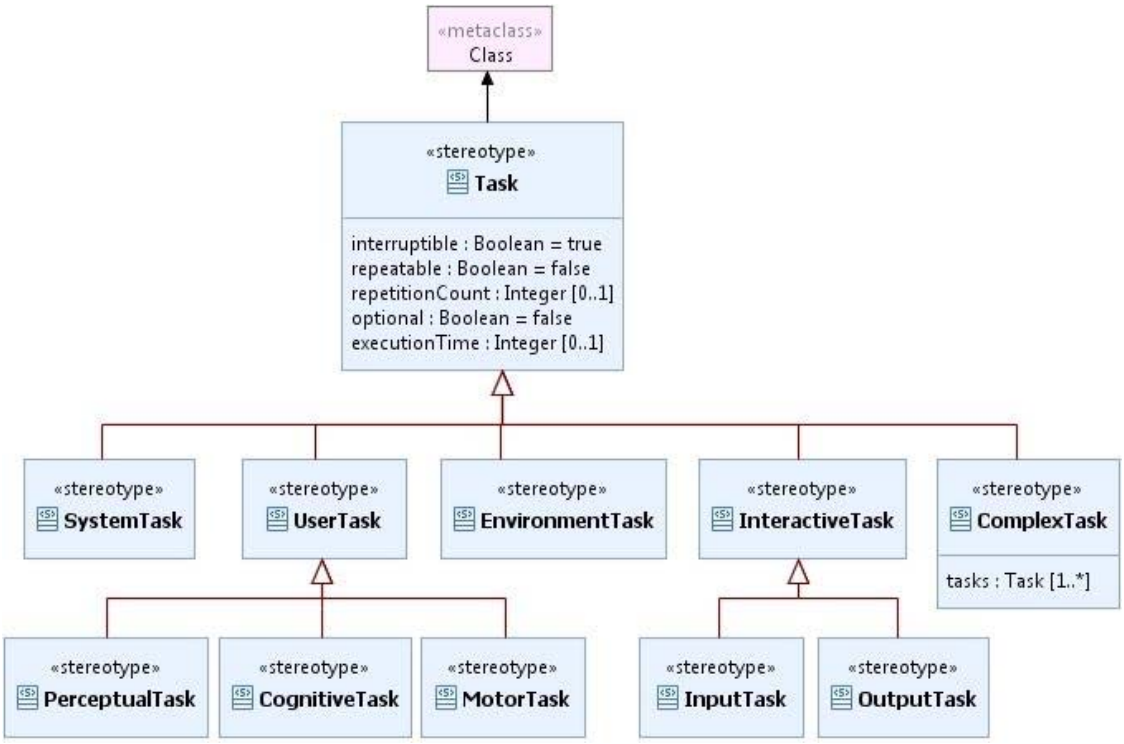
U procesu dizajna korisničkog interfejsa polazi se od modela zadataka. Na osnovu modela zadataka kreira se model korisničkog interfejsa nezavisan od načina komunikacije. Nakon toga se generiše model korisničkog interfejsa zavisan od korišćenih načina komunikacije. Ovaj model se prevodi u model konačnog korisničkog interfejsa koji je prilagođen specifičnoj implementacionoj platformi. Na slici 5.12 prikazani su osnovni koncepti u modelima korisničkih interfejsa. Iz zajedničkog elementa (*UIModelingPrimitive*) izvedene su primitive za modelovanje zadataka (*Task*), apstraktnog korisničkog interfejsa (*AbstractUIComponent*), konkretnog korisničkog interfejsa (*ConcreteUIComponent*) i konačnog korisničkog interfejsa (*FinalUIComponent*). Osnovni motiv za uvođenje prikazane klasifikacija je mogućnost uspostavljanja relacija sa faktorima konteksta u svakom od modela, tj. delu dizajna korisničkog interfejsa.



Slika 5.12. *Osnovni skup primitiva za modelovanje korisničkih interfejsa.*

5.2.1. Model zadataka

Uopšteno posmatrano, modeli zadataka se mogu koristiti u različitim fazama razvoja softvera. U ranim fazama, tj. prikupljanju zahteva i analizi, modeli zadataka specificiraju načine korišćenja korisničkog interfejsa sistema. U tom pogledu, njihova osnovna namena je pregled aktivnosti koje korisnici vrše kada interaguju sa sistemom kako bi postigli određeni cilj. Ovde postoje razvijeni UML formalizmi za opise stanja i aktivnosti [Bergh05, Bergh07]. U fazi dizajna, modeli zadataka mogu poslužiti kao polazna tačka u procesu kreiranja korisničkog interfejsa kroz nekoliko nivoa apstrakcije. U kontekstu korišćenja zadataka u fazi dizajna, uvedena su osnovna UML proširenja kao na slici 5.13.



Slika 5.13. *Osnovni koncepti modela zadataka.*

Zadatak može biti sistemski (*SystemTask*), korisnički (*UserTask*), povezan sa okruženjem (*EnvironmentTask*), interaktivni (*InteractiveTask*) i složen zadatak (*ComplexTask*). Sistemski zadatak je vezan za funkcije koje vrši programski sistem. Zadatak korisnika se odnosi na aktivnosti vezane za čoveka. U zavisnosti od nivoa obrade čoveka, zadatak korisnika može biti percepcijski, kognitivni i motorički. Zadatak okruženja je uveden kako bi se modelovale aktivnosti koje nisu eksplicitno vezane za čoveka ili programski sistem, već se odigravaju kao posledica pojava u okruženju kao što su na primer različiti tipova događaja okruženja. Interaktivni zadatak je povezan sa komunikacijom čoveka i računara. Uzimajući u obzir tokove informacija u komunikaciji on može biti ulazni i izlazni, pri čemu je referentna tačka za određivanje smera tokova računar. Složen zadatak predstavlja agregaciju ostalih tipova i omogućava modelovanje kompleksnih zadataka sastavljenih od različitih tipova jednostavnijih zadataka.

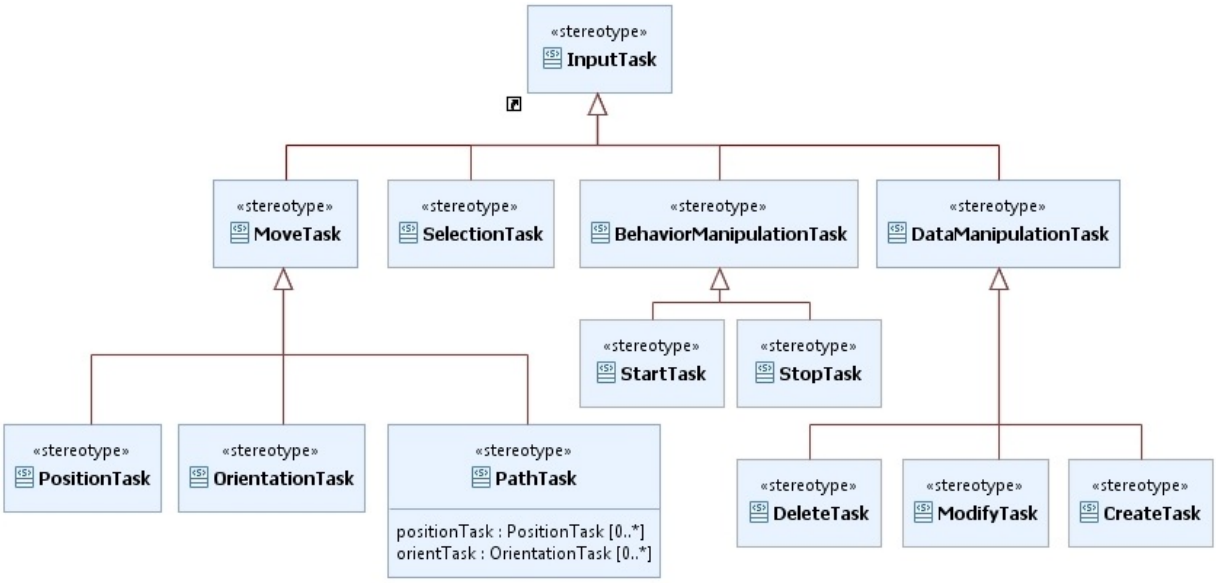
Radi boljeg razumevanja modela zadataka, na slici 5.14 je opisana klasifikacija ulaznih zadataka. Prilikom kreiranja klasifikacije ulaznih zadataka u obzir smo uzeli i proširili postojeću taksonomiju [Constantine3] koja opisuje konkretne tipove akcije koje korisnik može preduzeti kada interaguje sa korisničkim interfejsom.

Zadatak pomeranja (*MoveTask*) opisuje pokret komponente korisničkog interfejsa. Zadatak je generičke prirode i ne zavisi od načina komunikacije. Ovaj zadatak smo dalje klasifikovali na osnovu istraživanja [Foley84] koje predlaže podelu ulaznih grafičkih uređaja na osnovu funkcija koje obavljaju. Iako je klasifikacija kreirana u kontekstu stonog računarstva, ona je korišćena u oblasti sveprisutnog računarstva [Ballagas06]. U skladu sa tim zadatak pomeranja smo dalje klasifikovali u tri podtipa:

- Pozicioni zadatak (*PositionTask*) – specificira poziciju u koordinatama aplikacije. Obično predstavlja deo komande za postavljanje objekta u odgovarajući region. Zadatak može biti kontinualne ili diskretne prirode. Kontinualni pozicioni zadatak predstavlja kontinuiranu promenu pozicije objekta, dok diskretni ulazni zadatak opisuje promenu pozicije objekta po izvršenju zadatka.

- Orijentacioni zadatak (*OrientatiomTask*) – specificira orijentaciju u koordinatnom sistemu. Po analogiji sa pozicionim tipom, može biti kontinualan ili diskretan.

- Zadatak putanje (*PathTask*) – predstavlja serije pozicija i orijentacija u toku vremena tako da uvek ima kontinualnu prirodu.



Slika 5.14. *Tipovi ulaznih zadataka.*

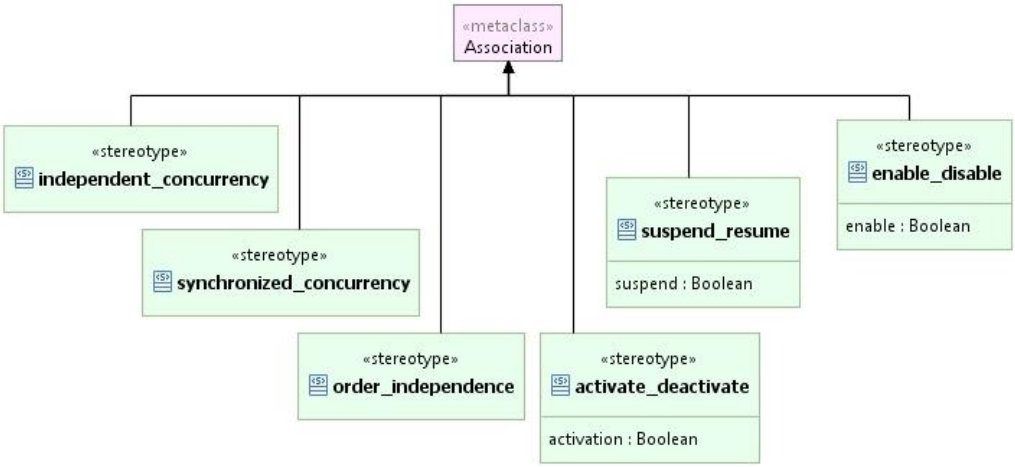
Zadatak selekcije (*SelectionTask*) predstavlja izbor iz skupa ponuđenih opcija kao što je na primer skup komandnih dugmadi ili stavki menija. Kao specifični slučajevi ovog zadatka mogu se posmatrati izbor opcije prepoznavanjem govora ili pokreta. Zadatak manipulacije ponašanja (*BehaviorManipulationTask*) može označavati pokretanje (*StartTask*) ili zaustavljanje (*StopTask*) akcije sistema preko korisničkog interfejsa. Zadatak manipulacije podacima (*DataManipulaitonTask*) predstavlja promenu podataka pridruženih korisničkom interfesju koja može rezultovati u promeni stanja sistema. U opštem slučaju, ova akcija se može javiti u obliku kreiranja (*CreateTask*), brisanja (*DeleteTask*) ili ažuriranja (*ModifyTask*) komponente korisničkog interfejsa. Ovde se zadatak kopiranja može posmatrati kao specifičan slučaj zadatka kreiranja.

Izlazni zadatak (*OutputTask*) se može javiti u obliku prezentovanja ili prenošenja informacija (*PresentationTask*) čoveku korišćenjem određenog načina komunikacije.

Modeli zadataka moraju uzeti u obzir i relacije između zadataka. Za opisivanje relacija između zadataka iskoristili smo postojeću klasifikaciju temporalnih operatora koji definišu vremenski redosled izvršavanja zadataka [Mori02]. Deo operatora je definisan preko skupa pridodanih vrednosti stereotipa *Task*, dok je deo modelovan kao skup proširenja UML asocijacije (slika 5.15).

Za detaljnije opisivanje zadatka uvedene su sledeće pridodane vrednosti (prikazane na slici 5.13):

- *interruptible* – određuje nivo prioriteta izvršavanja zadatka tako što kaže da li može biti prekinut ili suspendovan dok se izvršava;
- *repetable* – govori da li se zadatak izvršava iterativno;
- *repetitionCount* – definiše broj iteracija ukoliko se zadatak izvršava iterativno;
- *optional* – govori da li je izvršavanje zadatka opciono;
- *predictedExecutionTime, perceived ExecutionTime* – predviđeno i efektivno vreme izvršavanja zadatka.



Slika 5.15. *Steretipovi asocijacija za modelovanje relacija između zadataka.*

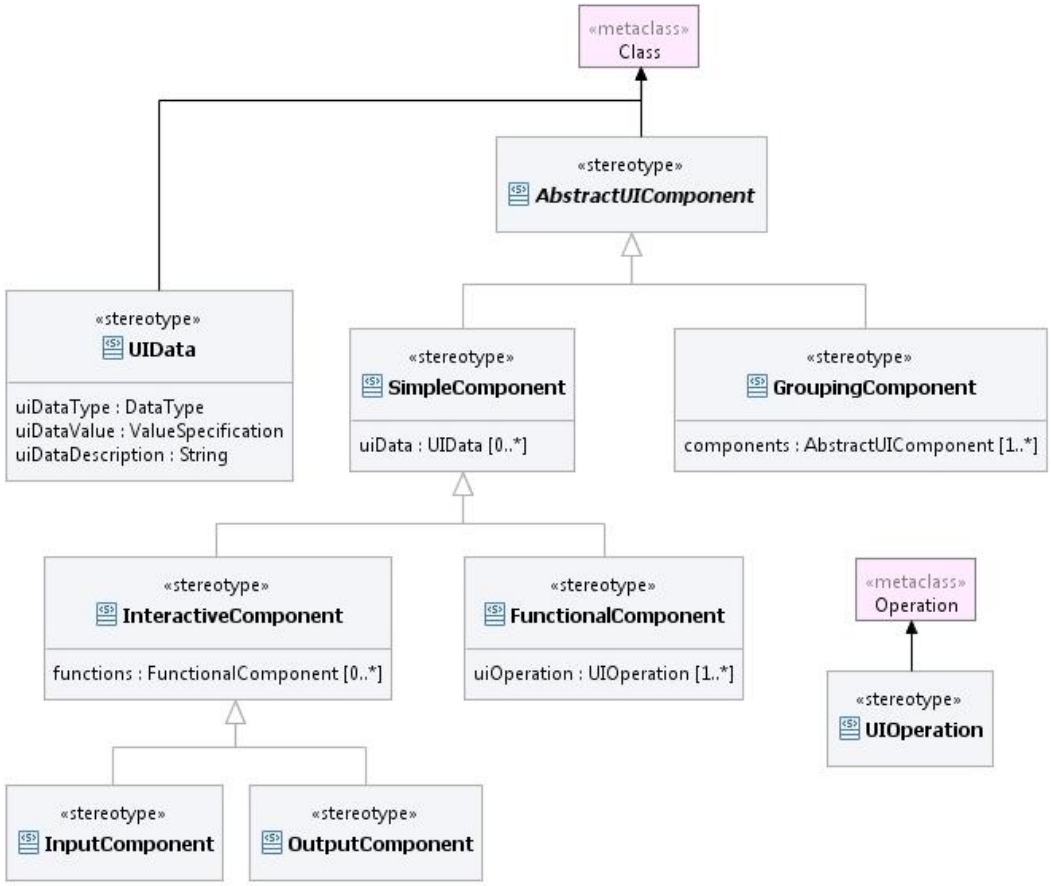
Za opisivanje relacija između zadataka uvedena su sledeća proširenja asocijacija:

- *independent_concurrency* – dva zadatka se izvršavaju proizvoljnim redosledom bez specifičnih ograničenja;
- *synchronized_concurrency* – dva zadatka se izvršavaju konkurentno, ali moraju biti sinhronizovani kako bi razmenjivali podatke;
- *order_independence* – dva zadatka se moraju izvršiti, ali kada jedan započne sa izvršavanjem mora se kompletirati pre nego što počne izvršavanje drugog zadatka;
- *activate_deactivate* – prvi zadatak se aktivira/deaktivira sa početkom izvršavanjem drugog zadatka (atribut *activation* bliže određuje tip relacije);
- *suspend_resume* – jedan zadatak može prekinuti drugi tako da po njegovom završetku prekinuti zadatak nastavlja izvršavanje restauracijom zapamćenog stanja (atribut *suspend* bliže određuje tip relacije);
- *enable_disable* – jedan zadatak omogućava/onemogućava izvršavanje drugog po svom završetku (atribut *enable* bliže određuje tip relacije).

5.2.2. Model korisničkog interfejsa nezavisan od načina komunikacije

Osnovna prednost korišćenja apstraktnih definicija korisničkog interfejsa se može posmatrati sa dva stanovišta. Sa stanovišta računara ova prednost se ogleda u apstrahovanju raznovrsnosti uređaja i tehnologija za koje se interfejs dizajnira. Na ovaj način dizajneri mogu opisati korisnički interfejs apstraktnim terminima pri čemu nisu opterećeni detaljima vezanim za specifični uređaj ili programski jezik. Ovo nas dovodi do prednosti sa stanovišta čoveka koja se ogleda u tome što se dizajneri mogu skoncentrisati na semantiku interakcije ne obazirući se na specifičnosti i detalje određenih načina komunikacije.

Na slici 5.16 je prikazan osnovni skup UML proširenja modela korisničkog interfejsa nezavisnog od načina komunikacije (apstraktni korisnički interfejs).

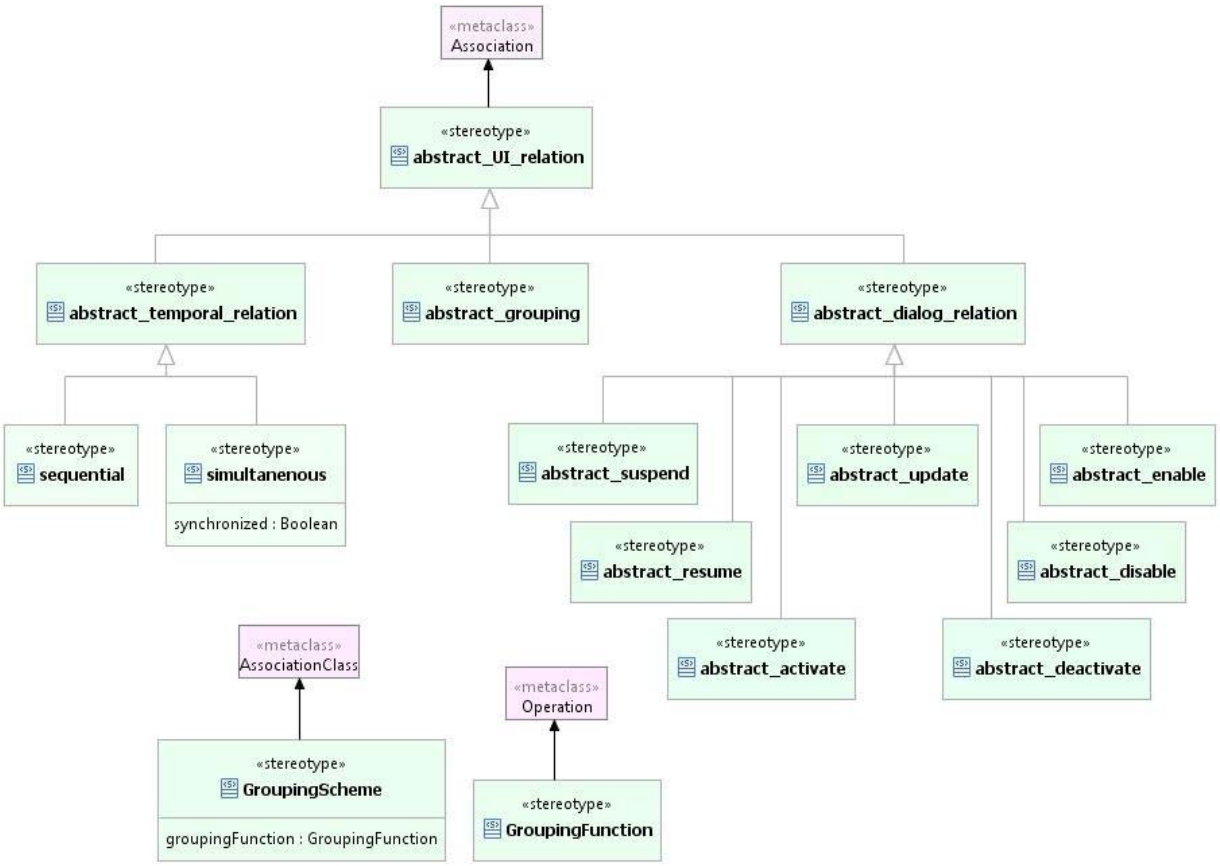


Slika 5.16. *Osnovni skup proširenja modela apstraktnog korisničkog interfejsa.*

Sa stanovišta složenosti identifikovali smo dva osnovna tipa komponenti kao jednostavnu komponentu (*SimpleComponent*) i složenu komponentu (*ComplexComponent*). Složena komponenta integriše veći broj jednostavnih ili složenih komponenti i modeluje mehanizam grupisanja kontrola korisničkog interfejsa. Osnovnoj komponenti su pridruženi podaci korisničkog interfejsa (*UIData*) koji su opisani tipom i skupom vrednosti. U zavisnosti od toga da li osnovna komponenta omogućava komunikaciju sa korisnikom ili realizuje funkcionalnosti korisničkog interfejsa, ona može biti interaktivna (*InteractiveComponent*) ili funkcionalna (*FunctionalComponent*). Interaktivna komponenta može biti

ulazna (*InputComponent*) ili izlazna (*OutputComponent*). Ovakva klasifikacija omogućava modelovanje standardnih ulaznih i izlaznih kontrola u korisničkim interfejsima. Funkcionalnosti kontrola korisničkog interfejsa su modelovane kao stereotip *FunctionalComponent* i opisane su skupom operacija korisničkog interfejsa (*UIOperation*). Funkcionalna komponenta omogućava modelovanje različitih načina realizacije ponašanja korisničkog interfejsa kao što su na primer obrađivači događaja ili delegiranje metoda. Kao takva ona nije direktno na raspolaganju korisniku, već je vezana za konkretnu interaktivnu komponentu.

Pri opisivanju relacija između apstraktnih komponenti korisničkog interfejsa oslonili smo se na tipove relacija između zadataka, kao i na apstrahovanje relacija koje se mogu naći u korisničkim interfejsima. Osnovni skup proširenja za opisivanje relacija između komponenti korisničkog interfejsa prikazan je na slici 5.17.



Slika 5.17. *Osnovni skup proširenja za modelovanje relacija u modelu apstraktnog korisničkog interfejsa.*

Za opisivanje relacija između komponenti korisničkog interfejsa uvedeni su osnovni stereotipovi asocijacija: apstraktna relacija grupisanja (*abstract_grouping*), apstraktna vremenska relacija (*abstract_temporal_relation*) i apstraktna relacija dijaloga (*abstract_dialog_relation*).

Apstraktna relacija dijaloga opisuje promene stanja apstraktnih komponenti korisničkog interfejsa koje u njoj učestvuju i ima sledeće varijante:

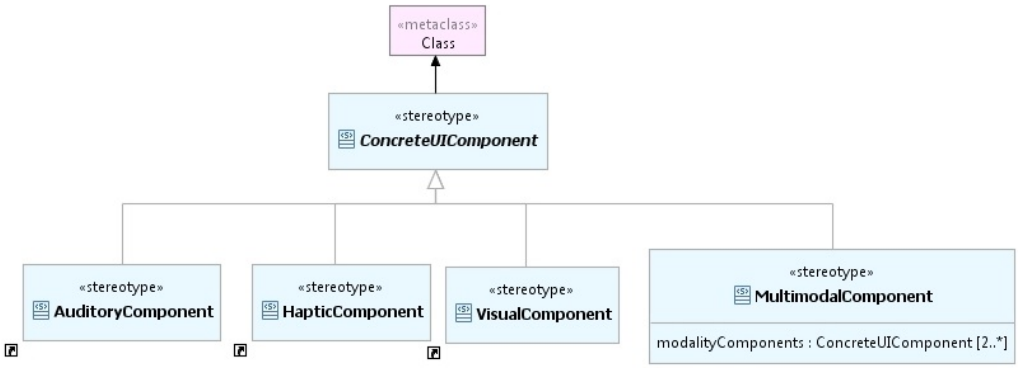
- *abstract_suspend* – modeluje relaciju u kojoj jedna komponenta suspenduje drugu u kontekstu izvršavanja funkcija. Za svaku *suspend* postojati reverzna *resume* relacija kako bi model bio koherentan;
- *abstract_resume* – modeluje relaciju u kojoj jedna komponenta omogućava drugoj da nastavi sa funkcionisanjem na osnovu zapamćenog stanja;
- *abstract_disable* – modeluje relaciju u kojoj jedna komponentu onemogućava funkcionisanje druge komponente;
- *abstract_enable* - modeluje relaciju u kojoj jedna komponentu omogućava funkcionisanje druge komponente;
- *abstract_update* – modeluje relaciju u kojoj jedna komponenta ažurira podatke druge komponente;
- *abstract_activate* – modeluje relaciju u kojoj jedna komponenta aktivira drugu;
- *abstract_deactivate* – modeluje relaciju u kojoj jedna komponenta deaktivira drugu.

Prostorne i vremenske relacije opisuju fizička ograničenja između apstraktnih komponenti u prostoru i vremenu. Kako je model nezavisan od načina komunikacije, ne možemo pretpostaviti da li će se preslikati u grafički, vokalni ili višenačinski korisnički interfejs. Ova činjenicu je potrebno uzeti u obzir pri opisivanju prostornih i vremenskih relacija. Imajući ovo u vidu uveli smo relacije apstraktnog grupisanja (*abstract_grouping*) i apstraktne vremenske relacije (*abstract_temporal_relation*). Relacije grupisanja mogu biti dodatno opisane šemom grupisanja (*GroupingScheme*) i funkcijom grupisanja (*GroupingFunction*). Na osnovu postojeće klasifikacije vremenskih relacija [Allen83] izveli smo dva osnovna tipa vremenskih relacija – sekvencijalnu (*sequential*) i simultanu (*simultanenous*). Simultana relacija ima šest varijanti (*equal*, *meets, overlaps*, *during*, *starts*, *finishes*), dok sekvencijalna ima jedan podtip (*before*). Svaka od ovih relacija ima inverznu relaciju, osim relacije *equal* koja je simetrična. Zbog preglednosti ove relacije nisu prikazane na slici 5.17.

5.2.3. Model korisničkog interfejsa zavisan od načina komunikacije

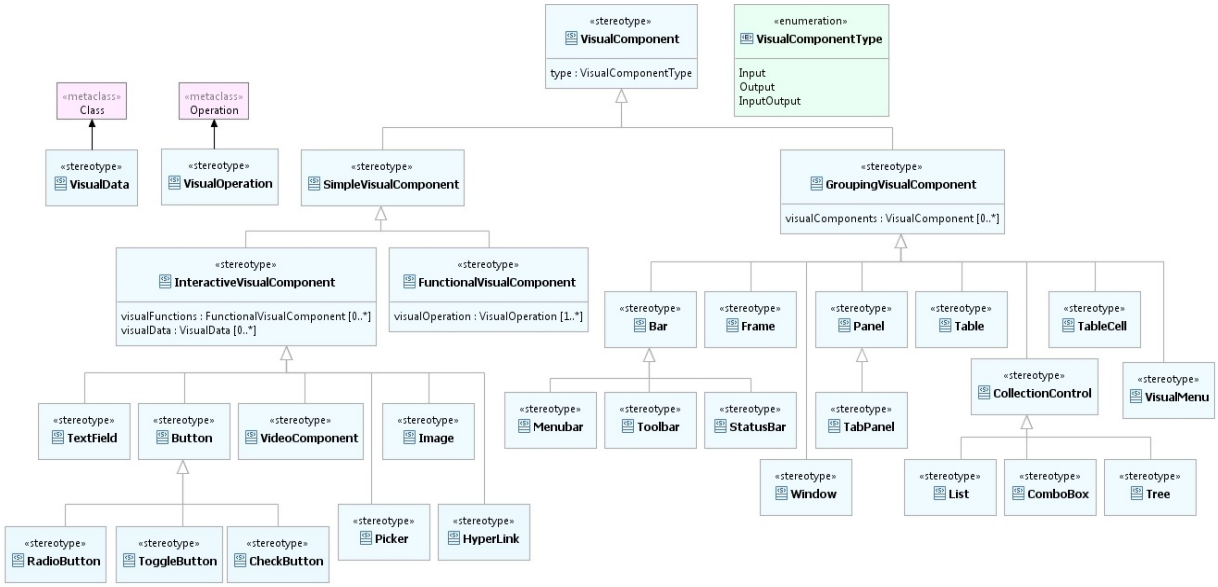
Model korisničkog interfejsa zavisan od načina komunikacije (model konkretnog korisničkog interfejsa) omogućava opise korisničkih interfejsa koji uzimaju u obzir načine komunikacije sa čovekom, ali ne i detalje tehnologije implementacije. Može se posmatrati i kao međuopis između apstraktnog modela i modela zavisnog od platforme implementacije. Ovaj model treba da omogući specifikaciju prezentacije i ponašanja korisničkog interfejsa koristeći elemente koje čovek može opaziti. Pored toga, može se uočiti postojanje preklapanja i zajedničkih elemenata između različitih standarda za kreiranje korisničkih interfejsa, tako da se nameće potreba za kreiranjem uopštenijeg pogleda na postojeće platforme. Model konkretizuje apstraktni korisnički interfejs za dati način komunikacije opisom konkretnih komponenti korisničkog interfejsa i relacija između njih. U tom

pogledu entiteti ovog modela predstavljaju apstrakcije vidžeta koji se mogu naći u postojećim tehnologijama za izradu korisničkih interfejsa (kao što su Java AWT/Swing, HTML, Flash Action Script, VoiceXML i dr.). Komponenta konkretnog korisničkog interfejsa (*ConcreteUIComponent*) predstavlja entitet (prozor, dugme, tekstualni polje, glasovni meni, glasovna ulazna komponenta, glasovna izlazna komponenta) koji čovek može opaziti i kojim može upravljati (slika 5.18). Tipovi komponenti su izvedeni na osnovu načina komunikacije. Na slici 5.19 su prikazane vizuelna (*VisualComponent*), auditorna (*AuditoryComponent*) i dodirna (*HapticComponent*) komponenta. Svaka od komponenti sama po sebi omogućava jednonačinsku komunikaciju sa korisnikom. Kako bi omogućili dizajn komponenti koje će obezbediti višenačinsku komunikaciju sa korisnikom uveden je multimodalni tip (*MultimodalComponent*) koji integriše veći broj različitih unimodalnih tipova. Modeli konkretnih korisničkih interfejsa će biti opisani na primerima vizuelnog i glasovnog korisničkog interfejsa koji su ujedno i najzastupljeniji u praksi.



Slika 5.18. *Osnovni skup proširenja modela korisničkog interfejsa zavisnogod načina komunikacije.*

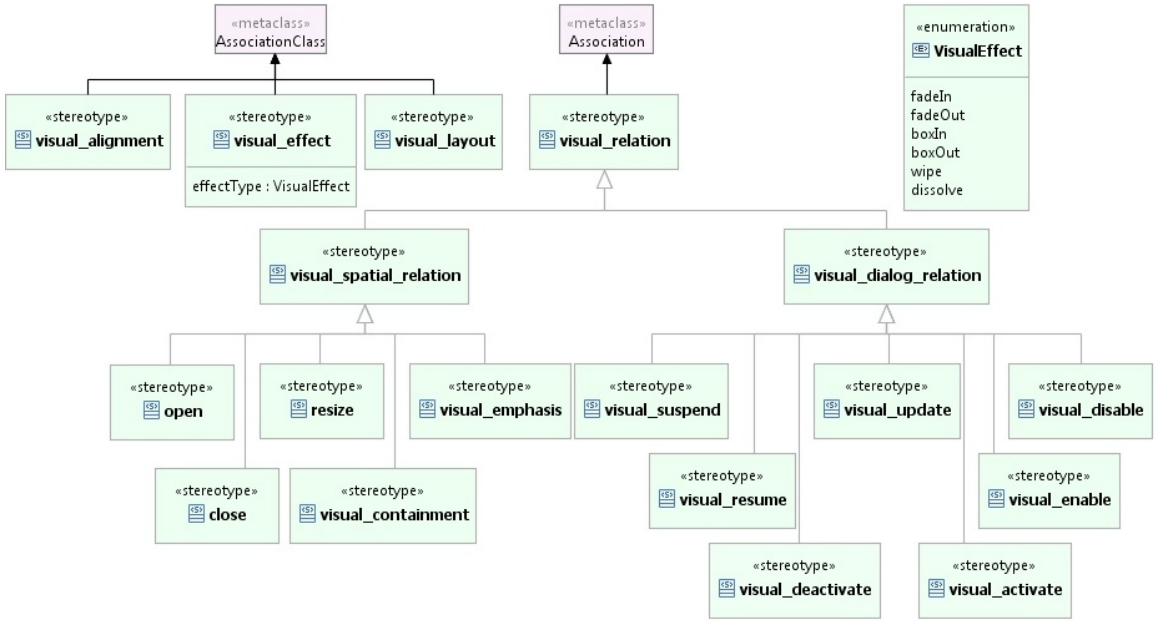
Vizuelna komponenta može biti ulazna ili izlazna u zavisnosti od konkretne namene i konteksta korišćenja. Zbog toga smo uveli nabrojivi tip vizuelne komponente (*ConcreteComponentType*) koji omogućava opisivanje vizuelne komponente kao ulazne, izlazne ili ulazno-izlazne. Na slici 5.19 je prikazan deo skupa proširenja za modelovanje vizuelnih korisničkih interfejsa. Model apstrahuje standardan skup kontrola koje se mogu naći u vizuelnim korisničkim interfejsima.



Slika 5.19. *Proširenja za modelovanje standardnih elemenata vizuelnih korisničkih interfejsa.*

Sa stanovišta složenosti, vizuelna komponenta (*VisualComponent*) može biti jednostavna (*SimpleVisualComponent*) ili komponenta grupisanja (*GroupingVisualComponent*). Komponente grupisanja predstavljaju kontejnere koji omogućavaju izvršavanje logički povezanih zadataka koje realizuju pojedinačne komponente. Ovde smo svrstali standardne kontejnere koji se mogu naći u grafičkim korisničkim interfejsima kao što su okvir (*Frame*), panel (*Panel*), vizuelni meni (*Menu*) i dr. Kontrole kao što su stablo (*Tree*), tabela (*Table*) ili homogene kolekcije (*CollectionComponent*) smo takođe uvrstili u ovu grupu kontrola jer ih možemo posmatrati kao specifične hijerarhijske strukture jednostavnijih komponenti. U skladu sa modelom apstraktnog korisničkog interfejsa, jednostavna vizuelna komponenta (*SimpleVisualComponent*) može biti interaktivna (*InteractiveVisualComponent*) ili funkcionalna (*FunctionalVisualComponent*). Interaktivna komponenta realizuje prezentacione aspekte korisničkog interfejsa i zadužena je za komunikaciju sa korisnikom. Svakoj interaktivnoj komponenti su pridruženi podaci korisničkog interfejsa (*VisualData*) koji određuju stanje komponente. U interaktivne komponente spadaju standardne kontrole kao što su tekstualna polja (*TextField*), dugmad (*Button*), slike (*Image*) i dr. Funkcionalne komponente realizuju akcije vezane za pojedinačne interaktivne komponente i na taj način određuju ponašanje korisničkog interfejsa. Zbog toga ove komponente imaju skup operacija vizuelnog korisničkog interfejsa koje su modelovane stereotipom *VisualOperation*. U izvršnoj verziji korisničkog interfejsa, ove komponente se mogu preslikati u obrađivače događaja ili komponente koje realizuju skupove servisa (metoda). Zbog preglednosti prikazan je deo proširenja i izostavljene su pridodane vrednosti pojedinačnih komponenti.

Skup proširenja za opisivanje relacija između komponenti vizuelnih korisničkih interfejsa je prikazan na slici 5.20. Relacije smo klasifikovali kao vizuelne relacije dijaloga (*visual_dialog_relation*) i prostorne vizuelne relacije (*visual_spatial_relation*). Vizuelne relacije dijaloga opisuju promene stanja vizuelnih komponenti koje u njoj učestvuju. Prostorne vizuelne relacije imaju navigacionu i prostornu semantiku, jer je vizuelna percepcija čoveka vezana za opažanje pojava u prostoru.



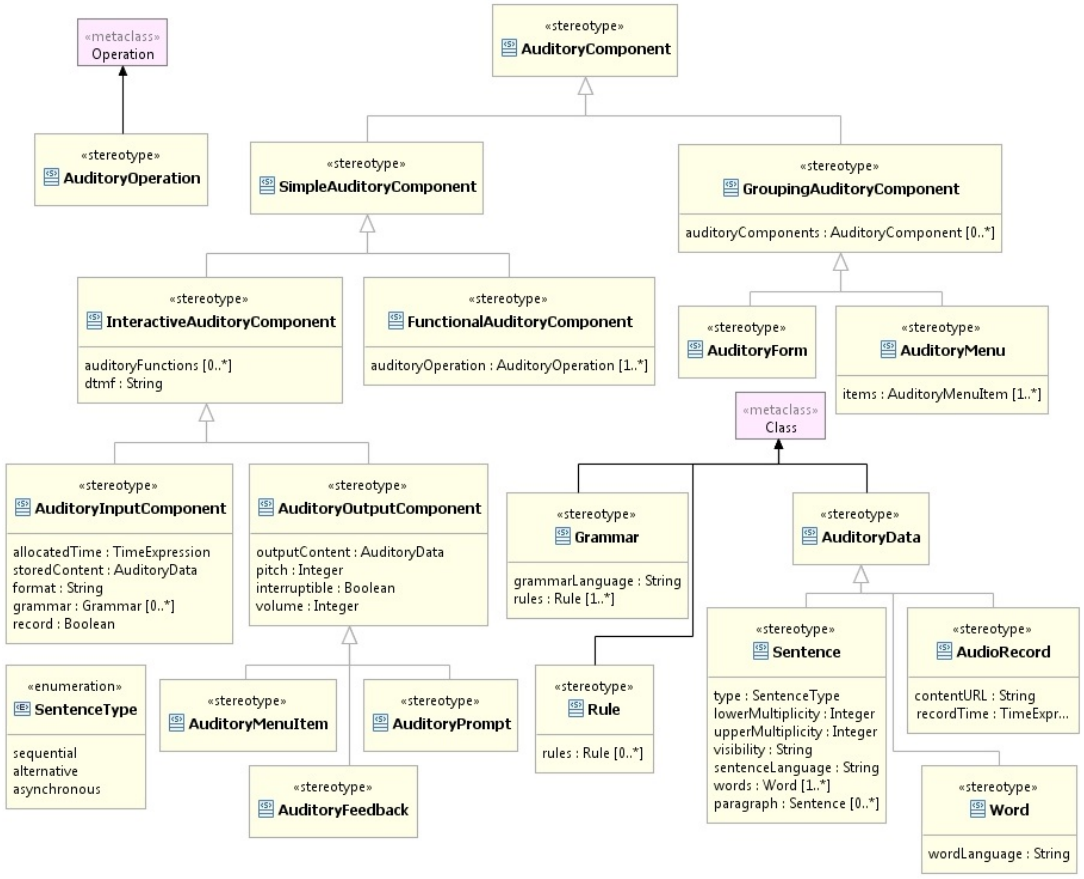
Slika 5.20. *Proširenja za modelovanje relacija između vizuelnih komponenti.*

Značenje vizuelnih relacija dijaloga je preslikano iz modela apstraktnog korisničkog interfejsa, s tim što smo uveli dva proširenja – *visual_activate* i *visual_deactivate*. Semantika ovih relacija je vezana sa pokretanje ili zaustavljanje funkcionalnosti vezanih za pojedinačne vizuelne interaktivne komponente. Prostorna vizuelna relacija ima sledeće varijante:

- *open/close* – modeluje relaciju u kojoj jedna komponenta inicira otvaranje/zatvaranje drugog kontejnera;
- *resize* – modeluje relaciju u kojoj se menjaju dimenzije komponente (promenu može direktno inicirati korisni ili druga komponenta);
- *visual_emphasis* – modeluje relaciju u kojoj se vizuelno naglašava komponenta;
- *visual_containment* – modeluje relaciju sadržavanja između kontejnerske komponente i komponenti koje ona sadrži.

Semantika uvedenih relacija može biti obogaćena proširenjima asocijacionih klasa. Na taj način relacija vizuelnog sadržavanja može biti dodatno opisana rasporedom (*visual_alignment*) i ravnanjem (*visual_alignment*) komponenti. Pored toga, svakoj od relacija može biti pridružen vizuelni efekat (*visual_effect*) u obliku različitih tipova animacija (kao što su *fade in*, *fade out*, *wipe*, *disolve* i dr.).

Na slici 5.21 prikazan je osnovni skup proširenja za modelovanje glasovnih korisničkih interfejsa.



Slika 5.21. *Platformski generička proširenja za modelovanje glasovnih korisničkih interfejsa.*

Glasovna komponenta (*AuditoryComponent*) može biti jednostavna (*SimpleAuditoryComponent*) ili komponenta grupisanja ili glasovni kontejner (*GroupingAuditoryComponent*). Glasovni kontejner smo dalje klasifikovali kao glasovnu formu (*AuditoryForm*) i glasovni meni (*AuditoryMenu*). Glasovna forma predstavlja opšti oblik kontejnera koji omogućava dijalog između sistema i korisnika, tj. razmenu glasovnih informacija u komunikaciji čoveka i računara. Glasovni meni predstavlja specijalizovani kontejner koji korisniku omogućava izbor različitih opcija. Razlog za uvođenje ovog tipa je što se tradicionalni glasovni dijalozi često sastoje od skupa glasovnih opcija koje stoje na raspolaganju korisniku. Jednostavna glasovna komponenta (*SimpleAuditoryComponent*) može biti interaktivna (*InteractiveAuditoryComponent*) ili funkcionalna (*FunctionalAuditoryComponent*). Interaktivna glasovna komponenta je zadužena za komunikaciju sa korisnikom, dok funkcionalna komponenta definiše ponašanje interaktivne komponente preko skupa glasovnih operacija (*AuditoryOperation*). Interaktivna komponenta kao metaatribut ima *dtmf* prečicu na tastaturi. DTMF (*Dual Tone Multi-Frequency*) predstavlja sistem koji se koristi u telefoniji i koji se sastoji u dodeljivanju specifičnih frekvencija dugmadima tastature kako bi ih glasovni mehanizam precizno identifikovao. Upotreba ovog sistema se može uopštiti u glasovnim korisničkim interfejsima.

Kod glasovnih korisničkih interfejsa postoji razlika između ulaznih i izlaznih komponenti. U skladu sa tim su izvedena dva osnovna tipa iz interaktivne glasovne komponente.

Glasovna ulazna komponenta (*InputAuditoryComponent*) je zadužena za prijem glasovnih informacija iz okruženja prepoznavanjem govora ili snimanjem audio sadržaja. Pridodana vrednost *allocatedTime* predstavlja vremenski okvir u kojem korisnik može kreirati glasovne iskaze komponenti. Atribut *storedContent* sadrži podatke ulazne glasovne komponente (proširenje *AuditoryData*). Format ulaznih podataka je definisan atributom *format*. U konkretnom slučaju, format ulaznih podataka može biti unapred određen (na primer logički tip, numerički podaci, podaci alfabeta, datum, vreme i dr.). Atribut *record* govori da li komponenta skladišti glasovni ulaz. Kao što je prethodno rečeno, ulazna glasovna komponenta može vršiti preoznavanje govora ili obično snimaje audio sadržaja. U slučaju prepoznavanja govora, mora postojati mehanizam koji će ulaznim podacima dati odgovarajuću semantiku kako bi korisnički interfejs znao koje akcije treba da preduzme. Zbog toga su uvedena proširenja koja omogućavaju obradu glasovnih informacija i njihovu interpretaciju simbolima konkretnog jezika. Gramatika (*Grammar*) sadrži skup pravila (*Rule*) koja se primenjuju u obradi ulaznih glasovnih iskaza (*Sentence*) kako bi ih sistem na odgovarajući način interpretirao. Atribut *grammarLanguage* specificira jezik kojim iskazi moraju biti izgovoreni kako bi ih sistem prepoznao, tj. čija će gramatika biti korišćena. Pravila gramatike mogu biti jednostavna ili složena tako da sadrže jednostavnija pravila. Zbog toga je atribut *rules* pridodan stereotipu *Rule*. Gramatika obrađuje glasovni iskaz koji predstavlja ulaz u vidu reči, fraze, rečenice ili paragrafa. Iskaz može biti jednostavan (reč, fraza, rečenica) ili složen (paragraf). Zbog toga je sterotipu *Sentence* pridodan atribut *paragraph* koji omogućava modelovanje iskaza različite složenosti. Atribut *type* određuje način na koji ulaz korisnika mora biti iskazan kako bi ga sistem prepoznao i dobija tri vrednosti (*SentenceType*):

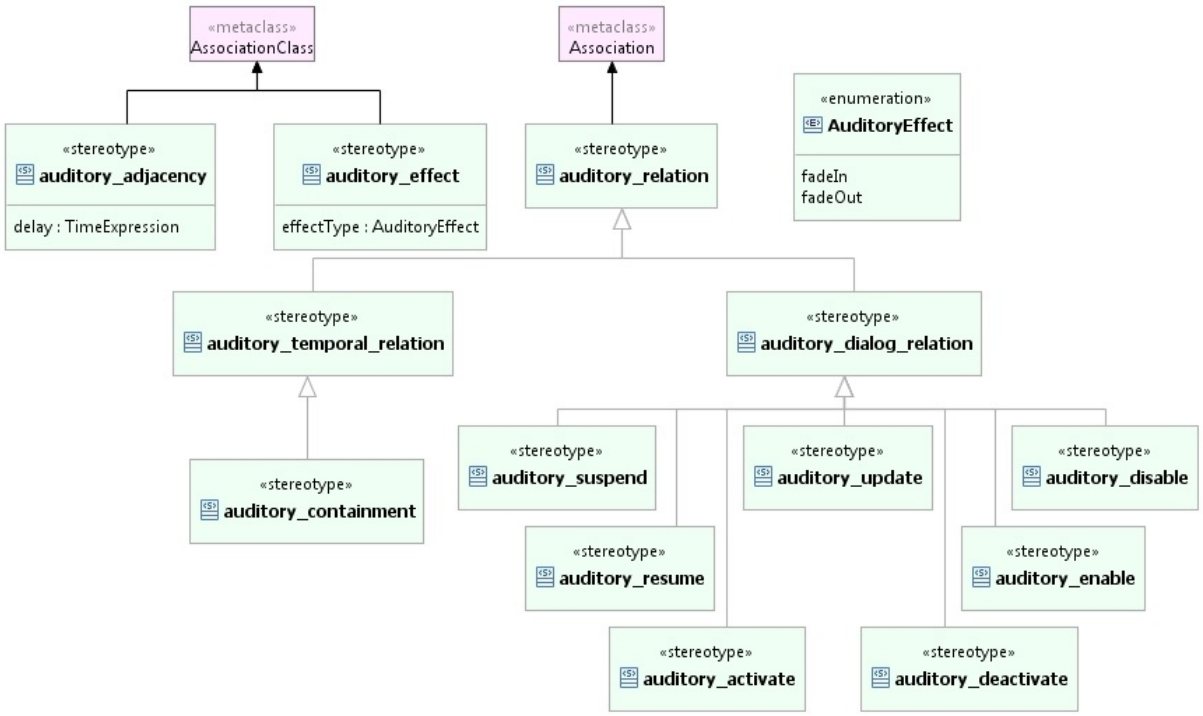
- *sequential* – ulaz predstavlja sekvencu iskaza koji se obrađuju redosledom kojim su izgovoreni;
- *alternative* – ulaz predstavlja opcione iskaze;

- *asynchronous* – ulaz predstavlja skup iskaza sa proizvoljnim redosledom unosa.

Atribut vidljivosti (*visibility*) definiše opseg korišćenja iskaza, tj, da li se može koristiti u jednoj (*private*) ili ga može referencirati veći broj različitih gramatika (*public*). Iskaz se sastoji od reči (*Word*) kao elementarnih entiteta iskaza. Atribut *wordLanguage* označava jezik kojim reč mora biti izgovorena kako bi je sistem prepoznao. Uvođenje atributa jezika na nivou gramatike, iskaza i reči omogućava dizajn glasovnih korisničkih interfejsa koji podržavaju komunikaciju na većem broju jezika. Podrazumevana vrednost ovih atributa je određena vrednošću atributa okružujućeg elementa. Pridodanim atributima *lowerMultplicity* i *upperMultiplicity* stereotipa *Sentence* je određen broj ili opseg broja ponavljanja reči u okviru okružujućeg iskaza.

Glasovna izlazna komponenta (*AuditoryOutputComponent*) kreira glasovni izlaz ka korisniku ili okruženju. Podaci komponente su predstavljeni atributom *outputContent*. Jačina i visina zvuka koji komponenta proizvodi su predstavljeni atributima *volume* i *pitch*, respektivno. Atribut *interruptible* govori da li glasovni izlaz može biti ulazom korisnika. Iz izlazne glasovne komponente izvedena su tri tipa - stavka glasovnog menija (*AuditoryMenuItem*), izlazna komponenta koja daje povratnu informaciju u vezi sa prethodno izvršenom akcijom ili ulazom (*AuditoryFeedback*), i komponenta koja predstavlja jednostavan upit korisniku samostalan ili u sklopu složenijeg dijaloga (*AuditoryPrompt*). Dve poslednje komponente su apstrahovane iz postojećih tehnologija kao što je *VoiceXML*.

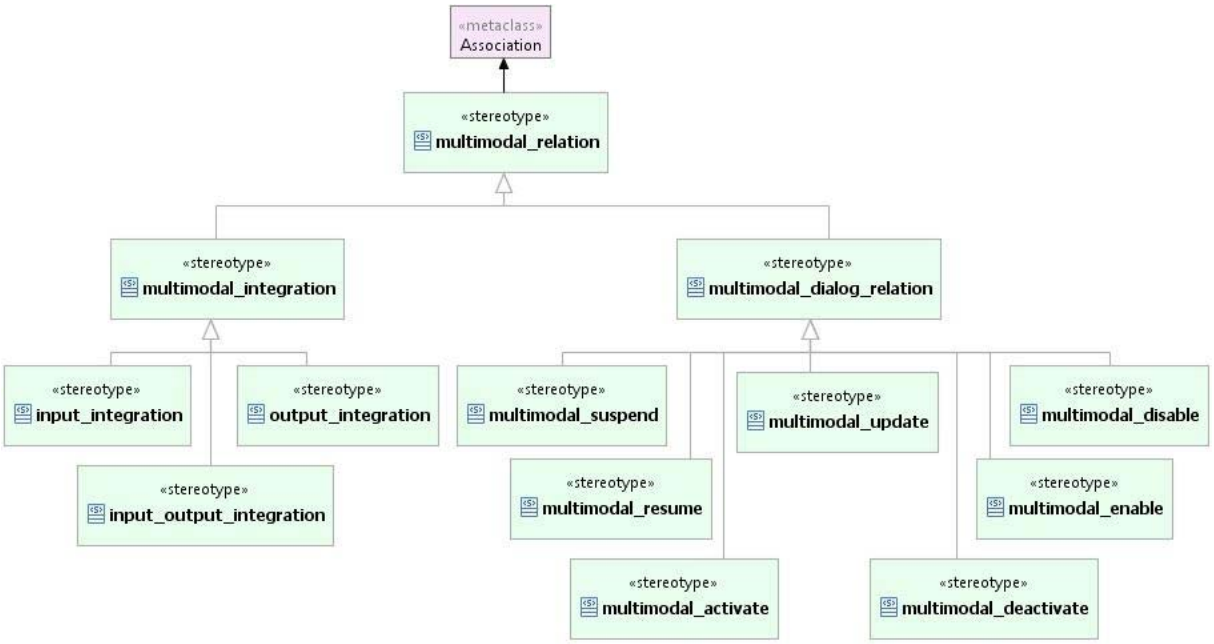
Na slici 5.22 priikazan je skup proširenja za modelovanje relacija između komponenti u glasovnim korisničkim interfejsima.



Slika 5.22. *Proširenja za modelovanje relacija između glasovnih komponenti.*

Relacija između glasovnih komponenti korisničkog interfejsa (*auditory_relation*) može biti glasovna relacija dijaloga (*auditory_dialog_relation*) i temporalna glasovna relacija (*auditory_temporal_relation*). Značenje glasovnih relacija dijaloga je preslikano iz modela apstraktnog korisničkog interfejsa, s tim što smo uveli dva proširenja – *auditory_activate* i *auditory_deactivate*. Semantika ovih relacija je vezana sa pokretanje ili zaustavljanje funkcionalnosti vezanih za pojedinačne glasovne interaktivne komponente. Sa stanovišta komunikacije sa korisnikom, relacija glasovnog sadržavanja (*auditory_containment*) ima vremensku semantiku i izvedena je iz glasovne vremenske relacije. Semantika glasovnih relacija može biti dodatno proširena stereotipima *auditory_adjacency* i *auditory_effect*. Stereotip *auditory_adjacency* omogućave definisanje vremenskih ograničenja u relacijama između interaktivnih glasovnih komponenti. Ovo ograničenje je specificirano atributom *delay* koji određuje vremenski razmak u izvršavanju dve glasovne komponente. Stereotip *auditory_effect* omogućava dodavanje zvučnih efekata relacijama dijaloga.

Na slici 5.23 prikazan je generički skup proširenja za modelovanje relacija između komponenti različitih načina komunikacije u višenačinskim korisničkim interfejsima.



Slika 5.23. *Generička proširenja za modelovanje relacija u višenačinskim korisničkim interfejsima.*

Višenačinska relacija (*multimodal_relation*) može biti višenačinska relacija dijaloga (*multimodal_dialog_relation*) i višenačinska relacija integracije (*multimodal_integration*). Relacije dijaloga se uspostavljaju između interaktivnih komponenti koje koriste različite načine komunikacije sa korisnikom. Ove relacije omogućavaju promenu stanja i sinhronizaciju podataka između specifičnih interaktivnih komponenti. Na primer, ako se vizuelni fokus prebaci sa tekstualnog polja na meni, mora se aktivirati glasovna reprezentacija istog menija. Relacija višenačinske integracije komponenti korisničkog interfejsa je važna sa aspekta komunikacije sa korisnikom. Ona može ulazna (*input_integration*), izlazna (*output_integration*) ili ulazno-izlazna (*input_output_integration*). U skladu sa preporukama [Larson06] ulazna integracija može imati tri varijante:

- sekvencijalni ulaz (*sequential_input*) - jednonačinski ulaz pri čemu se načini komunikacije mogu menjati u toku vremena;

- sinhroni ulaz (*sequential_output*) – višenačinski ulaz pri čemu se modaliteti interpretiraju redosledom kojim se koriste bez spajanja pre intrerpretacije;

- kompozitni ulaz (*composite_input*) – višenačinski ulaz pri čemu se modaliteti prethodno integrišu, a potom interpretiraju kao jedinstven ulaz.

5.2.4. Model korisničkog interfejsa prilagođen tehnologiji implementacije

Modeli korisničkog interfejsa prilagođeni tehnologiji implementacije (modeli konačnog korisničkog interfejsa) predstavljaju apstrakcije konkretnih tehnologija za implementaciju korisničkih interfejsa. Možemo ih posmatrati kao adaptere između modela korisničkog interfejsa zavisnog od načina komunikacije i samih implementacija.

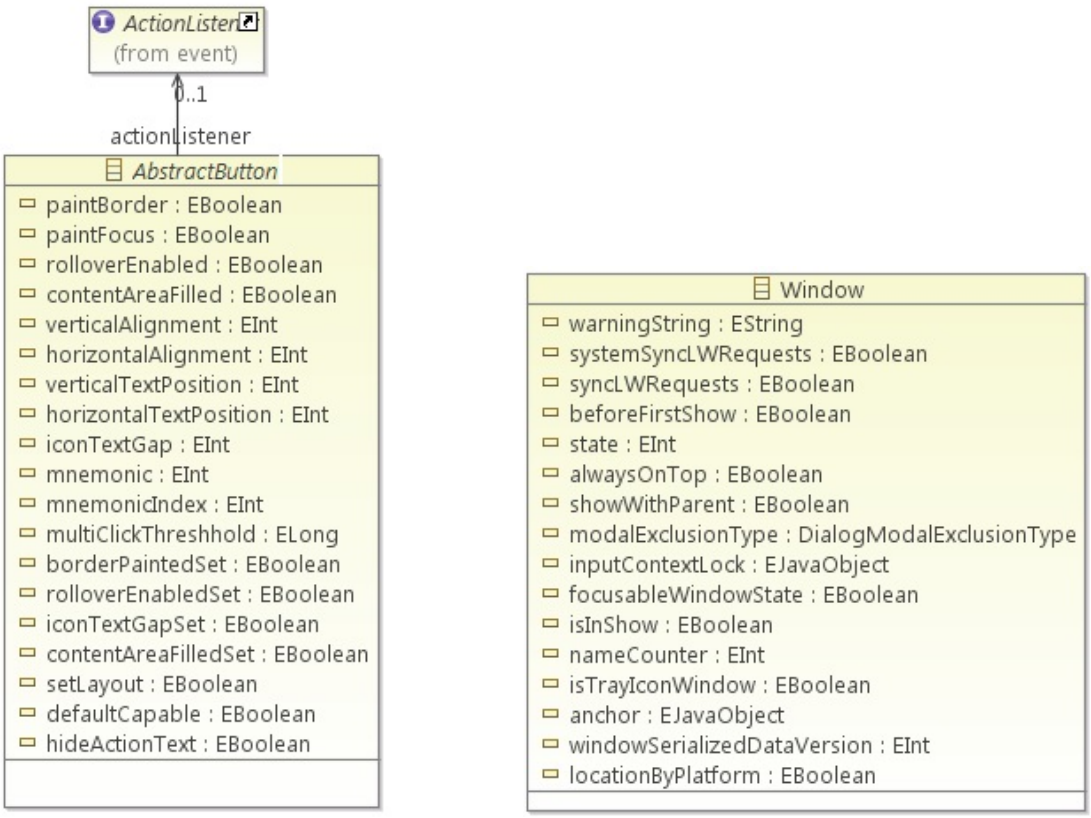
U dosadašnjim pristupima pod modelom konačnog korisničkog interfejsa smatrao se model nivoa apstrakcije instanci (MDA nivo M0). Drugim rečima, model konačnog korisničkog interfejsa predstavlja implementaciju korisničkog interfejsa u nekom od jezika za opis korisničnih interfejsa koja se može prevesti i izvršiti. U našem pristupu, model konačnog korisničkog interfejsa je UML model čiji je metamodel prilagođen specifičnoj tehnologiji za kodiranje korisničkog interfejsa. Ovaj model služi kao osnova za generisanje programskog kôda.

Sa jedne strane, ovakav pristup zahteva postupak kreiranja metaopisa postojećih tehnologija. Deklarativni jezici za opis korisničkih interfejsa (kao što su UIML, XAML, MXML i dr.) imaju dobro-definisanu gramatiku i hijerarhijsku specifikaciju, tj. sami po sebi imaju definisan metamodel. Imperativni jezici i tehnologije zasnovane na njima (kao što su na primer Java AWT/Swing) iziskuju vreme i napor u kreiranju odgovarajućih metamodela. Pored toga, uvođenje ovih modela produžava modelski zasnovan razvoj za još jednu fazu. Umesto generisanja programskog kôda direktno iz modela konkretnog korisničkog interfejsa, ovde se konkretni korisnički interfejs preslikava u konačni interfejsa (*model-to-model* transformacija) iz kojeg se generiše programski kôd (*model-to-text transformacija*).

Sa druge strane, ovakav pristup pruža dodatnu fleksibilnost u razvoju. Pre svega, jednom kreirani metamodeli tehnologija za izradu korisničkih interfejsa se mogu dalje koristiti u različitim domenima. Uzimajući u obzir raznovrsnost i kompleksnost tehnologija, semantika interakcije će biti bolje očuvana razdvajanjem metamodela zavisnih od načina komunikacije i modela zavisnih od platforme implementacije. Za dati način komunikacije (na primer vizuelni) postoji veliki broj tehnologija za implementaciju korisničkog interfejsa. Pored toga, ovakav pristup omogućava integraciju različitih tehnologija (na primer JavaSpeech i Java Swing) u slučaju razvoja višenačinskih interfejsa. Sa aspekta adaptacije, ovakav pristup obezbeđuje konzistentno ponašanje korisničkog interfejsa kako sa stanovišta tehnologije, tako i sa stanovišta konteksta interakcije.

Kreiranje metamodela iz specifikacija programskog jezika je podržano od strane različitih alata i tehnologija. EMF omogućava kreiranje metamodela (*ECORE* modeli) uvlačenjem (*eng. import*) drugih formata kao što su XMI, XML šema, *Rational Rose* dijagram klasa ili Java izvorni kôd sa anotacijama. Ovaj postupak još nije u potpunosti automatizovan u samom alatu, pa je potrebno vršiti određene korekcije. Kada govorimo o deklarativnim jezicima oni se relativno jednostavno mogu opisati XML šemama. U pogledu imperativnih jezika situacija je nešto složenija. Na primer, da bi se iskoristio izvorni Java kôd potrebno je svaki identifikator, metodu, klasu ili interfejs anotirati sa @model. Ovo se može postići kreirajem alata koji će izvršiti odgovarajuće modifikacije na nivou kôda i pripremiti ga za uvlačenje od strane EMF.

Korišćenjem opisanih tehnika, kreirani su metamodeli Java Swing i Java AWT tehnologija za izradu korisničkog interfejsa (slika 5.24). Metamodeli su kreirani na osnovu anotiranog Java kôda, pri čemu su korišćene postojeće tehnike i alati [Wolff10]. Na slici su prikazani pojedinačni elementi modela. Kao primer Java Swing komponente prikazani su komponenta apstraktno dugme (apstraktna klasa *AbstractButton*) i njoj pridruženi osluškivač događaja (*interfejs ActionListener*). Kontejnerska komponenta prozor (*Window*) je izdvojena u kontekstu Java AWT tehnologije.

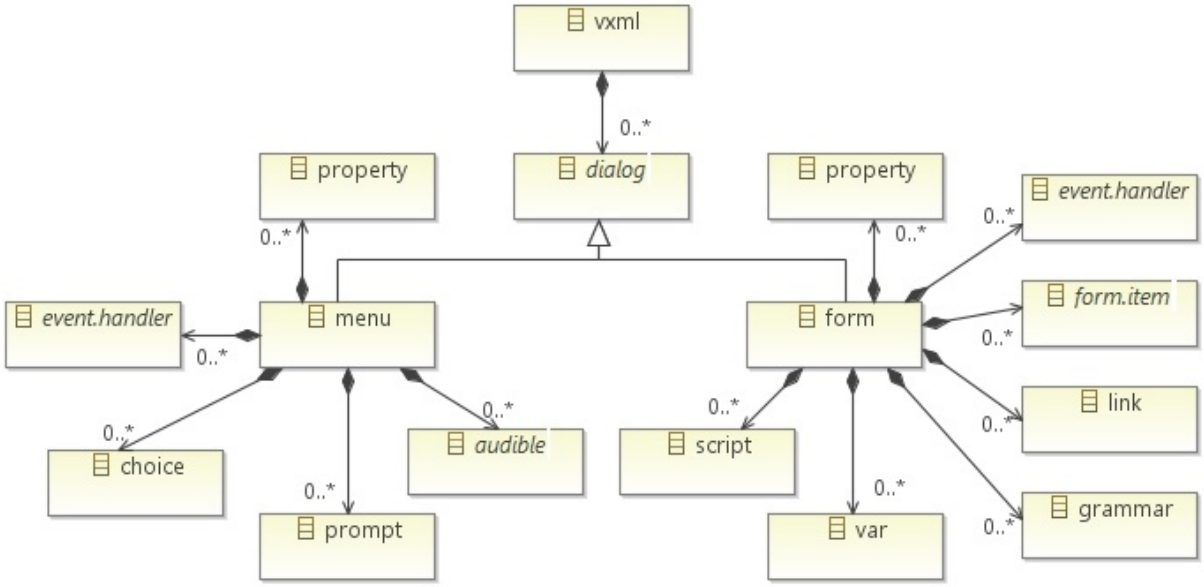


(a) (b)

Slika 5.24. *Kontrole (a) JavaSwing i (b) Java AWT tehnologije kao koncepti metamodela konačnog korisničkog interfejsa.*

Na slici 5.25 prikazan je deo metamodela VoiceXML govorne platforme. Metamodel je kreiran uvlačenjem XML šeme u EMF okruženje i demonstrira upotrebu deklarativnog jezika. Šema je dobijena XSL transformacijom DTD šeme koja opisuje strukturu i gramatiku VoiceXML jezika. Na

slici je prikazana definicija apstraktnog dijaloga (apstraktna klasa *dialog*). Entitet *vrml* modeluje koreni element VoiceXML dokumenta. Entitet *dialog* je apstraktan i kao takav definisan u DTD šemi jezika, dok se u okviru VoiceXML dokumenata javlja u obliku menija (*menu*) i forme (*form*). Svaki od ovih elemenata može sadržati odgovarajuće tipove. Tipovi *audible, event.handler* i *form.item* predstavljaju apstraktne klase. Iz svakog od njih su izvedeni konkretni tipovi koji mogu biti instancirani. Zbog preglednosti, izvedeni tipovi nisu prikazani, ali se oni mogu naći u specifikaciji samog jezika. Primera radi, entitet *audible* označava audio materijal koji se prezentuje korisniku i može biti audio klip (*audio*), parsirani skup karaktera (*PCDATA*) ili sadržaj dobijen transformacijom teksta u govor *Text-To-Speech* (*var*).



Slika 5.25. *Deo metamodela govorne platforme VoiceXML dobijen uvlačenjem XML šeme koja opisuje rečnik i strukturu jezika.*

5.3. *Rezime poglavlja*

U ovom poglavlju smo definisali proširenja jezika za modelovanje softverskih sistema. Zbog standardizacije, široke prihvaćenosti i podrške u vidu dostupnih alata opredelili smo se za proširenje jezika UML. U uvodnom odeljku smo dali pregled načina proširenja jezika UML i ukratko opisali mehanizam za koji smo se opredelili, tj. UML profile. Prvi deo poglavlja je posvećen profilima za modelovanje konteksno-osetljive interakcije čoveka i računara. Drugim rečima, opisani su profil okruženja, profil računarske platforme i profil čoveka. U drugom delu je dat pregled profila za razvoj korisničkih interfejsa, tj. profil zadataka, profil korisničkog interfejsa nezavisnog od načina komunikacije, profil korisničkog interfejsa zavisnog od načina komunikacija i profili korisničkih interfejsa prilagođenih tehnologiji implementacije, respektivno.

Definisana proširenja će biti korišćena u razvoju kontekstno-osetljivih korisničkih interfejsa. U narednom poglavlju opisujemo mogućnosti korišćenja predloženih proširenja u procesu razvoja softvera. U poglavlju 7, govorićemo o mogućnostima automatizacije procesa razvoja korisničkih interfejsa, tj. transformacijama modela korisničkih interfejsa korišćenjem predloženih proširenja.

6. Proces razvoja kontekstno-osetljivih korisničkih interfejsa

U prethodnom poglavlju smo proširili jezik UML konceptima vezanim za kontekstno-osetljivu interakciju čoveka i računara, kao i primitivama specifičnim za razvoj korisničkih interfejsa. U ovom poglavlju ćemo opisati kako se predložena proširenja mogu primeniti u standardnom procesu razvoja softvera (*Unified* procesa). Drugim rečima, govorićemo o proširenju standardnog procesa razvoja softverskih sistema. Većina aktuelnih metodologija razvoja je usredsređena na predstavljanje procedura, informacija i softverskih struktura. Aspekti vezani za dizajn interakcije sa čovekom nisu integrisani u sam proces razvoja, već se spominju naknadno i rešavaju od slučaja do slučaja korišćenjem nestandardizovanih jezika, alata i tehnika koji su vrlo često isuviše speciifični za određeni domen [Constantine09]. Pojednostavljeno govoreći, proces razvoja softvera opisuje kako se korisnički zahtevi transformišu u konkretan softverski sistem. Ovaj proces se odvija kroz više faza, kreiranjem više modela na različitim nivoima apstrakcije. U tom pogledu, proces razvoja softvera obezbeđuje sistematičan pristup izradi programskog sistema sa jasno definisanim fazama, kao i aktivnostima i modelima koji se u njima koriste. Zbog široke prihvaćenosti, standardizacije i dostupnosti razvojnih alata, opredelili smo se za istraživanje mogućnosti proširivanja metodologije za razvoj softverskih sistema poznate pod nazivom *Unified* proces. *Unified* proces je generički razvojni proces koji može da se specijalizuje za različite klase softverskih sistema, oblasti primene, vrste organizacija, nivoe kompetentnosti i veličine projekata. *Unified* proces je tesno povezan sa UML jezikom koji čini njegov sastavni deo. Ključni osobine *Unifed* procesa su: upravljan slučajevima korišćenja, orijentisan ka arhitekturi, iterativan i inkrementalan [Jacobson99].

U ovom poglavlju opisaćemo neke od mogućih pristupa proširenju *Unified* procesa sa konceptima vezanim za kontekstno-osetljive korisničke interfejse. Proširenje se ogleda u integraciji odgovarajućih UML profila sa modelima koji se realizuju u pojedinim fazama procesa razvoja.

Najpe dajemo kratak osvrt na *Unified* razvojni proces po fazama. Uz kratak opis svake faze, ističemo nedostatke sa stanovišta interakcije čoveka i računara i dajemo predlog proširenja. Potom opisujemo osnovnu ideju proširenja razvojnog procesa. Zatim opisujemo upotrebu UML profila opisanih u prethodnom poglavlju i njihovu vezu sa pojedinim fazama razvoja softvera.

6.1. *Osvrt na postojeći proces razvoja softvera*

Razvoj nekog softverskog sistema odvija se kroz više faza, kreiranjem većeg broja modela na različitim nivoima apstrakcije. Aktivnosti kroz koje se obično prolazi u realizaciji softverskih sistema su [Jacobson99]: specifikacija zahteva, analiza, dizajn, implementacija i testiranje.

U daljem tekstu ćemo ukratko opisati svaku od faza u procesu razvoja softvera, kao i potrebna proširenja modela koji se koriste u ovim fazama sa stanovišta kontektsno-osetljive interakcije čoveka i računara, tj, razvoja korisničkih interfejsa. Iako će faze biti opisane u naznačenom redosledu, treba

naglasiti da se one u određenoj meri preklapaju i da se razvoj sistema u celini odvija iterativno i inkrementalno.

6.1.1. Specifikacija korisničkih zahteva

Specifikacija zahteva je proces pronalaženja odgovora na pitanje *šta sistem treba da radi*. Pri specifikaciji, između ostalog, identifikuju se [Jacobson99]: *okruženje sistema*, odnosno kontekst u kome sistem treba da radi; *funkcionalni zahtevi* na osnovu kojih se dolazi do preciznog razumevanja zahtevane funkcionalnosti softverskog sistema; *nefunkcionalni zahtevi* koji identifikuju ograničenja implementacionog i sistemskog okruženja, kao što su performanse, platformska zavisnost i održavanje. Pri specifikaciji zahteva kreira se model slučajeva korišćenja čiji su osnovni elementi učesnici i slučajevi korišćenja. *Unified* proces koristi UML dijagrame slučajeva korišćenja (*eng. use case diagrams*) koji omogućuju jednostavnu definiciju korisnika (*eng. actors*), slučajeva korišćenja i subjekata (*eng. subject*). Subjekat je posmatrani sistem na koji se odnose slučajevi korišćenja. Dodatno je moguće opisati ponašanje pojedinih slučajeva korišćenja pomoću dijagrama aktivnosti i objektnih dijagrama.

Specifikacija zahteva treba da omogući detaljnije opise faktora od interesa za kontekstno-osetljivu interakciju između čoveka i računara. Ovi opisi predstavljaju osnovu kasnijih faza izrade korisničkog interfejsa. Potrebna semantička proširenja specifikacije zahteva pre svega se odnose na preciznije opisivanje konteksta interakcije čoveka i sistema. U tom pogledu potrebno je detaljnije opisati *profile korisnika sistema*, *okruženje u kojem se odvija interakcija i uređaje sa kojima čovek interaguje*. Uvođenjem profila čoveka u ranim fazama razvoja moguće je predvideti više različitih varijanti korisničkih interfejsa, gde je svaka varijanta prilagođena određenom tipu korisnika. Klasifikacija tipova može biti izvršena na osnovu izabranog kriterijuma i primenom odgovarajućih testova kojima će se ispitati relevantne osobine čoveka za dati kontekst. Opis okruženja interakcije, kao što su lokacija i tip prostora i uslovi u okruženju su značajni zbog rasprostranjene upotrebe prenosivih uređaja. Ovi uređaji se danas koriste u vrlo raznolikim uslovima, i zato je važno da se interakcija između čoveka i računara prilagodi prostoru i uslovima u kojima se ta interakcija zaista odvija. Ovo nas dovodi i do mogućnosti opisa uređaja na kojima će korisnički interfejs biti razmešten, pre svega sa aspekta kapaciteta obrade i tehnika interakcije sa čovekom koje uređaj koristi. Navedena proširenja mogu da budu značajna u kasnijim aktivnostima razvoja, prvenstveno u analizi i projektovanju, jer se mnoge odluke mogu zasnivati na preciznije definisanim zahtevima. Uzimajući u obzir iterativan i inkrementalan karakter razvoja, u sukscesivnim iteracijama moguće je proširiti ili sažeti postojeće specifikacije zahteva tako da one obuhvataju relevantne aspekte kao što su poželjni profili korisnika, okruženja ili uređaja.

6.1.2. Analiza

Analiza je faza u kojoj se specificirani zahtevi preciznije definišu i struktuiraju. Osnovni cilj ove aktivnosti jeste ostvarivanje jasnijeg i preciznijeg razumevanja zahteva, radi lakšeg pristupanja realizaciji softvera. Analiza predstavlja prvi korak ka dizajnu. Rezultat analize je model analize koji

definiše realizaciju slučajeva korišćenja i identifikuje odgovarajuće strukture. Za razliku od specifikacije zahteva, u kojoj se daje opis sistema na jeziku koji je razumljiv i za korisnika, modeli u analizi primarno koriste jezik inženjera razvoja. U analizi se, na platformski nezavisan način, skicira realizacija funkcionalnosti unutar sistema i daje opis interne strukture sistema. Prilikom analize, sistem se opisuje pomoću dijagrama analizacionih klasa. *Unified* proces definiše tri vrste analizacionih klasa pomoću tri stereotipa: interfejsne (*eng. boundary*) klase opisuju elemente sistema preko kojih je sistem u interakciji sa okruženjem; obradne (*eng. control*) klase opisuju obradni proces, kao što su proračuni ili poslovna logika; entiteti (*eng. entity*) opisuju perzistentne elemente sistema poput baze podataka ili različitih tipova dokumenata.

Iako je moguće napraviti analogiju između interfejsnih klasa i korisničkog interfejsa, nije moguće detaljnije opisati aspekte relevantne za komunikaciju između čoveka i računara. Važno semantičko proširenje analize bila bi mogućnost *opisa interakcije između čoveka i računara nezavisnog od načina komunikacije i platforme*. Ovo bi se postiglo uvođenjem *modela zadataka i modela apstraktnog korisničkog interfejsa*. Na taj način je moguće proveriti i međuzavisnost različitih faktora (ljudskih i računarskih faktora), tako da se u ranoj fazi izrade korisničkih interfejsa može doći do zaključaka da li su neke od odluka ispravne sa stanovišta opštih principa interakcije između čoveka i računara.

6.1.3. Dizajn

U dizajnu se sistem oblikuje do njegove konačne forme, u skladu sa svim funkcionalnim i nefunkcionalnim zahtevima. Opis specifikacije zahteva i uopštenu strukturu sistema, dobijenu analizom, u dizajnu preciznije definišemo i razrađujemo. U dizajnu se detaljno razrađuju i nefunkcionalni zahtevi, poput upotrebe programskih jezika, komponenti, operativnih sistema, distribuiranih tehnologija, tehnologija baza podataka i tehnologija korisničkih interfejsa. Dizajn predstavlja apstrakciju implementacije. Takođe, u dizajnu sistem dekomponujemo u delove sa kojima se jednostavnije radi i upravlja. Dizajn pomoću odgovarajuće notacije obezbeđuje i vizuelizaciju strukture i funkcionalnosti sistema. U fazi dizajna se donosi odluka o hardverskim i softverskim platformama koje će koristiti realizovani softverski sistem. U dizajnu se izrađuju dve vrste modela: model dizajna (*eng. design model*) i model razmeštaja (*eng. deployment model*). U modelu dizajna sistem se oblikuje do detalja, zadržavajući konceptualnu strukturu definisanu u analizi koliko je to god moguće. Za razliku od analize, u kojoj kao rezultat dobijamo konceptualni, platformski nezavisan model, model dizajna je potpuniji i formalniji. Model dizajna je manje generički, jer je prilagođen implementaciji. Za opis modela dizajna se koristi više vrsta UML dijagrama uključujući dijagrame klasa, dijagrame objekata i dijagrame stanja.

Sa stanovišta kontekstno-osetljivih korisničkih interfejsa, faza projektovanja zahteva dodatne elemente, jer se pri projektovanju donosi odluka o korišćenju načina komunikacije čoveka i sistema, softverskoj platformi (sistemskoj i aplikativnoj) i hardverskoj platformi. Razvoj korisničkih interfejsa je danas značajno usložnjen činjenicom da se interakcija između čoveka i računara odvija na izuzetno velikom broju raznolikih platformi. Ova raznolikost se ogleda u pogledu velikog broja različitih

hardverskih uređaja, softverskih tehnologija za izradu korisničkih interfejsa i tehnika (i načina) komunikacije sa korisnikom. Zbog toga je potrebno obezbediti mehanizme koji mogu apstrahovati raznolikost platformi, a istovremeni očuvati semantiku interakcije u datom kontekstu. Uz to je potrebno u što je moguće većoj meri olakšati i povećati produktivnost razvoja. Zbog toga predlažemo modelovanje korisničkih interfejsa na većem broju nivoa apstrakcije. Na ovaj način se korisnički interfejs najpre specificira konceptima nezvisnim od načina komunikacije i platforme, a zatim se automatizovanim preslikavanjima dolazi do konkretne implementacione forme. Ovo se postiže korišćenjem UML profila korisničkih interfejsa i sistema transformacija. Profili korisničkih interfejsa su opisani u prethodnom poglavlju, dok će mehanizmi transformacija biti opisani u narednom poglavlju.

6.1.4. Implementacija

Implementaciji se pristupa na osnovu dizajna. Osnovni rezultat implementacije predstavljaju implementacione komponente zajedno sa modelom implementacije. Komponente mogu biti izvorni kôd, izvršne datoteke i binarne datoteke različitih formata. Model implementacije opisuje kako se projektni koncepti, poput klasa, preslikavaju u neku od navedenih komponenti.

U našem pristupu modeli implementacije se generišu iz modela korisničkih interfejsa prilagođenih tehnologiji implementacije.

6.1.5. Testiranje

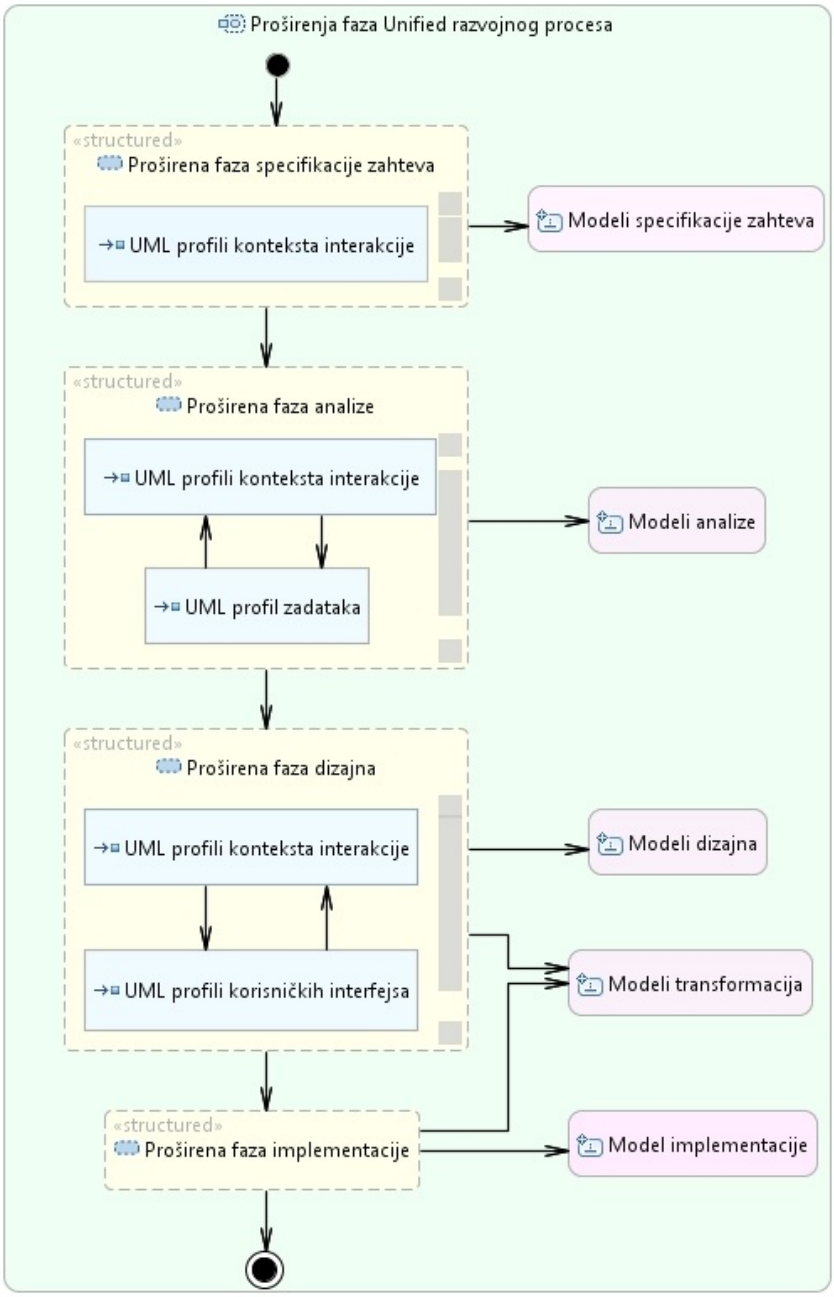
Nakon implementacije, pristupa se testiranju rešenja kako bi se proverilo da li programski sistem ispunjava sve postavljene funkcionalne i nefunkcionalne zahteve. U modelu testiranja se definišu testni slučajevi, procedure za testiranje, kao i testne komponente. Testni slučaj opisuje koji deo sistema se ispituje, šta se ispituje, šta je ulaz, a šta izlaz ispitivanja, i u kojim se uslovima testiranje izvodi. Testna procedura definiše kako se jedan ili više testnih slučajeva realizuju. Testne komponente automatizuju jednu ili više testnih procedura. Testne komponente mogu da se realizuju korišćenjem "skript" jezika ili nekog programskog jezika. Da bi se olakšalo i unapredilo testiranje softverskih sistema razvijeni su okviri za testiranje kao što je *JUnit* koji automatizuju generisanje testnih slučajeva, procedura i komponenti.

Za razliku od drugih softverskih sistema, u kojima se pri testiranju proverava da li sistem zadovoljava postavljene funkcionalne i nefunkcionalne zahteve, testiranje korisničkih interfejsa je znatno složeniji problem. Sa stanovišta korisničkih interfejsa, faza testiranja treba da, pored ispitivanja funkcionalnosti, utvrdi da li su zadovoljene zahtevane performanse interakcije i ergonomske karakteristike interfejsa. Prilikom ispitivanja korisničkih interfejsa potrebno je u obzir uzeti veliki broj ljudskih faktora koji nisu direktno povezani sa funkcionalnošću sistema. Ovde postoje specijalizovani testovi za merenje performansi i kapaciteta interakcije čoveka i računara [Farmer03]. Na primer, teorija kognitivnog naprezanja je jedna od najčešće korišćenih teorijskih baza za predviđanje i merenje performansi interakcije čoveka i računara [Oviatt06]. Ovde je krajnji cilj dizajn korisničkih interfejsa koji će obezbediti optimalno korišćenje čovekovih kognitivnih kapaciteta. Zbog složenosti, niza

specifičnosti i mnogo šireg opsega u odnosu na sam razvojni proces, proširenja faze testiranja sa stanovišta ergonomskih karakteristika korisničkog interfejsa nisu razmatrana.

6.2. *Predlog proširenja standardnog procesa razvoja*

U prethodnom odeljku smo opisali faze *Unified* procesa sa kritičkim osvrtom na nedostatke sa stanovišta razvoja kontekstno-osetljivih korisničkih interfejsa. U ovom odeljku opisujemo predložena proširenja *Unified* procesa uvođenjem razvijenih UML profila u pojedine faze razvoja. Na slici 6.1 su u vidu dijagrama aktivnosti prikazana proširenja pojedinih faza i modeli koji se u njima koriste.



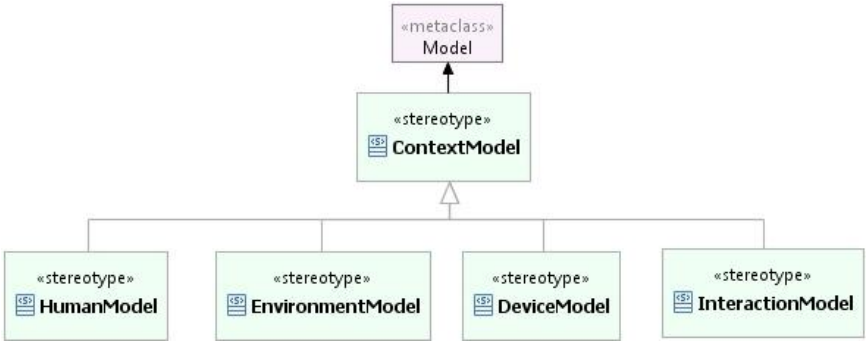
Slika 6.1. *Proširenja pojedinačnih faza procesa razvoja posmatrana kroz proširenja modela koji se u njima koriste.*

6.2.1. Proširenje specifikacije korisničkih zahteva

U skladu sa ranije opisanim nedostacima, predlažemo da se faza specifikacije zahteva profilima modela kontekstno-osetljive interakcije čoveka i računara. Drugim rečima, fazu je potrebno proširiti sledećim profilima:

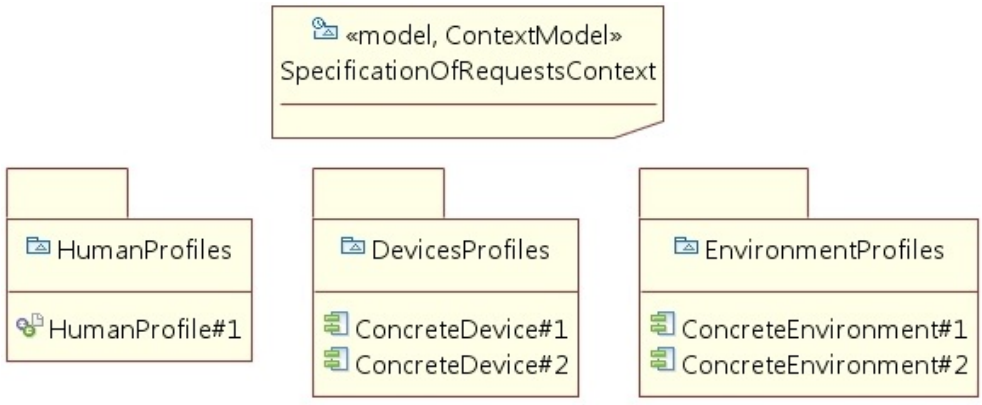
- Profil čoveka,
- Profil okruženja interakcije,
- Profil računarske platforme.

Faza specifikacije zahteva se proširuje modelima zasnovanim na proširenjima definisanim u odgovarajućim profilima. Radi bolje preglednosti i jednostavnije integracije modela kontekstno-osetljive interakcije uvodimo proširenja UML metaklase *Model* kao na slici 6.2.



Slika 6.2. *Vrste modela kontekstno-osetljive interakcije kao UML proširenja.*

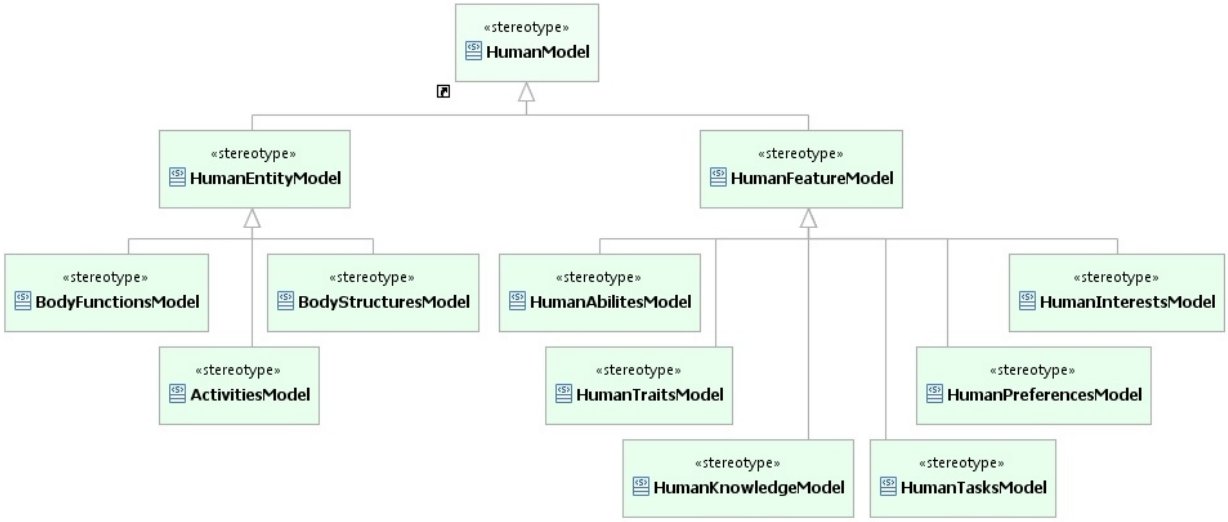
Sa tehničkog aspekta razvoja, ova klasifikacija omogućava kreiranje modela kao odvojenih celina u zasebnim datotekama ili hijerarhijsko grupisanje u jednoj datoteci. Ovakav princip rada je zastupljen u EMF okruženju, gde sam model predstavlja zaseban repozitorijum entiteta, dok se specifične vrste dijagrama generišu kao odvojene datoteke i predstavljaju različite poglede na model. Na slici 6.3 je dat jednostavan primer modela konteksta interakcije u fazi specifikacije zahteva. U narednim sekcijama model će biti razrađen po podmodelima čoveka, okruženja i platftorme.



Slika 6.3. *Pojednostavljen primer modela konteksta interakcije u fazi specifikacije zahteva.*

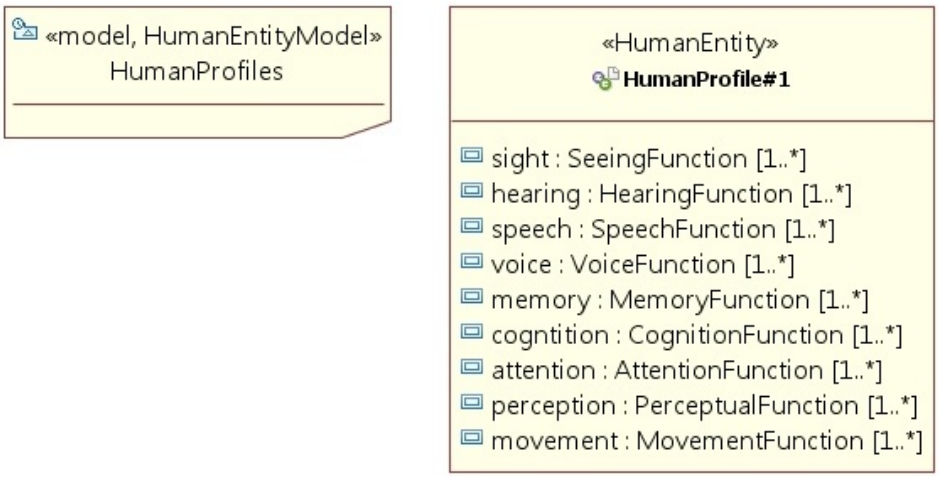
6.2.1.1. *Proširenje modelima čoveka*

Opisi čoveka će biti kreirani korišćenjem profila čoveka. Generički modeli čoveka i profili izvedeni iz njih opisani su u poglavljima 4.2.1 i 5.1.3, respektivno. Modeli čoveka opisuju osobine čoveka relevantne za specifičan kontekst korišćenja sistema. U nekim sistemima, kao što su vremenski kritični sistemi od značaja je uvođenje i opis funkcija čoveka. Sa druge strane, u sistemima tipa pretraživača ili edukativnih sistema od značaja je i modelovanje interesovanja, znanja i mehanizama učenja čoveka. Iz ovoga proizilazi da modeli čoveka mogu biti različitog nivoa složenosti. Zbog toga smo dodatno klasifikovali stereotip *HumanModel* da bismo u ranoj fazi razvoja najpre razdvojili, a kasnije i jednostavnije integrisali modele čoveka na osnovu skupa svojstava koje opisuju (slika 6.4). Na osnovu predložene klasifikacije ljudskih faktora izvedeni su model entiteta čoveka i model svojstava čoveka. Oni su dalje klasifikovani u skladu sa specifičnim skupom parametara čoveka koje opisuju.



Slika 6.4. *Klasifikacija modela čoveka.*

Na slici 6.5 je prikazan pojednostavljen modela čoveka koji predstavlja stereotip modela entiteta (*HumanEntityModel*).



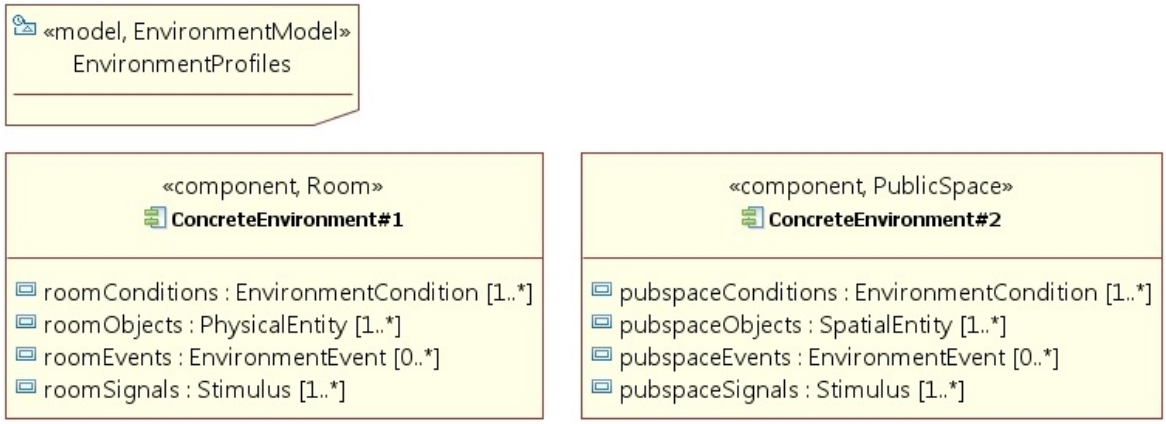
Slika 6.5. *Pojednostavljen primer modela entiteta čoveka u kojem su korišćene ICF funkcije za opis senzorskih, percepcijskih, kognitivnih i motoričkih mehanizama.*

Elementi modela su profili korisnika opisani senzorskim, percepcijskim, kognitivnim i motoričkim funkcijama čoveka preuzetim iz ICF klasifikacije. Atributi eniteta na slici su tipa UML stereotipova za opis funkcija čoveka. Multiplikativnosti ovih atributa imaju vrednost *jedan ili više* jer je svaka od funkcija preciznije opisana skupom izvedenih funkcija (videti primer klasifikacije funkcije vida čoveka u odeljku 4.2.1.1.2 - funkcije čoveka).

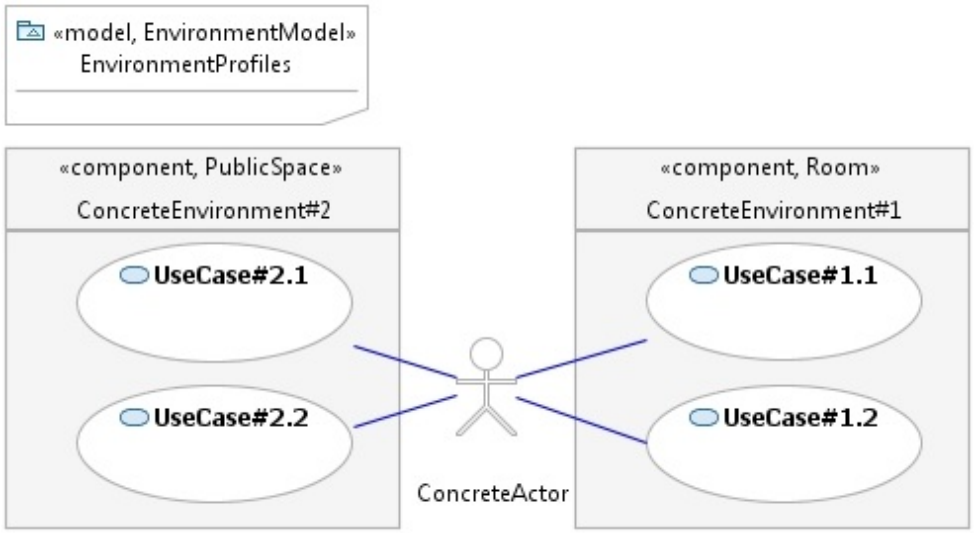
Modelovanje čoveka je od posebnog značaja u slučaju projektovanja korisničkog interfejsa za korisnike sa posebnim potrebama.

6.2.1.2. Proširenje modelima okruženja interakcije

Pored modela čoveka, potrebno je uvesti i opise okruženja i uslove u kojima se odvija komunikacija čoveka i softverskog sistema. Modeli okruženja se kreiraju na osnovu generičkih modela i iz njih izvedenih proširenja opisanih u poglavljima 4.2.2 i 5.1.1, respektivno. Na slici 6.6 je dat prikaz pojednostavljenog modela okruženja sa strukturnog aspekta i aspekta slučajeva korišćenja. U konkretnom slučaju, dva tipa okruženja (prostorija i javni prostor) opisani su atributima tipa proširenja za modelovanje uslova u okruženju, objekata, događaja i fizičkih signala.



(a)

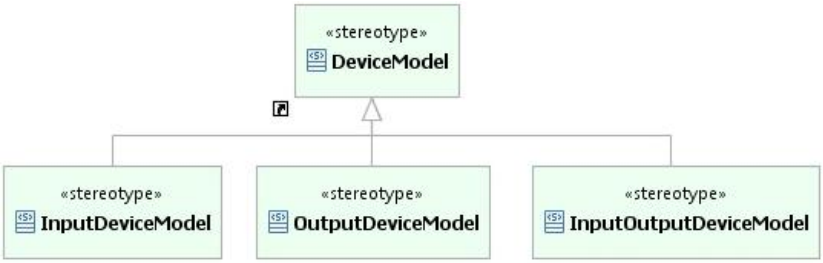


(b)

Slika 6.6. *Pojednostavljen model okruženja, (a) strukturni pogled, (b) pogled slučajeva korišćenja.*

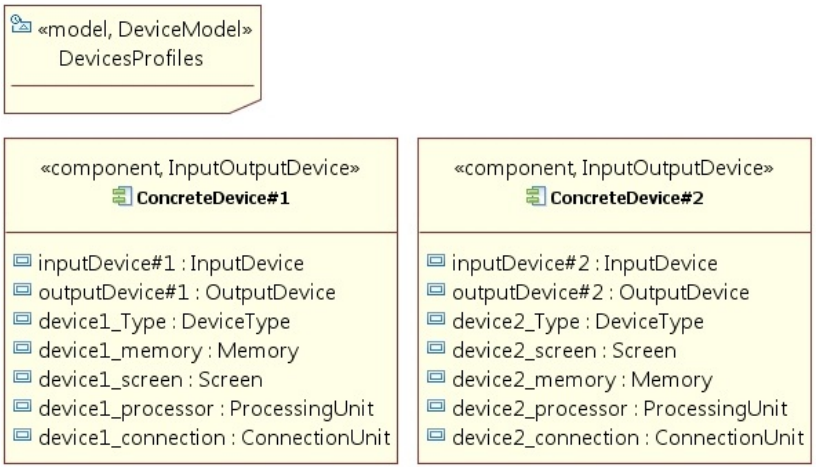
6.2.1.3. *Proširenja modelima računarske platforme*

Modeli računarskog uređaja se kreiraju korišćenjem profila računarskih uređaja opisanih u poglavlju 5.1.2. Za opisivanje uređaja na nivou modela uveli smo proširenja kao na slici 6.7.

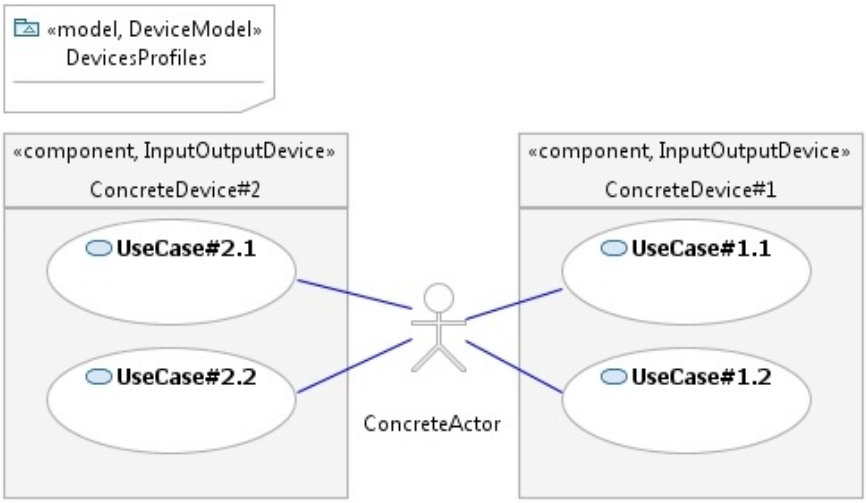


Slika 6.7. *Tipovi modela uređaja izvedeni iz osnovnog tipa.*

Na slici 6.8 je dat prikaz pojednostavljenog modela uređaja sa strukturnog aspekta i aspekta slučajeva korišćenja. U konkretnom slučaju, u fazi specifikacije zahteva su predviđene dve platforme ulazno-izlaznog tipa sa kojima korisnik može da interaguje. Konkretni uređaji su opisani osnovnim skupom atributa tipa UML proširenja za modelovanje uređaja.



(a)

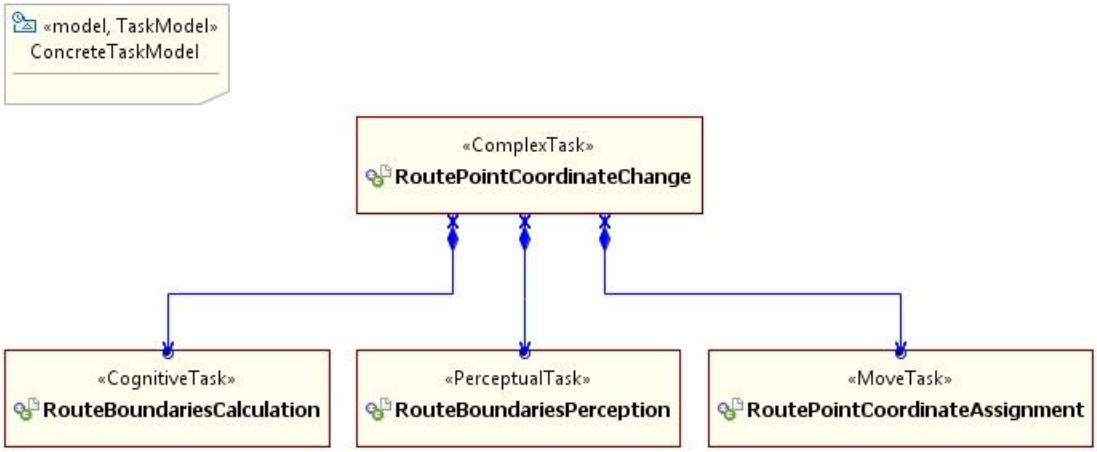


(b)

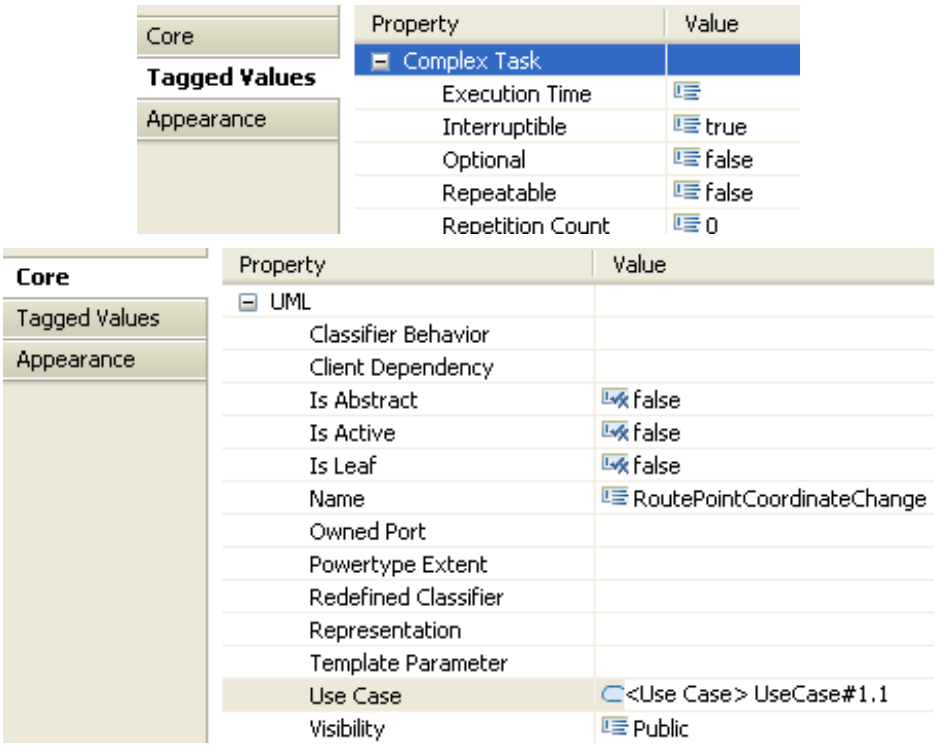
Slika 6.8. *Pojednostavljen model uređaja, (a) strukturni pogled, (b) pogled slučajeva korišćenja.*

6.2.2. Proširenje analize

U fazi analize se vrši dalja razrada projektnih odluka iz faze specifikacije zahteva. Zbog toga će u fazi analize modeli čoveka, okruženja i računarskog uređaja biti detaljnije opisani. Sa druge strane, u fazi analize se kreiraju nekompletni i apstraktni modeli koji opisuju interakciju između čoveka i računara. U ovoj fazi je potrebno opisati interakciju čoveka i računara na način nezavisan od načina komunikacije i tehnologije implementacije, tj. dati opis interakcije sa stanovišta logičkih aktivnosti i tokova informacija između čoveka i računara. Ovde ćemo koristiti proširenja za modelovanje zadataka opisana u poglavlju 5.2.1. Na slici 6.9a dat je pojednostavljen primer modela zadatka promene koordinate tačke putanje u kontekstu praćenja misija bespilotne letelice.



(a)



(b)

Slika 6.9. *Primer korišćenja proširenja za modelovanje zadataka. (a) Struktura složenog zadatka. (b) Deo za specifikaciju označenih vrednosti i slučaja korišćenja koji zadatak realizuje.*

Zadatak je složen i sastoji se od ulaznog zadatka tipa promene pozicije tačke prevlačenjem, zadatka percepcije granica promenjene putanje i zadatka računanja granica putanje angažovanjem kognitivnih funkcija čoveka. Prvi zadatak je interaktivne prirode, dok su preostala dva vezana za čoveka. Na slici 6.9b prikazani su delovi za specifikaciju označenih vrednosti zadatka i slučaja korišćenja koji zadatak realizuje. Ovakav interfejs za kreiranje modela je specifičan za EMF okruženje. U konkretnom primeru, zadatak realizuje slučaj korišćenja *UseCase#1.1* identifikovan u prethodnoj fazi (slike 6.6 i 6.8). Na osnovu tih modela možemo reći da će korisnik vršiti zadatak u zatvorenoj prostoriji (*ConcreteEnvironment#1*) korišćenjem specifičnog ulazno-izlaznog uređaja (*ConcreteDevice#2*).

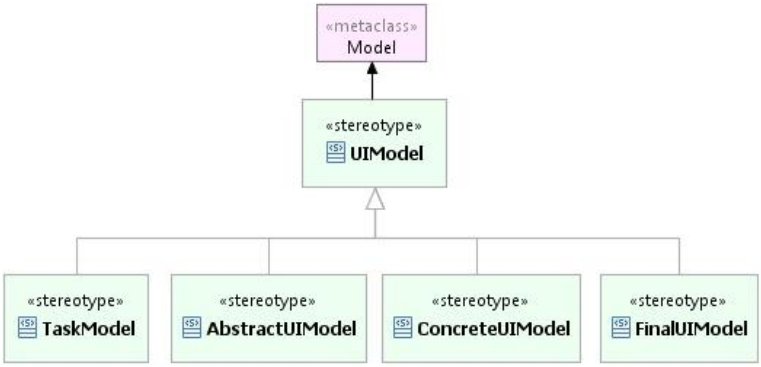
6.2.3. Proširenje dizajna

U fazi dizajna je poželjno obezbediti mehanizme koji će olakštati specifikaciju korisničkog interfejsa sa stanovišta korišćenih načina komunikacije, a nakon toga i sa stanovišta korišćene tehnologije implementacije. Donošenje odluka u vezi sa pomenutim aspektima je omogućeno na osnovu modela konteksta interakcije iz prethodnih faza. Svojstva modela korisnika i okruženja interakcije će uticati na izbor specifičnih načina komunikacije, dok će modeli uređaja u određenoj meri odrediti izbor tehnologije implementacije. Imajući u vidu prethodno navedene probleme, a u cilju unapređenja faze dizajna, predvideli smo kreiranje modela korisničkih interfejsa na većem broju nivoa apstrakcije (tabela 6.1). Oslanjajući se na podatke iz specifikacije zahteva i analize kreira se i detaljnije opisuje model zadataka. Ovaj model je zapravo kreiran u fazi analize kao mehanizam realizacije slučajeva korišćenja iz faze specifikacije zahteva. Nakon toga, kreira se model apstraktnog korisničkog interfejsa u kojem su zadaci pretočeni u primitive korisničkog interfejsa visokog nivoa apstrakcije koje nisu vezane za određen način komunikacije ili tehnologiju. Sledeći korak predstavlja kreiranje modela u kojem bi apstraktne komponente korisničkog interfejsa bile preslikane u komponente koje angažuju specifične načine komunikacije. Ovi modeli bi bili dodatno prošireni modelima kontekstno osetljive interakcije, tj. načinima komunikacije, efektima i faktorima konteksta koji su opisani u poglavlju 4.2.4. Ovakav model bi se na kraju preslikao u model prilagođen određenoj tehnologiji implementacije. Definisanje mehanizama preslikavanja između koncepata na predloženim nivoima omogućava automatizaciju dela razvoja. O transformacijama između modela korisničkih interfejsa će biti reči u narednom poglavlju.

Tabela 6.1. *Hijerarhijska organizacija modela dizajna u skladu sa predloženim nivoima apstrakcije.*

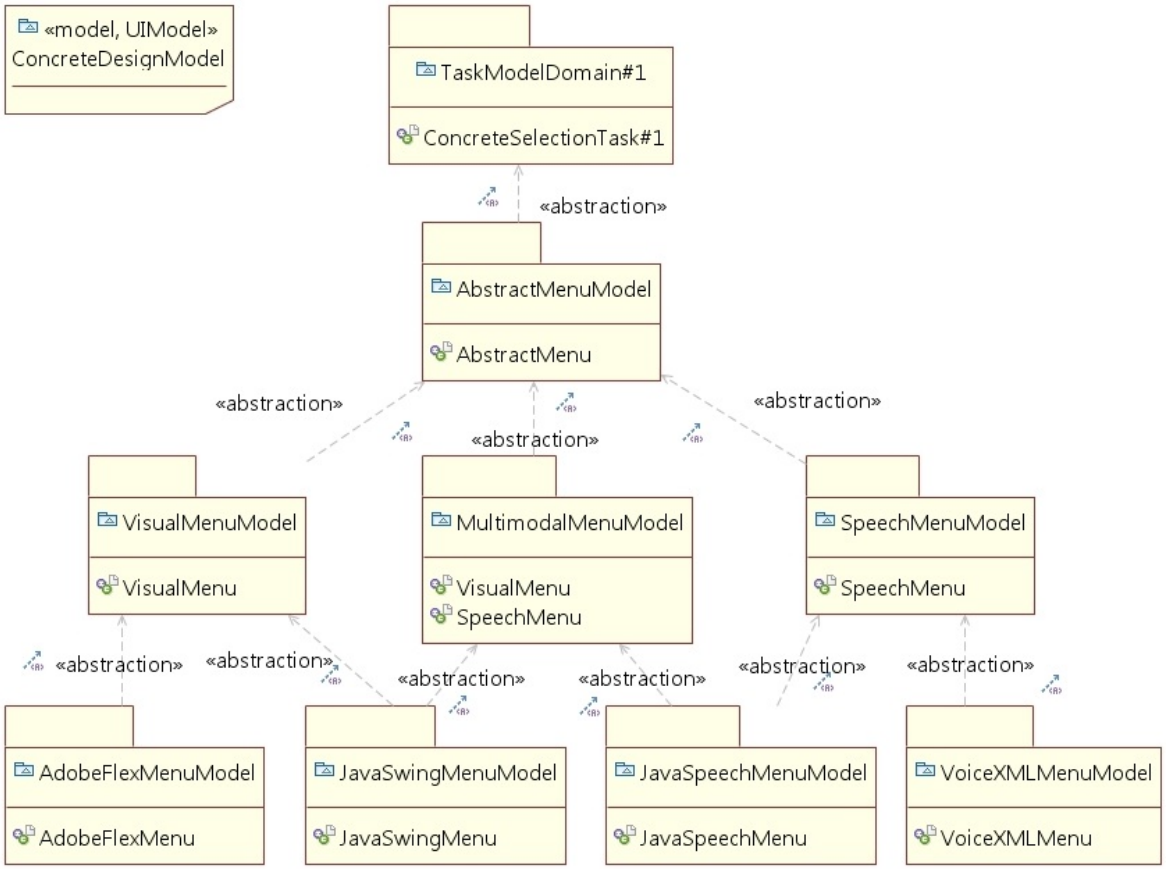
Model korisničkog interfejsa	Semantika modela
Model zadataka	*Opis korisničkog interfejsa sa stanovišta zadataka korišćenja.*
Model korisničkog interfejsa nezavisan od načina komunikcije (apstraktni korisnički interfejs)	*Opis korisničkog interfejsa sa stanovišta komponenti koje realizuju zadatke, ali su opisane nezavisno od načina komunikacije.*
Model korisničkog interfejsa zavisan od načina komunikcije (konkretni korisnički interfejs)	*Opis korisničkog interfejsa u skladu sa korišćenim načinima komunakacije. Na primer, vizuelni, govorni ili višenačinski interfejs.*
Model korisničkog interfejsa prilagođen tehnologiji implementacije (konačni korisnički interfejs)	*Opis korisničkog interfejsa prilagođen specifičnoj tehnologiji. Na primer, govorni interfejs može biti opisana JavaSpeech ili VoiceXML modelom.*

Za definisanje modela korisničkih interfejsa na predloženim nivoima apstrakcije predlažemo korišćenje UML proširenja opisanih u poglavlju 5.2.4. Dodatno smo uveli proširenja na nivou UML metaklase *Model* kako bi razdvojili i obezbedili različite mehanizme grupisanja odgovarajućih modela (slika 6.10).



Slika 6.10. *UML proširenja modela korisničkih interfejsa.*

Na slici 6.11 prikazana je hijerarhija modela korisničkih interfejsa. Radi preglednosti i lakšeg razumevanja za konkretan primer je odabran meni kao standardna kontrola korisničkih interfejsa.



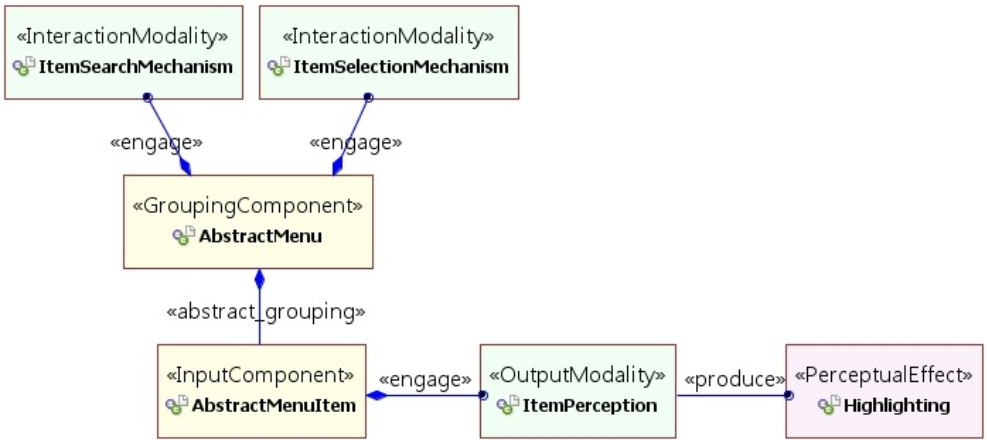
Slika 6.11. *Pojednostavljen primer hijerarhije modela korisničkih interfejsa na slučaju menija kao standardne kontrole.*

Na vrhu hijerarhije je model zadataka (*TaskModelDomain#1*) koji između ostalih obuhvata konkretan zadatak selekcije između skupa ponuđenih opcija (*ConcreteSelectionTask#1*). Model zadataka se

preslikava u apstraktni model menija (*AbstractMenuModel*). Apstraktni model menija se dalje može preslikati u modele menija prilagođene vizuelnim, govornim i višenačinskim platformama (*VisualMenuModel*, *SpeechMenuModel i MultimodalMenuModel*). Iz ovih modela su izvedeni modeli konačnih korisničkih interfejsa vezani za tehnologije implementacije. Tako će se vizuelni interfejs preslikati u modele *AdobeFlex* i *JavaSwing* platformi, dok će govorni interfejs biti opisan terminima *JavaSpeech* i *VoiceXML* platformi. Kod višenačinskih platformi treba voditi računa o mogućnosti integracije tehnologija. U konkretnom slučaju, *JavaSwing* i *JavaSpeech* tehnologije su razvijene nad zajedničkom *Java* platformom sa stanovišta specifikacije jezika i izvršnog okruženja. Modeli na različitim nivoima apstrakcije su povezani standardnim UML stereotipom relacije zavisnosti (*abstraction*).

U nastavku ćemo detaljnije opisati modele menija na različitim nivoima apstrakcije. Modeli daju objedinjen i integrisan pogled na korisnički interfejs sa aspekta samih kontrola interfejsa, kao i sa aspekta interakcije čoveka i računara. Kontrole korisničkih interfejsa su opisane proširenjima za modelovanje korisničkih interfejsa (poglavlje 5.2), dok su entiteti vezani za interakciju čoveka i računara opisani konceptima kontekstno-osetljive interakcije (poglavlje 4.2.4). Opis korisničkog interfejsa proširen je konceptima načina komunikacije i efekata. Na ovaj način je omogućena sprega faktora konteksta i korisničkog interfejsa kao što je prikazano na slici 4.20.

Na slici 6.12 dat je opis menija na nivou apstraktnog korisničkog interfejsa.

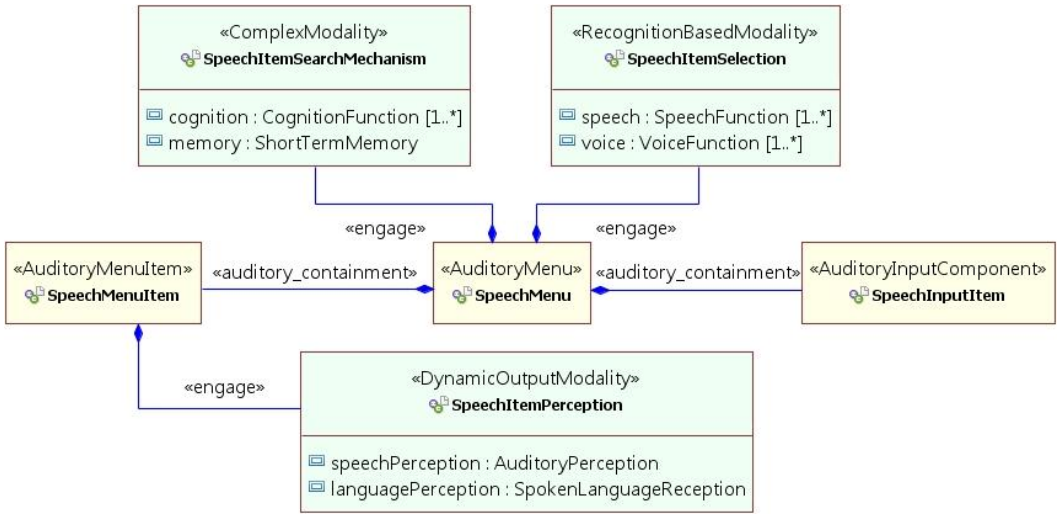


Slika 6.12. *Apstraktni opis menija sa stanovišta korišćenih načina komunikacije.*

Apstraktni meni sadrži skup stavki i modelovan je kao apstraktni stereotip grupisanja (*GroupingComponent*). Stavka je opisana kao ulazna komponenta (*InputComponent*) sa stanovišta korisničkog interfejsa. Apstraktni meni angažuje načine komunikacije sa čovekom u obliku mehanizma pretrage stavki menija (*ItemSearchMechanism*) i mehanizma selekcije stavki menija (*ItemSelectionMechanism*). Kako u ovom trenutku konkretan način komunikacije sa čovekom još nije definisan, mehanizmi su modelovani odgovarajućim stereotipom (*InteractionModality*). Sa stanovišta komunikacije sa čovekom, stavka menija angažuje odgovarajući mehanizam percepcije koji je modelovan kao izlazni način komunikacije. Sam mehanizam percepcije kreira percepcijski efekat isticanja stavke.

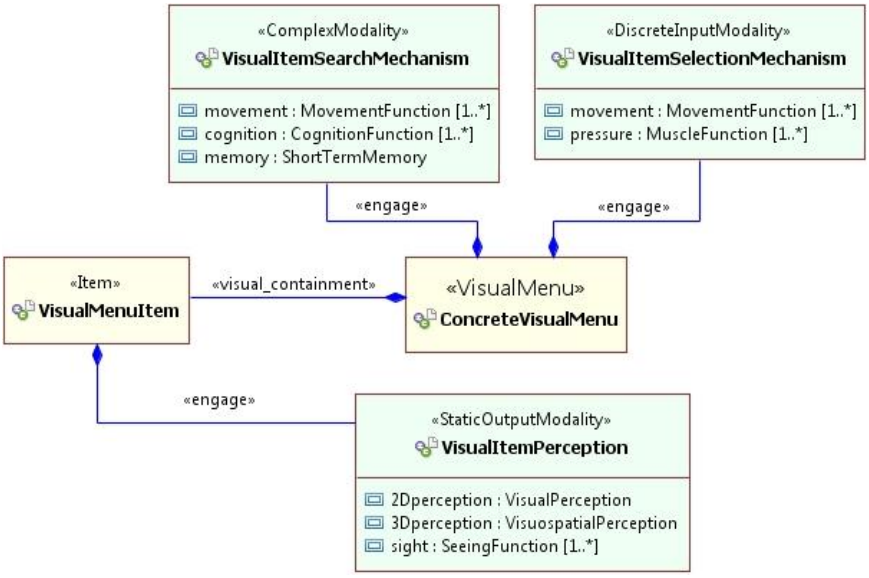
U slučaju korišćenja načina komunikacije vezanog za govor, apstraktni meni će se preslikati u

odgovarajući model konkretnog korisničkog interfejsa (slika 6.13). Govorni meni (*SpeechMenu*) sadrži stavke menije (*SpeechMenuItem*) koje su opisane kao izlazne komponente i ulaznu komponentu za prijem govornih podataka (*SpeechMenuItem*). Stavka menija angažuje ljudski mehanizam percepcije stavke menija koji je modelovan kao dinamički izlazni način komunikacije. Mehanizam percepcije stavke menija zahteva angažovanje specifičnih funkcija čoveka. Ovo je opisano atributima tipa ICF funkcija čoveka vezanih za percepciju govora i prijem informacija izgovorenog jezika. Meni kao složena komponenta angažuje mehanizme pretrage govorne stavke (*SpeechItemSearchMechanism*) i selekcije govorne stavke (*SpeechItemSelection*). Mehanizam pretrage govorne stavke je modelovan kao složen modalitet i podrazumeva angažovanje kognitivnih funkcija čoveka i funkcija kratkoročne memorije. Mehanizam selekcije stavke je modelovan kao ulazni modalitet prepoznavanja i angažuje glasovne i govorne funkcije čoveka. Zbog preglednosti i boljeg razumevanja, u opisivanju modela konkretnih korisničkih interfejsa naglasak je na faktorima čoveka.



Slika 6.13. *Opis glasovnog menija korišćenjem predloženih proširenja.*

Na slici 6.14 dat je opis vizuelnog menija.

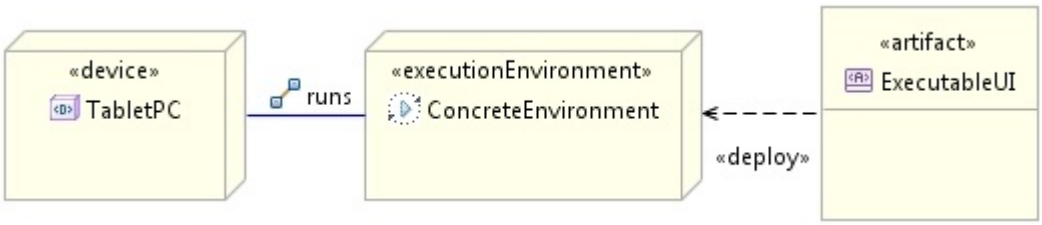


Slika 6.14. *Opis vizuelnog menija korišćenjem predloženih proširenja.*

Vizueni meni (*ConcreteVisualMenu*) se sastoji od stavki menija (*VisualMenuItem*). Stavka angažuje mehanizam percepcije stavke koji je modelovan kao statički izlazni modalitet. Mehanizam percepcije vizuelne stavke zahteva funkcije čoveka vezane za dvodimenzionalnu i trodimenzionalnu percepciju prostora, kao i senzorske funkcije čula vida. Vizuelni meni kao složena komponenta angažuje mehanizme pretrage i selekcije vizuelne stavke. Mehanizam vizuelne pretrage je modelovan kao složen modalitet koji angažuje kognitivne funkcije čoveka, funkcije pokreta i funkcije kratkoročne memorije. Selekcija vizuelne stavke je modelovana kao diskretni ulazni modalitet koji zahteva angažovanje funkcije čoveka vezanih za motoričke pokrete i funkcije mišića. Tipovi angažovanih funkcija su odgovarajuća UML proširenja ICF klasifikacije.

Model korisničkog interfejsa zavisan od načina komunikacije će se preslikati u model korisničkog interfejsa prilagođen konkretnoj platformi. Ovi modeli su opisani odgovarajućim metamodelima tehnologija implementacije. Mehanizmi kreiranja i primeri ovih metamodela su opisani u poglavlju 5.2.4.

Izvršna arhitektura projektovanog korisničkog interfejsa se može skicirati korišćenjem postojećih primitiva za modelovanje fizičkog razmeštaja softverskog sistema [UML10]. Na slici 6.15 dat je jednostavan primer dijagrama raspoređivanja koji ilustruje upotrebu postojećih UML proširenja za opis fizičke distribucije korisničkog interfejsa. Računarski uređaj (*TabletPC*), softversko izvršno okruženje (*ConcreteEnvironment)*, izvršna verzija korisničkog interfejsa (*ExecutableUI*), kao i relacije između njih, opisani su kao stereotipovi koji predstavljaju sastavni deo specifikacije jezika UML.



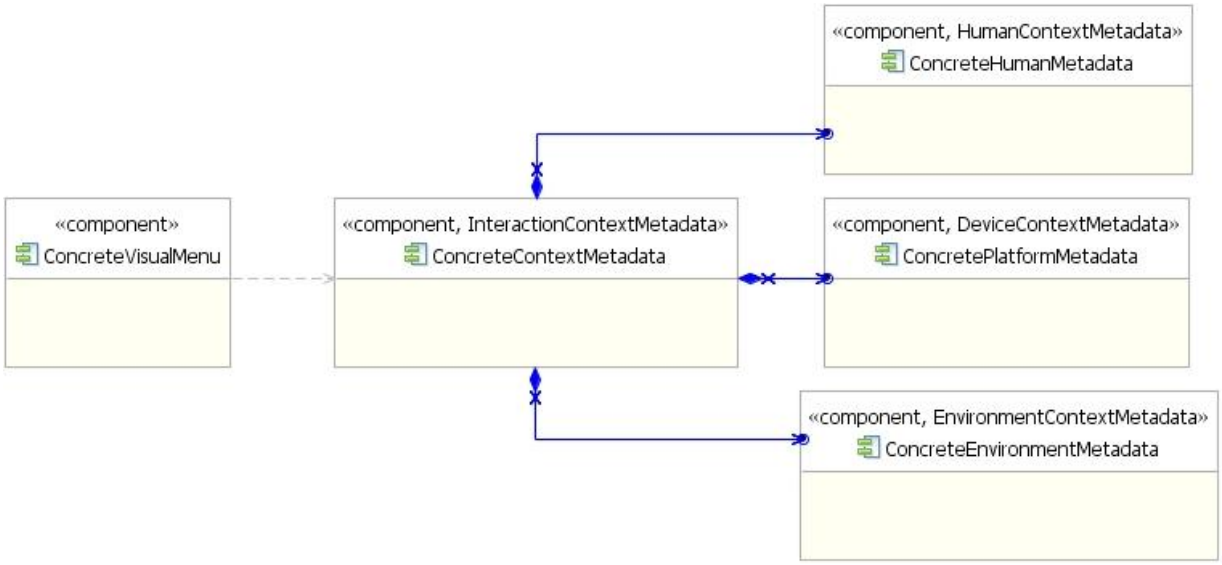
Slika 6.15. *Dijagram razmeštaja koji opisuje izvršnu arhitekturu korisničkog interfejsa.*

6.2.4. Proširenje implementacije

Modeli implementacije izvedeni su iz modela dizajna, tj. modela korisničkih interfejsa prilagođenih tehnologiji implementacije. Ovde se dizajnerske klase preslikavaju na implementacione komponente. Problem postojanja velikog broja tehnologija za izradu korisničkog interfejsa se može ublažiti definisanjem transformacija između modela dizajna i izvršnih komponenti. U kontekstu arhitekture vođene modelima, ovde govorimo o M2T (*eng. model-to-text*) transformacijama. Postojeća okruženja za rad sa UML jezikom, kao što je EMF, imaju integrisane mehanizme za generisanje programskog kôda za različite jezike, kako imperativne, tako i deklarativne.

Pored toga, u modelima implementacije je potrebno generisati komponente sa podacima vezanim za kontekstnu-osetljivu interakciju čoveka i računara koji potiču iz specifikacije zahteva, analize i dizajna. Ovi podaci su od značaja u pogledu mogućnosti adaptacije izvršne verzije korisničkog interfejsa, kao i sa aspekta analize postojećih korisničkih interfejsa. Zbog toga smo uveli odgovarajuća proširenja UML

komponente koja opisuju podatke vezane za kontekstno-osetljivu interakciju na implementacionom nivou (slika 6.16).



Slika 6.16. *Dijagram komponenti – pojednostavljen primer korišćenja proširenja za modelovanje podataka konteksta interakcije na nivou implementacije.*

Faza implementacije je usko povezana sa transformacijama modela dizajna niskog nivoa asptrakcije u programski kôd. O ovome će biti reči u narednom poglavlju.

6.3. *Rezime poglavlja*

U ovom poglavlju smo opisali pristup proširenju standardne metodologije razvoja softverskih sistema konceptima kontekstno-osetljive interakcije čoveka i računara. Proširenje se ogleda u integraciji razvijenih UML profila u pojedinim fazama razvoja. Drugim rečima, razmatrali smo upotrebu predloženih UML proširenja za detaljniji opis kontekstno-osetljive interakcije čoveka i računara u fazama specifikacije zahteva, analize, dizajna i implementacije.

U narednom poglavlju govorićemo o transformacijama modela korisničkih interfejsa. Tehnologije i alati za transformacije modela mogu da se koriste zajedno sa predloženim proširenjima jer između opisanih modela postoji niz međuzavisnosti. Na taj način je moguće automatizovati deo procesa razvoja.

7. Transformacije modela kontekstno-osetljivih korisničkih interfejsa

Kao što je opisano u prethodnom poglavlju, modelski zasnovan razvoj korisničkih interfejsa obuhvata kreiranje modela koji opisuju različite aspekte interfejsa i koji su definisani na različitim nivoima apstrakcije. Osnovni cilj modelski zasnovanog pristupa jeste povećanje produktivnosti i skraćenje vremena razvoja softverskog sistema tako što razvojni inženjeri mogu da koriste koncepte koji su bliži domenu problema, umesto da koriste koncepte iz domena tehnologije implementacije. Jedan od ključnih izazova modelski zasnovanog razvoja softvera jeste definisanje transformacija između modela različitih nivoa apstrakcije. U domenu projektovanja korisničkih interfejsa, sa jedne strane je potrebno očuvati semantiku interakcije čoveka i računara. Sa druge strane, cilj je doći do modela specifičnih implementacionih platformi koji se mogu direktno transformisati u programski kôd interfejsa.

Transformacije su predstavljene kao modeli sastavljeni od skupa transformacionih pravila, gde se kao ulaz koriste jedan ili više izvornih modela, a kao izlaz se dobija jedan ili više transformisanih modela. Postoji veći broj generičkih mehanizama transformacija modela, kao što su na primer, direktna manipulacija sa sadržajem modela, pristup vođen strukturom, relacioni pristup, transformacije zasnovane na grafovima, pristup zasnovan na šablonima ili hibridni pristup [Czarnecki06].

Inženjerstvo upravljano modelima je omogućilo podizanje nivoa apstrakcije u opisu transformacija tako da su modeli transformacija opisani odgovarajućim metamodelima. Neke od prednosti ovog pristupa sa aspekta razvoja korisničkih interfejsa su [Coutaz10]:

- Korišćenje i ponovna upotrebljivost postojećeg znanja: često upotrebljavane transformacije se mogu organizovati kao šabloni u bibliotekama koje se mogu koristiti u različitim alatima i na taj način omogućiti konzistentnost korisničkih interfejsa;
- Sistematičan i kontrolisan mehanizam prilagođavanja korisničkih interfejsa raznovrsnim kontekstima upotrebe korišćenjem različitih transformacija;
- Mogućnost komparativnih evaluacija korisničkih interfejsa;
- Mogućnost transformacija samih transformacionih modela, što dalje pruža moćan formalan mehanizam rekurzije u smislu adaptacije korisničkog interfejsa specifičnom kontekstu korišćenja, a da se pri tome očuva njegova upotrebljivost.

U ovom poglavlju opisujemo mogućnosti transformacija modela kontekstno-osetljivih korisničkih interfejsa. Osnovu za kreiranje ovih modela predstavljaju proširenja vezana za kontekstno-osetljivu interakciju čoveka i računara koja su opisana u prethodnom poglavlju. Najpre dajemo sažet pregled postojećih sistema transformacija u kontekstu razvoja korisničkih interfejsa. Zatim dajemo predlog sistema transformacija sa stanovišta konkretne tehnologije. Nakon toga, dajemo detaljniji opis specifikacije sistema transformacija koji polazi od modela zadataka i doseže do implementacionih modela vezanih za konkretne tehnologije.

7.1. *Postojeći pristupi u dizajnu transformacija modela korisničkih interfejsa*

U ovom odeljku dajemo sažet pregled značajnijih novijih istraživanja u oblasti transformacija modela korisničkih interfejsa na različitim nivoima apstrakcije koje uzimaju obzir kontekst interakcije između čoveka i računara.

U istraživanju [Aquino09] je opisan princip korišćenja transformacionih profila (*eng. transformation profiles*) u modelski zasnovanom razvoju korisničkih interfejsa. Transformacioni profil čini skup mapiranja modela (*eng. mapping model*) i transformacioni šablon (*eng. transformation template*). Sa jedne strane, mapiranja modela povezuju ulazni i izlazni model na fleksibilan način i predstavljaju preslikavanja između odgovarajućih elemenata oba modela. Sa druge strane, transformacioni šablon čine parametri koji specificiraju detalje strukture, rasporeda i stila korisničkih interfejsa. Namena transformacionih šablona je parametrizacija transformacija modela korisničkih interfejsa. Svaki parametar je odgovarajućeg tipa koji određuje elemente interfejsa na koje je parametar primenjiv, kao i elemente na koje parametar može uticati. Tipu parametra je pridružen i prioritet koji se koristi u rešavanju konflikata kada dva ili više parametara imaju uticaj na isti element interfejsa. Pored toga, tip parametra je opisan i odgovarajućim tipom podatka ili skupom mogućih vrednosti. Primera radi, tipovi parametara mogu predstavljati jednostavna svojstva elemenata interfejsa (kao što su boje ili fontovi) ili se mogu odnositi na složenije atribute jednog ili većeg broja elemenata (kao što su *layout* opcije, stilovi dijaloga, pozicija objekata, ravnanje elemenata). Parametri iz transformacionih šablona mogu redefinisati preslikavanja određena mapiranjima modela. Drugim rečima, mapiranja mogu biti izražena kao parametar u transformacionom šablonu. Zamišljeno je da ovakva organizacija transformacionih profila obezbedi ponovnu upotrebljivost, čitljivost i mogućnost prilagođavanja transformacija od strane dizajnera i krajnjeg korisnika. Na primer, umesto modifikacije mapiranja modela može se vršiti izbor vrednosti parametara u transformacionim šablonima pomoću posebnog editora [Aquino10]. Pored toga, različiti transformacioni šabloni mogu obuhvatati različite hardverske i softverske platforme, kao i karakteristike korisnika. Pristup zapravo predstavlja integraciju transformacionog i šablonskog pristupa u razvoju korisničkih interfejsa. Važno je napomenuti da je metamodel transformacionih šablona tesno povezan sa specifikacijom UsiXML jezika i omogućava generisanje deklaratvnih oblika transformacija u XML kontekstu. Posmatrajući metamodel transformacionih šablona može se uočiti da je njihovo korišćenje ograničeno na grafičke korisničke interfejse.

U radu [Savidis10] je opisan pristup transformacijama izvršnih modela korisničkih interfejsa na nivou programskog kôda, tj. refaktorizacija. Naglasak je na prilagođavanju postojećih korisničkih interfejsa bez uticaja na softversku arhitekturu. Suština pristupa je u postojanju alternativnih verzija jedne komponente korisničkog interfejsa. Na osnovu profila raspoređivanja (*eng. deployment profile)* vrše se zamene (*eng. replacement*) verzija komponenata u vreme izvršavanja.

Istraživanje[Sotte07] formalizuje korisnički interfejs kao graf modela i mapiranja između modela koji opisuju različite aspekte korisničkog interfejsa (*eng. octopus*). Autori uvode pojam mapiranja (*eng.*

mapping) kao oblik transformacija korisničkih interfejsa pri čemu je očuvana njihova upotrebljivost. Zbog toga mapiranja integrišu skup svojstava upotrebljivosti, tj transformacije su vođene principima upotrebljivosti (*eng. usability-driven transformations*). Ovde se pojam upotrebljivosti razmatra u širem smislu i podrazumeva mogućnost korišćenja korisničkog interfejsa u različitim kontekstima, tj. u različitim okruženjima i na različitim platformama. Pristup razmatra kontekst korišćenja kao kombinaciju faktora čoveka, okruženja i uređaja. Faktori konteksta vezani za okruženje i uređaje su specificirani u određenoj meri, dok faktori čoveka nisu precizirani i formalno opisani. Autori predlažu i teorijski metamodel mapiranja visokog nivoa apstrakcije. Metamodel mapiranja ostavlja mogućnost korišćenja proizvoljnog okvira upotrebljivosti, ali ne definiše mehanizme integracije sa ostalim konceptima metamodela. Ovo je važno sa obzirom da ne postoji opšti konsenzus oko kriterijuma upotrebljivosti. Na taj način bi uključivanje različitih sistema kriterijuma upotrebljivosti moglo zahtevati modifikacije predloženog metamodela mapiranja.

Pristup dizajnu pravila adaptacije korisničkih interfejsa je opisan u radu [Jaquero09]. Pravila adaptacije koriste mehanizam transformacije grafova korišćenjem XSLT jezika. Autori daju metamodel pravila adaptacije. Metamodel je generičke prirode i ne daje preciznu specifikaciju faktora konteksta. Razvijen je i vizuelni alat za specifikaciju pravila adaptacije na osnovu predloženog metamodela. Ograničenje je što alat omogućava kreiranje pravila adaptacije koja su primenjiva na modele kreirane korišćenjem UsiXML jezika. Alat zapravo generiše pravila adaptacije u XSLT tehnologiji.

Posmatrajući oblast automatizacije razvoja, tj. transformacija modela korisničkih interfejsa može se uočiti potreba za fleksibilnijim pristupom u procesu generisanja implementacija interfejsa iz modela na višim nivoima apstrakcije. Postojeći pristupi moraju integrisati fleksibilne mehanizme za specifikaciju konkretnih svojstava korisničkih interfejsa uzimajući u obzir kontekst korišćenja, tj. osobine čoveka, dostupnu hardversku i softversku platformu, kao i karakteristike okruženja.

Sa stanovišta razvoja korisničkih interfejsa, smatramo da alati za transformacije treba da podrže sledeće funkcionalnosti:

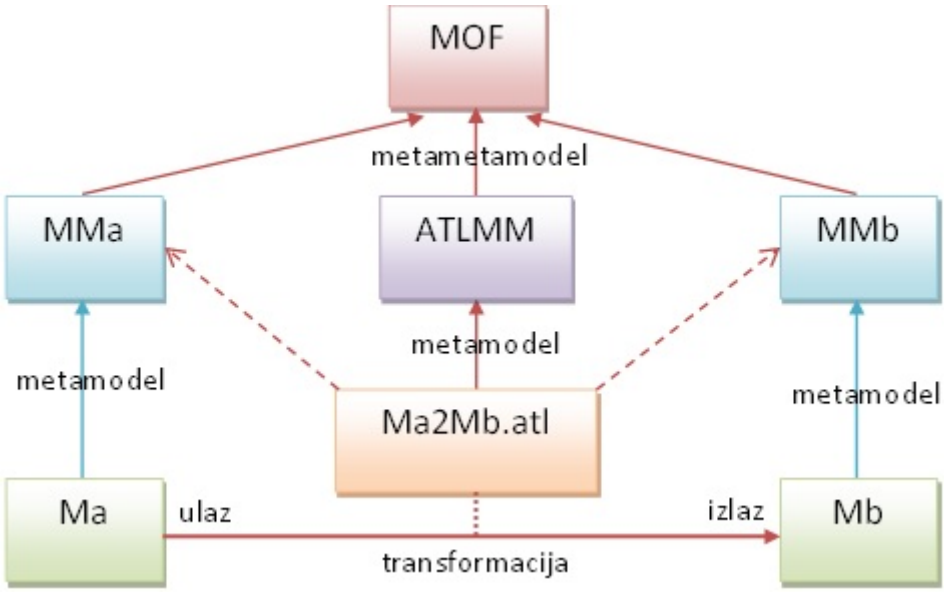
- Podrška većem broju različitih implementacionih platformi,
- Fleksibilnost koja se ogleda u tome da se unutar jedne platforme obezbedi podrška različitim strategijama implementacije, arhitekturama aplikacija i šablonima kodiranja,
- Proširivost u pogledu integracije novih transformacija,
- Podrška standardnim domenski specifičnim modelima,
- Potpunija integracija sa drugim vrstama alata koji automatizuju razvoj i održavanje softvera (alati za održavanje kôda, alati za kontrolu verzija, alati za upravljanje zahtevima, alati za automatizaciju radnih tokova, alati za testiranje i dr.).

7.2. *Predlog sistema transformacija*

U skladu sa analizom postojećih pristupa iz prethodnog odeljka, u ovom odeljku najpre opisujemo mehanizme i tehnologije transformacija za koje smo se opredelili. Nakon toga, dajemo predlog sistema transformacija modela kontekstno-osetljivih korisničkih interfejsa. Detaljnija specifikacija predloženog sistema transformacija će biti opisana u narednom odeljku.

7.2.1. Tehnologija transformacija

Kada govorimo o transformacijama razvojnih modela, opredelili smo se za korišćenja ATL jezika za transformacije [Jouault08]. Kao glavne prednosti jezika sa aspekta razvoja softvera mogu se izdvojiti otvorenost u pogledu korišćenja, mogućnosti transformacija modela, široka i lako dostupna baza znanja i zajednica korisnika, kao i podrška u vidu dostupnih alata [ATL11]. Kontekst ATL transformacije je prikazana na slici 7.1. Izvorni model *Ma* se transformiše u odredišni model *Mb* na osnovu transformacije *Ma2Mb.atl* čiji je jezik opisan *ATLMM* metamodelom. Svi metamodeli su napisani na MOF jeziku.



Slika 7.1. *Opšti scenario ATL transformacije.*

Uopšteno posmatrano, možemo odvojeno govoriti o specifikaciji samog jezika, kao i o specifikaciji okruženja za dizajn i izvršavanje transformacija. ATL je hibridni jezik za transformacije. Predstavlja mešavinu deklarativnih i imperativnih konstrukcija (tabela 7.1). Deklarativan stil specifikacije ima nekoliko prednosti. Zasnovan je na eksplicitnom zadavanju relacija između elemenata ulaznog i izlaznog modela i na taj način je intuitivniji za korišćenje od strane razvojnih inženjera. Ovaj stil naglašava relacije između elemenata modela i sakriva detalje vezane za selekciju elemenata modela, redosled okidanja i izvršavanja pravila, *traceability*, i dr. U nekim je slučajevima je komplikovano obezbediti čisto deklarativno rešenje za dati problem. Tada se mogu koristiti imperativne konstrukcije za realizaciju složenijih logika transformacije. ATL transformacije su jednodirekcione i rade na *read-only* ulaznim modelima i *write-only* izlaznim modelima. U toku izvršavanja transformacije,

omogućena je navigacija u ulaznim modelima, ali se ne mogu modifikovati. Sa druge strane, nije omogućena navigacija kroz izlazne modele. Bidirekciona transformacija se implementira kao par transformacija – po jedna za svaki smer. Transformacija je predstavljena kao skup pravila (*eng. rules*) koja specificiraju preslikavanje elemenata izvornog modela (*eng. source elements*) u elemente odredišnog modela (*eng. target elements*).

Tabela 7.1. *Pregled osnovnih primitiva ATL jezika.*

ATL koncept	Opis koncepta
standard rule	*Deklarativni element. Poziva se u trenutku nailaska na ulazni šablon u izvornim modelima. Kreira izlazni šablon u odredišnim modelima. Izvršava se za svako poklapanje u izvornom modelu i kreira novi izlazni element.*
lazy rule	*Deklarativni element. Eksplicitno se poziva iz drugog pravila. Prilikom izvršavanja kreira novi element sa istim sadržajem iz istog ulaznog elementa. Pogodan za transformacije oblika jedan prema više.*
unique lazy rule	*Deklarativni element. Eksplicitno se poziva iz drugog pravila. Kreira element izlaznog modela jednom za dati element ulaznog modela, dok sukscesivna izvršavanja istog pravila za dati element kreiraju reference na originalni izlazni element. Pogodan za transformacije tipa više prema jedan.*
called rule	*Imperativni element. Poziva se ekspicitno kao procedura/metod i može imati parametre.*
action block	*Imperativni element. Sekvenca iskaza koja se može koristiti u kombinacija sa ili umesto matched rules ili called rules. Obuhvaćene su konstrukcije tokova kontrole poput uslovnih naredbi, petlji, dodele vrednosti i dr.*
Helpers	*ATL ekvivalent Java metodama. Omogućavaju kreiranje kôda koji se može pozivati iz različitih tačaka transformacije.*
rule inheritance	*Ugrađen mehanizam nasleđivanja pravila koji omogućava dizajn polimorfnih pravila i ponovnu upotrebljivost kôda.*

Jezik integriše OMG OCL standard u pogledu tipova podataka i u pogledu deklarativnih izraza. Specifikacija jezika sa stanovišta izvršnog i razvojnog okruženja podrazumeva integrisan skup alata (*eng. ATL Development Tools - ADT*) koji predstavlja nadogradnju *Eclipse* platforme. Izvršno okruženje vrši prevođenje i izvršavanje ATL kôda, dok razvojno okruženje obezbeđuje editor za dizajn i *debug* transformacija. Važno je napomenuti da ADT sve modele interno čuva u XMI formatu.

Iako su MDE principi za kreiranje jezika za modelovanje definisani odgovarajućim standardima, u praksi se ovi jezici definišu korišćenjem različitih tehnologijakao što su na primer XML, baze podataka i MOF. Kako bi bila obuhvaćena raznovrsnost jezika i tehnologija modelovanja uveden je pojam tehnološkog prostora [Bézivin03]. Tehnološki prostor predstavlja radni kontekst sa skupom povezanih koncepata, bazom znanja, alatima i potrebnim znanjima i veštinama. Čine ga sistem reprezentacije i skup operatora za pristup i obradu podataka predstavljenih u njemu. Sa stanovišta transformacija modela korisničkih interfejsa kao jednu od prednosti ove tehnologije izdvajamo mogućnost reprezentacije i transformacije modela između različitih tehnoloških prostora (*eng. Technological Space - TS*). Sa obzirom da je komplikovano dati jasnu definiciju tehnološkog prostora, a radi boljeg razumevanja, navodimo primere [Bézivin03]: apstraktne i konkretne sintakse programskih jezika, ontološki inženjering, XML jezici i alati, sistemi za upravljanje bazama podataka, arhitektura upravljanja modelima. Svaki od navedenih primera čine odgovarajući koncepti u skladu sa definicijom.

Na primer, kod programskih jezika to su gramatika i program, XML prostor čine šeme i dokumenti, prostor baza podataka šeme i podaci, ontološki prostor čine *top-level* ontologija i iz nje izvedene ontologije, dok su ključni koncepti MDA model i metamodel. U kontekstu problema istraživanja ovog rada, možemo uspostaviti vezu između različitih tehnoloških prostora vezanih za razvoj kontekstno-osetljivih korisničkih interfejsa. U našem slučaju, sa jedne strane imamo MDA tehnološki prostor koji koristi UML jezik (pretpostavka je da su apstraktne sintakse jezika opisane metamodelima). Sa druge strane imamo tehnološke prostore vezane za različite pristupe u razvoju korisničkih interfejsa kao što su XML, UsiXML i dr., koji definišu konkretne sintakse. Na ovaj način moguće je uspostaviti formalan mehanizam integracije koncepata iz različitih istraživanja vezanih za razvoj kontekstno-osetljivih korisničkih interfejsa. U tom pogledu, ATL razvojno okruženja omogućava import/eksport različitih XML formata u/iz ATL MOF repozitorijuma modela pomoću posebnog skupa alata (tabela 7.2).

Tabela 7.2. *ATL podrška za rad sa tehnološkim prostorima.*

Alat	Opis alata
XMLinjector	*Prihvata ulazni XML dokument i kreira ekvivalentan model koji je u skladu sa XML metamodelom definisanim u MOF ili Ecore (Eclipse verzija MOF).*
XMLextractor	*Kreira XML dokument na osnovu modela koji je usklađen sa metamodelom definisanim u MOF ili Ecore.*
TCSinjector	*Prihvata tekstualni dokument napisan na domenski specificnom jeziku i kreira ekvivalentan model koji je usklađen sa metamodelom datog jezika definisanim u MOF ili Ecore.*
TCSextractor	*Kreira tekstualni dokument napisan na domenski specifičnom jeziku na osnovu modela usklađenog sa metamodelom datog jezika deifinisanim u MOF ili Ecore.*

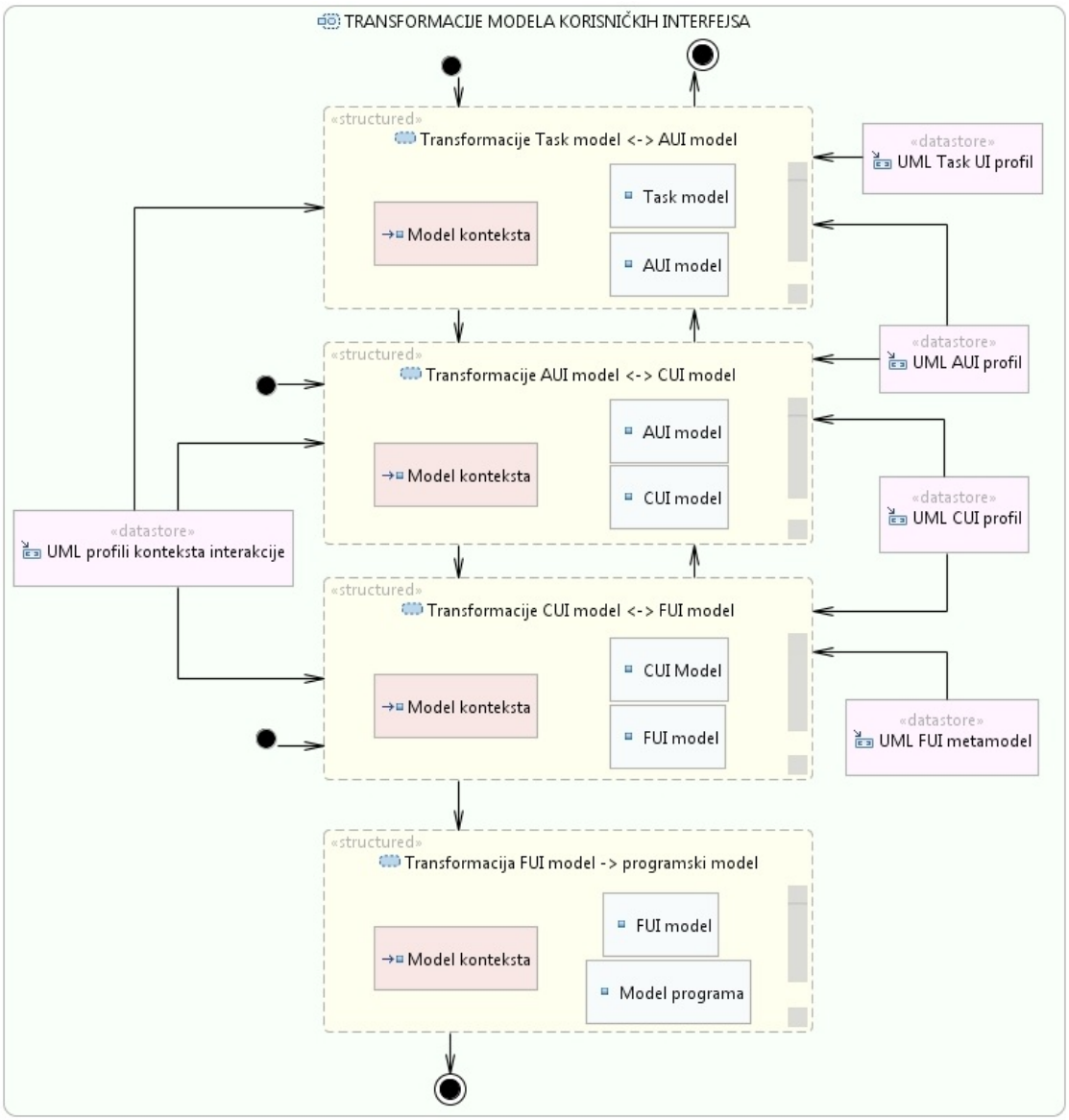
TCS (*eng. Textual Concrete Syntax*) je EMF alat koji omogućava specifikaciju tekstualnih konkretnih sintaksi domenski specifičnih jezika (DSL) tako što pridružuje sintaksnu informaciju metamodelima. Na ovaj način je moguće prebacivati DSL rečenice iz modela u tekst (*eng. extraction*) i obrnuto (*eng. injection*). Sam alat je praćen odgovarajućim editorom sa naprednim mogućnostima poput *content assist* i dr.

U pogledu tehnologija za generisanje programskog kôda iz modela izdvajaju se Acceleo i JET. Acceleo predstavlja implementaciju OMG MOF Model to Text (MTL) standarda[12] . Tehnologija je realizovana kao skup alata razvijenih nad Eclipse platformom. Alati su integrisani u razvojno okruženje koje omogućava dizajn i izvršavanje transformacija. Kao ulaz se koriste UML ili Ecore modeli. Transformacije kreiraju tekstualne datoteke u skladu sa sintaksama specifičnih programskih jezika kao što su Java, C++, C#. Pored toga, odgovarajuće razvojno okruženje omogućava kreiranje editora transformacija namenjenih specifičnim platformama. Tako su, na primer, već razvijene transformacije za generisanje kôda za *Android* aplikacije. JET (*Java Emiter Templates*) predstavlja specijalizovani okvir za generisanje Java kôda. Okvir je integrisan u osnovnu verziju EMF okruženja.

[12] OMG MOF Model to Text Standard, http://www.omg.org/spec/MOFM2T/1.0/

7.2.2. Mogućnosti transformacija razvojnih modela korisničkih interfejsa

U ovom odeljku dajemo sažet pregled predloženog sistema transformacija razvojnih modela korisničkih interfejsa. Osnovu za kreiranje razvojnih modela predstavljaju UML proširenja za modelovanje kontekstno-osetljive interakcije čoveka i računara i kontekstno-osetljivih korisničkih interfejsa. Transformacije modela korisničkih interfejsa se izvršavaju u proširenim faza dizajna i implementacije. Na slici 7.2 svaka transformacija je prikazana kao struktuirana aktivnost koja obuhvata odgovarajuće modele i UML profile kojima su ovi modeli opisani.



Slika 7.2. *Aktivnosti predloženog sistema transformacija modela korisničkih interfejsa.*

Sistem transformacija između PIM i PSM u kontekstu predloženih proširenja za modelovanje kontekstno-osetljivih korisničkih interfejsa predstavlja skup sukscesivnih transformacija, gde svaka transformacija uzima u obzir specifičnosti odgovarajućih modela korisničkih interfejsa. Na primer, transformacija između modela zadataka i modela apstraktnog korisničkog korisničkog u obzir uzima

odgovarajuće metamodele za opise ovih modela, tj UML profil zadataka i UML profil apstraktnog korisničkog interfejsa. Transformacije se mogu parametrizovati uvođenjem kontekstualnih faktora koji su predstavljeni modelima čoveka, okruženja i platforme. Ovi modeli su opisani odgovarajućim UML profilima konteksta interakcije. Na slici su ovi modeli objedinjeni kao ulazni model konteksta. Na sličan način se dizajniraju transformacije između ostalih modela korisničkih interfejsa dok se ne dođe do modela programskog kôda. Opisani postupak se odnosi na transformacije modela korisničkih interfejsa počevši od viših ka nižim nivoima apstrakcije (*eng. model reification*). Sa druge strane, jedna od prednosti našeg pristupa je mogućnost generisanja modela na višim nivoima apstrakcije iz modela nižih nivoa apstrakcije (*eng. model abstraction*). Rešenje se svodi na dizajn parova transformacija, po jedna za svaki smer, između odgovarajućih modela. Na ovaj je obezbeđen formalan mehanizam adaptacije korisničkog interfejsa u skladu sa promenama konteksta interakcije koji je opisan odgovarajućim modelima. Pored toga, sistem transformacija je fleksibilan u pogledu polaznog modela. Drugim rečima, u dizajnu korisničkog interfejsa kao početni model može se uzeti model zadataka, model apstraktnog interfejsa ili model konkretnog interfejsa.

Transformacije u opsegu od modela zadataka do modela konačnog korisničkog interfejsa spadaju u transformacije između apstraktnih sintaksi, tj. M2M tipa. Transformacija modela konačnog interfejsa u model programskog kôda predstavlja transformaciju između apstraktne i konkretne sintakse, tj. M2T transformaciju. M2M transformacije su realizovane korišćenjem ATL tehnologije. Na primeru M2T transformacija ilustrovaćemo primenu ATL mehanizama za generisanje programskog kôda iz modela. U odeljcima koji slede dajemo detaljniji opis transformacija modela kontekstno-osetljivih korisničkih interfejsa po aktivnostima predloženim na slici 7.2.

7.3. *Specifikacija sistema transformacija*

U ovom odeljku detaljnije opisujemo sistem transformacija između modela kontekstno-osetljivih korisničkih interfejsa u koje spadaju:

- Transformacija modela zadataka u model apstraktnog korisničkog interfejsa,
- Transformacija modela apstraktnog u model konkretnog korisničkog interfejsa,
- Transformacija modela konkretnog u model interfejsa prilagođen tehnologiji implementacije,
- Transformacija modela interfejsa prilagođenog tehnologiji u programski model interfejsa.

Faktori konteksta interakcije su formalno uključeni u sistem transformacija preko odgovarajućih modela i profila kojima su oni opisani. Sa obzirom na veliki broj faktora konteksta, domen korišćenja softverskog sistema će odrediti relevantne faktore koje treba uzeti u obzir prilikom dizajna korisničkog interfejsa. Predloženi profili za modelovanje kontekstno-osetljive interakcije daju formalizovan i sistematičan pregled ovih faktora i na taj način omogućuju i olakšavaju njihovu integraciju u proces razvoja korisničkih interfejsa.

Kako bi sistem transformacija bio što je moguće više fleksibilan i optimizovan sa stanovišta performansi i ponovne upotrebljivosti koristili smo ugrađene mehanizme ATL jezika – biblioteke

transformacija, superimpoziciju, tehnološki prostor i upitne module. Radi boljeg razumevanja sistema transformacija najpre ćemo ukratko opisati pomenute zajedničke mehanizme razvijene u kontekstu transformacija modela korisničkih interfejsa napisanih na UML jeziku.

Sistem transformacija predviđa korišćenje parametarskih modela eksternih za kontekstno-osetljive korisničke interfejse kao što je, na primer, model koji služi za prevođenje naziva elemenata transformisanih modela. Ovde se može kreirati xml dokument proizvoljne sintakse i strukture. Kako bi dokument bio upotrebljiv u kontekstu ATL transformacije potrebno ga je transformisati u *ecore* model. Ovaj mehanizam je poznat pod nazivom *XML injection*. Mehanizam zahteva postojanje metamodela (*XML.ecore*) koji opisuje XML dokument na visokom nivou. Na slici 7.3. je dat ATL ant skript koji realizuje *XML injection*. Najpe se učitavaju XML metamodel i xml dokument, zatim se poziva XML injection operacija, da bi se na kraju generisani model sačuvao u ecore formatu.

```
<atl.loadModel name="XML" path="${metamodelsPath}${xmlMetamodel}"
metamodel="MOF" modelHandler="EMF"/>
<atl.loadModel modelHandler="EMF" name="${task2auiParams}" metamodel="XML"
path="${paramsPath}${task2auiParams}.xml">

<injector name="XML"/> </atl.loadModel>

<atl.saveModel model="${task2auiParams}"
path="${paramsPath}${task2auiParams}.ecore"/>
```

Slika 7.3. *ATL Ant script koji učitava xml dokument u ecore model (XMLInjection).*

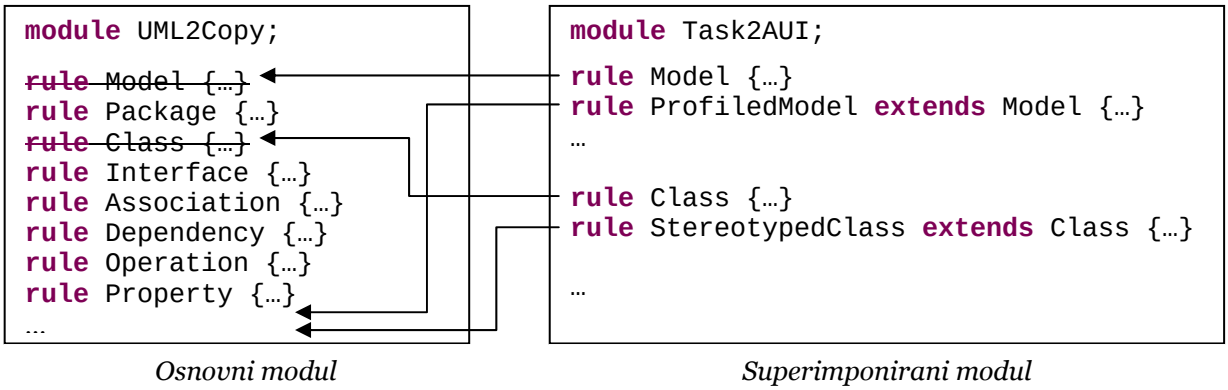
Na ovaj način se podacima (parametrima) generisanog modela može pristupati u transformaciji. Za tu namenu razvijena je ATL biblioteka (*eng. library*) koja sadrži skup ATL helper funkcija za čitanje parametara. Ova biblioteka se može koristiti u bilo kojoj transformaciji koja koristi xml parametarske modele. Na slici 7.4 dat je primer helper funkcije koja čita vrednost parametra za dati model i naziv parametra koji se u njemu nalazi.

```
helper def : getParameter(model : String, name : String) : String =
      XML!Element.allInstancesFrom(model)->
      select(e | e.name = 'param')->
      select(e | e.getAttrVal('name')=name )->first().getAttrVal('value');
```

Slika 7.4. *ATL helper za čitanje vrednosti XML elementa na osnovu naziva modela i elementa.*

ATL omogućava kombinovanje većeg broja transformacija i njihovo izvršavanje u datom trenutku. Ovaj mehanizam je poznat pod nazivom superimpozicija (*eng. superimposition*). Suština mehanizma se ogleda u tome da se jedna transformacija proglasi osnovnom transformacijom nad kojom se superponiraju drugi moduli. Krajnji rezultat je transformacioni modul koji sadrži uniju transformacionih pravila i helper funkcija svih modula, s tim što transformacioni modul može redefinisati pravila i funkcije modula nad kojima je superponiran. Na slici 7.5 je opisan scenario korišćenja superimpozicije u transformacijama UML modela kontekstno-osetljivih korisničkih interfejsa. *UML2Copy* modul sadrži transformaciona pravila koja vrše kopiranje elemenata UML modela iz ulaznog u izlazni model. Transformacija sadrži oko 200 pravila i pokriva relevantne koncepte UML metamodela. Svaka transformacija koja menja izvorni model će promeniti neke od elemenata modela, dok će nepromenjeni elementi biti kopirani u rezultujući model. Modul *Task2AUI* vrši transformaciju modela zadataka u model apstraktnog korisničkog interfejsa. Pravila transformišu

stereotipove zadataka u odgovarajuće stereotipove apstraktnog korisničkog interfejsa. Pre toga, odgovarajući profil mora biti primenjen na nivou modela. Da bi se ovo postiglo, modul *Task2AUI* je superimponiran nad modulom *UML2Copy*. On redefiniše pravila za elemente modela koji se menjaju. Redefinisanje je podržano mehanizmom nasleđivanja pravila (relacija *extends*). U konkretnom slučaju, između ostalih, biće redefinisano pravilo koje kopira UML modele (tako da omogućava proveru i primenu profila), kao i pravilo koje kopira UML klasu. Superimpozicija je relacija u vreme izvršavanja transformacija, rezultujući modul ne postoji kao takav, već se moduli istovremeno učitavaju i superimponiraju po redosledu navođenja. Svojstvo superimpozicije se može podesiti u konfiguracionom fajlu transformacije ili ATL ant skriptu za programsko pokretanje transformacije. Na ovaj način, superimpozicija omogućava ponovnu upotrebljivost i povećava preglednost transformacija.



Slika 7.5. *ATL mehanizam superimpozicije.*

Na sliici 7.6 je dat primer transformacionog pravila koje vrši kopiranje UML Model elementa iz ulaznog u izlazni model.

```
rule Model {
    from s : UML2!"uml::Model" in IN (s.oclIsTypeOf(UML2!"uml::Model"))
    to t : UML2!"uml::Model" (
        __xmiID__ <- s.__xmiID__,
        name <- s.name,
        visibility <- s.visibility,
        viewpoint <- s.viewpoint,
        eAnnotations <- s.eAnnotations,
        ownedComment <- s.ownedComment,
        clientDependency <- s.clientDependency,
        nameExpression <- s.nameExpression,
        elementImport <- s.elementImport,
        packageImport <- s.packageImport,
        ownedRule <- s.ownedRule,
        templateParameter <- s.templateParameter,
        templateBinding <- s.templateBinding,
        ownedTemplateSignature <- s.ownedTemplateSignature,
        packageMerge <- s.packageMerge,
        packagedElement <- s.packagedElement )
}
```

Slika 7.6. *Pravilo koje kopira UML koncept Model iz ulaznog u izlazni model transformacije.*

Upitni ATL moduli (*eng. ATL query*) omogućavaju vršenje različitih proračuna nad ulaznim modelima i nemaju izlazni model. Ovaj mehanizam je korišćen za kreiranje generatora kôda koji će biti opisan u odeljku 7.3.4.

7.3.1. Transformacija modela zadataka u model apstraktnog interfejsa

Ovaj korak sistema povezanih transformacija kao ulaz prihvata model zadataka i generiše model apstraktnog korisničkog interfejsa. Pretpostavka je da inženjer prethodno kreira model zadataka sa primenom odgovarajućeg profila i stereotipova. Ulazni model je i parametarski model koji je kreiran *XML injection* mehanizmom. U našem slučaju, ovaj model parametrizuje prevođenje naziva između elemenata ulaznog i izlaznog modela. Osnov za dizajn transformacija predstavljaju proširenja za modelovanje zadataka i apstraktnog korisničkog interfejsa koja su opisana u odeljku 5.2. Na slici 7.7 je dat primer pravila koje generiše stereotip interaktivne komponente apstraktnog korisničkog interfejsa iz stereotipa interaktivnog zadatka.

```
rule InteractiveTask2InteractiveComponent extends StereotypedClass {
from s : UML2!"uml::Class" in IN (
            s.isInputTask or s.isOutputTask or s.isInputTaskChild)
using {
functionalComp : UML2!"uml::Class" = thisModule.FunctionalComponent(s);}
to t : UML2!"uml::Class"(
name <- thisModule.getTaskParameter('InteractiveTask') + s.name,
ownedAttribute <- s.ownedAttribute->union(Sequence{optional,interruptible
,repeatable,repetitionCount,executionTime,functionalComponent})),
functionalComponent : UML2!"uml::Property"(
name <- 'dst_functionalComponent',
type <- functionalComp,
isUnique <- true,
isOrdered <- true,
lowerValue <- funLower,
upperValue <- funUpper,
association <- funAssoc,
aggregation <- #composite),
funUpper : UML2!"uml::LiteralUnlimitedNatural"(
value <- 0-1),
funLower : UML2!"uml::LiteralInteger"(
value <- 0),
funAssoc : UML2!"uml::Association"(
name <- 'ass_' + t.name + '_' + functionalComp.name,
memberEnd <- Sequence{functionalComponent,functionalComponentAss}),
functionalComponentAss : UML2!"uml::Property"(
name<-'src_'+(if(s.isInputTask)then 'inputComp' else 'outputComp' endif),
type<-UML2!Class.allInstancesFrom('OUT')->select(c|c.name=t.name)-
>first()
visibility <- #public,
isUnique <- true,
isOrdered <- true,
lowerValue <- funAssLower,
upperValue <- funAssUpper,
owningAssociation <- funAssoc),
funAssUpper : UML2!"uml::LiteralUnlimitedNatural"(value <- 1),
funAssLower : UML2!"uml::LiteralInteger"(value <- 1)
}
```

Slika 7.7. *Pojednostavljen prikaz pravila koje preslikava stereotip interaktivnog zadatka u interaktivnu komponentu apstraktnog interfejsa. Za konkretne tipove interaktivnih zadataka, nasleđivanjem pravila će se generisati odgovarajući tip komponente apstraktnog interfejsa (Slika 7.9).*

Uslov za okidanje pravila je da klasa bude označena kao stereotip koji predstavlja jedan od interaktivnih tipova zadataka. Pravilo nasleđuje pravilo *StereotypedClass* koje na osnovu pridodanih vrednosti stereotipa *Task* generiše odgovarajuće atribute (atribut *ownedAttribute*) klase apstraktnog interfejsa. Za svaku instancu interaktivnog tipa zadatka biće generisan par komponenti apstraktnog korisničkog interfejsa – interaktivna komponenta i njoj pridružena funkcionalna komponenta. Interaktivna komponenta će biti generisana u pravilima izvedenim iz tekućeg pravila (Slika 7.9), dok se funkcionalna komponenta kreira pozivanjem *lazy* pravila (promenljiva *functionalComp* klauzule *using*). Interaktivna i funkcionalna komponenta su povezane instancom asocijacije koja je generisana u tekućem pravilu.

Na slici 7.8 prikazano je *lazy* pravilo koje za dati element ulaznog modela (interaktivni zadatak) kreira stereotip *FunctionalComponent* klase zajedno sa stereotipom operacije apstraktnog interfejsa. Primena odgovarajućih stereotipova se vrši u imperativnom delu transformacije (*do* blok).

```
lazy rule FunctionalComponent{
from s : UML2!"uml::Class"
to t: UML2!"uml::Class"(
name <- s.name + thisModule.getTaskParameter('FunctionalComponent')),
uiOperation : UML2!"uml::Operation"(
name <- t.name + thisModule.getTaskParameter('UIOperation'),
class <- t,
visibility <- #public,
concurrency <- #sequential)
do{
t.applyStereotype('FunctionalComponent'.stereotype());
uiOperation.applyStereotype('UIOperation'.stereotype());
  }
}
```

Slika 7.8. *Lazy pravilo koje kreira stereotip FunctionalComponent sa stereotipom UIOperation operacije apstraktnog korisničkog interfejsa.*

Slika 7.9 opisuje pravilo koje na osnovu stereotipa izlaznog zadatka generiše stereotip izlazne komponente apstraktnog korisničkog interfejsa. Pravilo se okida pri nailasku na zadatak označen kao strereotip *OutputTask* u ulaznom modelu.

```
rule OutputTask2OutputComponent extends InteractiveTask2InteractiveComponent{
from s : UML2!"uml::Class" in IN (
s.isOutputTask)
to t : UML2!"uml::Class" (
name <- thisModule.getTaskParameter('OutputTask') + s.name)
do {
t.owner.packagedElement<-t.owner.packagedElement->
union(Sequence{funAssoc,functionalComp});
t.applyStereotype('OutputComponent'.stereotype());
   }
}
```

Slika 7.9. *Pravilo koje transformiše stereotip OutputTask modela zadataka u stereotip OutputComponent apstraktnog interfejsa.*

Slično preslikavanjima između klasa vrše se i preslikavanja između proširenja asocijacija modela zadataka i modela apstraktnog interfejsa. Ostali elementi modela će biti kopirani u izlazni model. Na

ovaj način je uspostavljena *refinement* relacija između modela korisničkih interfejsa na različitim nivoima apstrakcije. Kao podrška *traceability* svojstvu transformacija, tj. mogućnošću dolaženja do elemenata izvornog modela na osnovu elemenata izlaznog modela, prilikom preslikavanja između UML proširenja za modelovanje korisničkih interfejsa kreiramo anotacije pridružene izlaznim elementima koje nose metapodatke o elementima iz kojih su generisani. Na slici 7.10 prikazano je *lazy* pravilo koje se eksplicitno poziva prilikom preslikavanja između proširenja klasa. Generisanom elementu izlaznog modela se pridružuje *EAnnotation* instanca koja sadrži metapodatke o izvornom elementu (XMI identifikator, ime klase i ime primenjenog stereotipa).

```
lazy rule Task2AUIClassAnnotation{
from s : UML2!"uml::Class"
to t : UML2!"ecore::EAnnotation"(
eModelElement <- s,
source <- 'Task gen element',
details <- Sequence{sourceID,sourceName,sourceStereotype}),
sourceID : UML2!"ecore::EStringToStringMapEntry"(
key <- 'source class id',
value <- s.__xmiID__),
sourceName : UML2!"ecore::EStringToStringMapEntry"(
key <- 'source class name',
value <- s.name),
sourceStereotype : UML2!"ecore::EStringToStringMapEntry"(
key <- 'source stereotype',
value <- s.getAppliedStereotypes()->first().name)
}
```

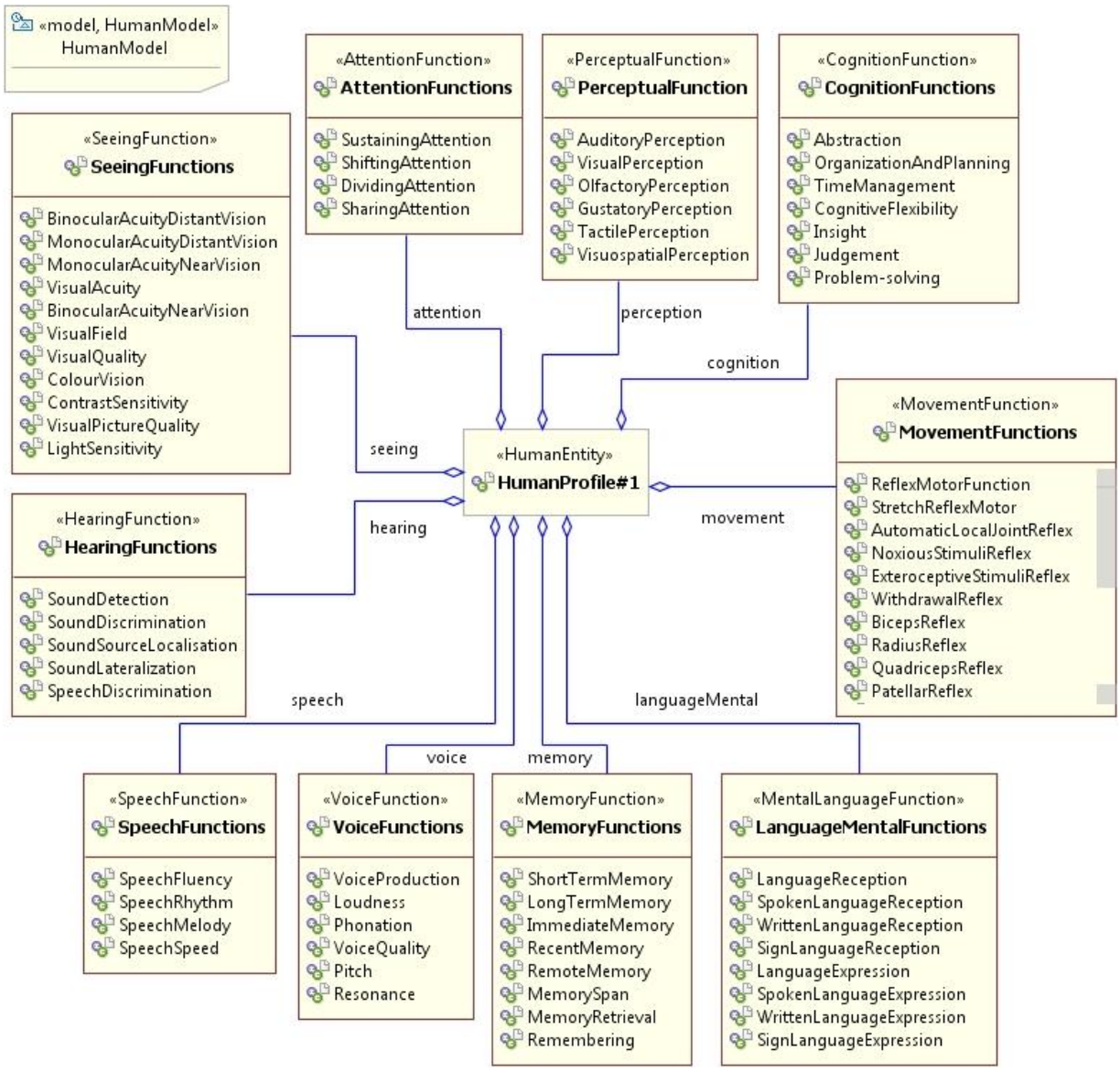
Slika 7.10. *Lazy pravilo koje kreira anotaciju sa metapodacima o ulaznom elementu modela zadataka. Anotacija je pridružena generisanom elementu modela apstraktnog interfejsa.*

7.3.2. Transformacija modela apstraktnog u model konkretnog interfejsa

Sledeća aktivnost u lancu transformacija je preslikavanje apstraktnog u konkretni korisnički interfejs. Na primeru ove aktivnosti pokazaćemo uvođenje i korišćenje modela konteksta interakcije u razvoju modela korisničkih interfejsa na različitim nivoima apstrakcije. Iz modela konteksta interakcije izdvojili smo model čoveka. Parametri modela čoveka će odrediti konkretne profile korisničkih interfejsa u koje će se preslikavati elementi modela apstraktnog korisničkog interfejsa. Drugim rečima, na osnovu analize telesnih funkcija konkretnog korisnika biće generisan model korisničkog interfejsa prilagođen načinu komunikacije optimalnom za datog korisnika.

Na slici 7.11 je dat primer modela čoveka koji opisuje njegove telesne funkcije. Funkcije su opisane kao stereotipovi ICF klasifikacije za opis telesnih funkcija. Svaki od entiteta na slici objedinjuje ugnježdene (*eng. nested*) elemente koji su modelovani kao stereotipovi funkcija određene kategorije. Čovek je opisan grupama funkcijama koje su relevantne za interakciju sa računarom kao što su funkcije vida, sluha, glasa, govora i dr. Telesne funkcije čoveka su dodatno opisane pridodanim vrednostima *impairment* i *preference* koje predstavljaju generičke kvalifikatore stepena ograničenja i individualnih sklonosti ka angažovanju telesnih funkcija, respektivno. Pridodane vrednosti su opisnih tipova

izvedenih iz UML *Enumeration* tipa (*PreferenceExtent* i *ImpairmentExtent*). Vrednosti nabrojivog tipa za opisivanje ograničenja u funkcionisanju su preuzete direktno iz ICF klasifikacije. Kako bi izrazili individualne sklonosti ka korišćenju određenih grupa telesnih funkcija čoveka, predlažemo nabrojivi tip sa tri vrednosti - nizak (*LowPreference*), umeren (*MediumPreference*) i visok (*HighPreference*) stepen sklonosti. Pretpostavka je da su funkcije modela čoveka kvantifikovane u pogledu ograničenja i individualnih sklonosti. Za tu namenu se mogu koristiti postojeći testovi za merenje individualnih sklonosti ka korišćenju i ograničenja telesnih funkcija čoveka.



Slika 7.11. *Model ljudskih funkcija. Grupe funkcija su razdvojene po entitetima koji objedinjavaju odgovarajuće podfunkcije.*

Za analizu funkcija čoveka sa stanovišta ograničenja i sklonosti ka angažovanju razvijene su odgovarajuće ATL biblioteke. Ove biblioteke rade sa modelima čoveka i koriste se u transformacijama modela korisničkih interfejsa. Na slici 7.12 je dat primer metode koja vrši proračun stepena ograničenja u funkcionisanju funkcija vida čoveka.

```
helper context UML2!"uml::Model" def : overallSeeingImpairment() : OclAny =
let functions : UML2!"uml::Class"=UML2!"uml::Class".allInstances()->
select(c |not c.getAppliedStereotypes()->
any(e|e.name='SeeingFunction').oclIsUndefined() or not
c.getAppliedStereotypes()->any(e|e.name='VisualAcuity').oclIsUndefined()
or not c.getAppliedStereotypes()->
any(e|e.name='VisualField').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='LightSensitivity').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='BinocularAcuityDistantVision').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='MonocularAcuityDistantVision').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='MonocularAcuityNearVision').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='BinocularAcuityNearVision').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='VisualQuality').oclIsUndefined() or not
c.getAppliedStereotypes()->any(e|e.name='ColourVision').oclIsUndefined()
or not c.getAppliedStereotypes()->
any(e|e.name='ContrastSensitivity').oclIsUndefined() or not
c.getAppliedStereotypes()->
any(e|e.name='VisualPictureQuality').oclIsUndefined()) in
if functions.isEmpty() then OclUndefined
else functions->iterate(im ; overall : Integer = 0 | overall +
im.bodyFunctionImpairmentSelf().integerImpairValue)/functions.size()
endif;
```

Slika 7.12. *Helper metoda koja vrši proračun stepena ograničenja funkcija vida modela čoveka.*

Metoda čita vrednosti ograničenja stereotipova funkcija čula vida i računa srednju vrednost. Na slici 7.13a je prikazana metoda transformacionog modula koja poziva prethodno opisanu metodu biblioteke. Metoda transformacionog modula na slici 7.13b utvrđuje da li korisniku odgovara vizuelni način komunikacije na osnovu proračuna ograničenja i sklonosti ka angažovanju funkcija čula vida.

```
helper  def : humanSeeingImpairment : Integer =
UML2!"uml::Model".allInstancesFrom('ContextModel')->select(m |
m.isStereotypeApplied('HumanModel'.stereotype()))->
first().overallSeeingImpairment();
```

(a)

```
helper def : isVisualHumanType : Boolean =
thisModule.humanSeeingImpairment.round().stringImpairValue = 'NoImpairment'
and thisModule.humanSeeingPreference.round()>=
'MediumPreference'.integerPrefValue;
```

(b)

Slika 7.13. *Helper metode u okviru transformacije modela apstraktnog u konkretni interfejs koje na osnovu proračuna svojstava funkcija čula vida čoveka govore da li se radi o vizuelnom tipu.*

Na slici 7.14 dat je pojednostavljen prikaz hijerarhije transformacionih pravila koja će se izvršavati za vizuelni tip korisnika. Ovde je iskorišćen mehanizam nasleđivanja ATL pravila. Osnovno pravilo sadrži uslov koji proverava da li se radi o vizuelnom tipu korisnika. Izvedeno pravilo se izvršava pri nailasku na stereotip ulazne komponente apstraktnog interfejsa i kreira klasu sa stereotipom interaktivne vizuelne komponente koja je označena kao ulazna (pridodana vrednost *type* dobija vrednost *Input*).

```
rule AUIClass2VisualClass extends AUIStereotypedClass {
      from s : UML2!"uml::Class" in IN (
         thisModule.isVisualHumanType)
      to t : UML2!"uml::Class"()
}
rule InputComponent2InputInteractiveComponent extends AUIClass2VisualClass
{
      from s : UML2!"uml::Class" in IN (
            s.isInputComponent)
      to t : UML2!"uml::Class"(
            name <- thisModule.getAUIParameter('InputComponent') + s.name)
      do{
      t.applyStereotype('InteractiveVisualComponent'.stereotype());
      t.setValue('InteractiveVisualComponent'.stereotype(),'type','Input');
      }
}
```

Slika 7.14. *Transformaciono pravilo koje preslikava klasu apstraktnog korisničkog interfejsa u klasu profila vizuelnog korisničkog interfejsa. Pravilo konsultuje podmodel čoveka modela konteksta i izvršava se za vizuelni tip. Na slici je dato pravilo koje vrši proveru tipa čoveka i pravilo izvedeno iz njega koje vrši preslikavanje između odgovarajućih stereotipova interaktivnih komponenti.*

Pored biblioteka za analizu modela čoveka , razvijene su i ATL biblioteke za analizu modela okruženja, računarskih uređaja i koncepata višenačinske interakcije. Model konteksta interakcije je definisan nezavisno od modela korisničkog interfejsa i koristi se u razvoju korisničkih interfejsa opisanih modelima na različitim nivoima apstrakcije. Na ovaj način se faktori konteksta interakcije mogu formalno opisati i integrisati u automatizovani proces razvoja korisničkih interfejsa.

7.3.3. Transformacija modela konkretnog u model konačnog interfejsa

U ovom koraku se model korisničkog interfejsa prilagođen određenim načinima komunikacije prevodi u model prilagođen specifičnoj tehnologiji implementacije. Na ovaj način će model korisničkog interfejsa biti pogodan za generisanje teksta programa, dok će sa druge strane biti očuvana semantika kontekstno-osetljive interakcije čoveka i računara. Ovu vrstu transformacije ćemo opisati na primeru preslikavanja u model korisničkog interfejsa prilagođen *Java Swing* tehnologiji.

Model korisničkog interfejsa koji opisuje koncepte specifične tehnologije i relacije između njih se prosleđuje kao zaseban ulazni model. Ovi modeli se mogu dobiti mehanizmima opisanim u poglavlju 5.2.4. Transformacija vrši preslikavanje stereotipova komponenti ulaznog modela korisničkog interfejsa u komponente koje nasleđuju ili implementiraju odgovarajuće koncepte date tehnologije.

Koncepti konteksta interakcije se mogu koristiti u ovoj fazi sa obzirom da model konteksta predstavlja jedan od ulaznih modela. Model kreiran na ovaj način je prilagođen potrebama generisanja teksta programa. Slično prethodnim fazama transformacija, i ovde je korišćen princip pridruživanja anotacija elementima izlaznog modela koje sadrže metapodatke o elementima iz kojih su generisani (slika 7.15).

```
rule VisualUIStereotypedClass extends Class {
from s : UML2!"uml::Class" in IN (
'VisualUI'.profile().ownedStereotype.includes(s.getAppliedStereotypes()->
first()))
to t : UML2!"uml::Class" (
   name <- s.getAppliedStereotypes()->first().name + '_' + s.name,
   eAnnotations <- s.eAnnotations ->
   union(Set{thisModule.Visual2FUIClassAnnotation(s)}))
}

lazy rule Visual2FUIClassAnnotation{
   from s : UML2!"uml::Class"
   to t : UML2!"ecore::EAnnotation"(
      eModelElement <- s,
      source <- 'Visual gen element',
      details <- Sequence{sourceID,sourceName,sourceStereotype}),
   sourceID : UML2!"ecore::EStringToStringMapEntry"(
      key <- 'source class id',
      value <- s.__xmiID__),
   sourceName : UML2!"ecore::EStringToStringMapEntry"(
      key <- 'source class name',
      value <- s.name),
   sourceStereotype : UML2!"ecore::EStringToStringMapEntry"(
      key <- 'source stereotype',
      value <- s.getAppliedStereotypes()->first().name)
}
```

Slika 7.15. *Primer korišćenja lazy pravila koje kreira anotaciju sa metapodacima o ulaznom elementu sa stereotipom profila vizuelnog interfejsa. Anotacija je pridružena generisanom elementu.*

Na slici 7.16 je dat primer transformacionog pravila koje preslikava komponentu sa stereotipom *Button* profila vizuelnog inerfejsa u klasu koja nasleđuje *JButton Swing* komponentu i implementira *ActionListener* osluškivač događaja. Generisanoj klasi je pridružena operacija koja je propisana interfejsom osluškivača (*actionPerformed*). UML standard daje mogućnost definisanja implementaciono specifičnog ponašanja operacije preko *OpaqueBehavior* koncepta. U konkretnom slučaju, konsultovana je vrednost atributa *repeatable* koji vodi poreklo od odgovarajuće pridodane vrednosti elementa modela zadataka. U zavisnosti od vrednosti atributa, generisan je kostur tela metode osluškivača događaja.

```
helper def:JButton : UML2!"uml::Class" = 'javax::swing::JButton'.class();
helper def:ActionListener:UML2!"uml::Interface"=
'java::awt::event::ActionListener'.interface();
helper def:ActionEvent:UML2!"uml::Class"=
'java::awt::event::ActionEvent'.class();
```

(a)

```
rule Button2SwingButton extends VisualUIStereotypedClass {
      from s : UML2!"uml::Class" in IN (s.isButtonVisualComponent)
      using { repeatable : UML2!"uml::Property" = s.attribute->
      select(p|p.name = 'repeatable')->first();
      }
      to t : UML2!"uml::Class" (
        name <- s.getAppliedStereotypes()->first().name + '_' + s.name,
        ownedOperation <- s.ownedOperation ->union(Sequence{apOperation}),
        ownedBehavior <- s.ownedBehavior -> union(Sequence{apBehavior})),
      generalization : UML2!"uml::Generalization"(
        general <- thisModule.JButton,
        specific <- s),
      interface : UML2!"uml::InterfaceRealization"(
        contract <- thisModule.ActionListener,
        implementingClassifier <- s),
      apOperation : UML2!"uml::Operation"(
        name <- 'actionPerformed',
        class <- s,
        visibility <- #public,
        isStatic <- false,
        isAbstract <- false,
        ownedParameter <- Sequence{apOpPar},
        concurrency <- #concurrent,
        type <- 'OclVoid'.primitiveType()),
      apOpPar : UML2!"uml::Parameter"(
        name <- 'event',
        type <- thisModule.ActionEvent,
        effect <- #update,
        direction <- #"in"),
      apBehavior : UML2!"uml::OpaqueBehavior"(
        specification <- apOperation,
        name <- apOperation.name + 'Behavior',
        language <- 'language'.value(),
        body <- if (repeatable.default = 'true') then
        '//TO DO'.asynchronousBody() else '' endif )
}
```

(b)

```
helper context String def : asynchronousBody() : String =
'new Thread() {\n public void run() {\n' + self + '\n  }\n}.start();';
```

(c)

Slika 7.16. *Helper metode koje referenciraju Java Swing UML tipove po imenu (a). Transformaciono pravilo koje preslikava Button stereotip profila vizuelnog interfejsa u komponentu dugme specifičnu za Java Swing platformu (b). Primer metode koja generiše sadržaj tela operacije u okviru UML koncepta OpaqueBehavior (c).*

Kao podršku generisanju teksta programa korisničkog interfejsa, razvijena je transformacija koja generiše pristupne (*eng. accessor*) metode sa kosturima tela za javne atribute modela (jednostavnih tipova i tipova kolekcija). Pored toga, posebna transformacija vrši preslikavanje UML OCL tipova podataka u Java tipove. Pomenute transformacije se mogu primeniti nad modelima prilagođenim Java tehnologiji implementacije.

7.3.4. Transformacija modela konačnog interfejsa u programski model interfejsa

Ovde ćemo opisati korišćenje ATL upitnih modula na primeru transformacije modela interfejsa prilagođenog tehnologiji implementacije u tekst programa. Ovakav način realizacije mehanizma generisanja kôda olakšava integraciju, upotrebu i održavanje predloženog sistema transformacija sa obzirom na to da su svi nivoi transformacija (tj. M2M i M2T) realizovani u istoj tehnologiji. Upitni moduli vrše različite tipove proračuna nad ulaznim modelom i nemaju izlazni model.

U našem slučaju, ulazni model je model interfejsa prilagođen tehnologiji implementacije. Modul vrši iteraciju kroz elemente ulaznog modela i generiše Java komponente sa svim relevantnim svojstvima i relacijama iz odgovorajućih tipova UML klasifikatora. Na slici 7.17 su prikazani segmenti ATL upita koji se odnose na iteraciju ulaznog modela (a), generisanje izvornih programskih datoteka (b), i generisanje odgovarajućih Java primitiva. Rezultat obrade svakog UML klasifikatora se ispisuje u tekstualnom formatu (*writeTo()*), gde je putanja zadata preko parametarske funkcije (*pathName()*).

```
query UML2Java = UML2!"uml::Classifier".allInstances()->
iterate(e; acc : Boolean = true | if e.ignore() then acc
else acc and e.modelToText().writeTo(e.pathName()) endif);
```

(a)

```
helper context UML2!"uml::Classifier" def : modelToText() : String =
      self.packageDecl() + '\n' + self.importDecl() + '\n' + self.toCode();
```

(b)

```
helper context UML2!"uml::Class" def : toCode() : String =
self.visibilityStr +
self.abstractStr +
self.finalStr +
'class ' + self.name + self.extendsClause() + self.implementsClause() +
' {\n' +
self.nestedClassifier->iterate(e; acc : String = '' | acc + e.toCode()) +
self.ownedAttribute->iterate(e; acc : String = '' | acc + e.toCode()) +
self.ownedOperation->iterate(e; acc : String = '' | acc + e.toCode()) +
'}\n\n';
```

(c)

Slika 7.17. *ATL upit koji transformiše UML elemente modela prilagođenog Java tehnologiji u odgovarajuće Java komponente (a). Helper metoda koja generiše Java izvornu datoteku (b). Helper metoda koja iz UML klase generiše Java klasu (c).*

7.4. *Rezime poglavlja*

U ovom poglavlju smo opisali mogućnosti automatizacije procesa razvoja kontekstno-osetljivih korisničkih interfejsa. U skladu sa tim je predložen i razvijen integrisani sistem transformacija modela kontekstno-osetljivih korisničkih interfejsa na različitim nivoima apstrakcije. Demonstrirana je upotreba standardnih tehnologija za dizajn i transformacije modela kontekstno-osetljivih korisničkih interfejsa. Na ovaj način je moguće povećati produktivnost razvoja korisničkih interfejsa, a da pri tome bude očuvana semantika interakcije čoveka i računara, kao i mogućnost integracije sa ostatkom softverskog sistema. Pored toga, razvijeni alati za transformacije nisu vezani isključivo za razvoj korisničkih interfejsa, već su primenjivi i u drugim oblastima.

8. Studije slučajeva

U ovom odeljku ćemo demonstrirati upotrebu predloženih proširenja za dizajn modela i transformacija kontekstno-osetljivih korisničkih interfejsa. Primena predloženog pristupa će biti opisana na primerima dva projekta iz različitih domena. U projektu daljinskog nadzora korišćenjem bespilotne letelice razvijen je korisnički interfejs softverskog sistema u domenu sistema kritičnih za misiju (*eng. mission-critical systems*). Ovde ćemo opisati proces projektovanja korisničkog interfejsa softverske instrument table u kojem je korisnik modeliran sa stanovišta fizioloških i kognitivnih funkcija. U projektu vezanom za učenje putem igara bavili smo se razvojem korisničkih interfejsa edukativnih igara. Razvoj korisničkog interfejsa edukativne igre posmatra čoveka sa aspekta osobina vezanih za učenje, kao što su tipovi inteligencije. U nastavku dajemo opis studija slučajeva sa naglaskom na modelima čoveka i razvojnim modelima korisničkih interfejsa.

8.1. *Studija slučaja letačke instrument table BL*

Osnovni cilj projekta daljinskog nadzora bespilotne letelice je koncipiranje, dizajn i implementacija softverskog alata koji omogućava planiranje, izvršavanje i praćenje misija bespilotne letelice. Radi boljeg razumevanja ukratko ćemo opisati opšte principe softverskog sistema, dok se detalji mogu naći u [Jovanović09b]. Na osnovu analize podataka koji se javljaju u operacijama koje vrši bespilotna letelice, izvršena je njihova klasifikacija u sledeće kategorije [Jovanović07]:

- Globalni pozicioni podaci – određuju apsolutnu poziciju letelice i dobijaju se od zasebnog provajdera.
- Podaci vezani za misiju letelice – definišu specifičan kontekst misije bespilotne letelice u zavisnosti od zadatka koji izvršava. U opštem slučaju, ovi podaci će opisivati kontrolne tačke planirane putanje leta letelice, planiranu i stalnu putanju leta, kao i raznovrsne opise objekata koji predstavljaju ciljeve dejstva.
- Parametri stanja letačke platforme – ova kategorija podataka obuhvata letačke podatke (vazdušna brzina, zemaljska brzina, barometarski visinomer i dr.) i parametre stanja pogona (količina goriva, prosečna potrošnja goriva, temperatura glave motora i drugi).

Odgovornosti operatora vezane za upravljanje letelicom i izvršavanje misija letelice se mogu podeliti po sledećim zadacima:

- Primarni zadatak misije u kojem operator, navodeći letelicu planiranom putanjom dejstva, prati kontrolne tačke i izveštava o ciljevima uočenim na poznatim koordinatama.
- Sekundarni zadatak misije u kojem operator upravlja letelicom kroz okruženje.
- Zadatak nadgledanja sistema u kojem operator nadgleda letačke instrumente i prati parametre okruženja i pogona letelice.

U skladu sa analizom tipova podataka letelice i odgovornosti operatora projektovan je softverski sistem sastavljen od sledećih celina [Jovanović10]:

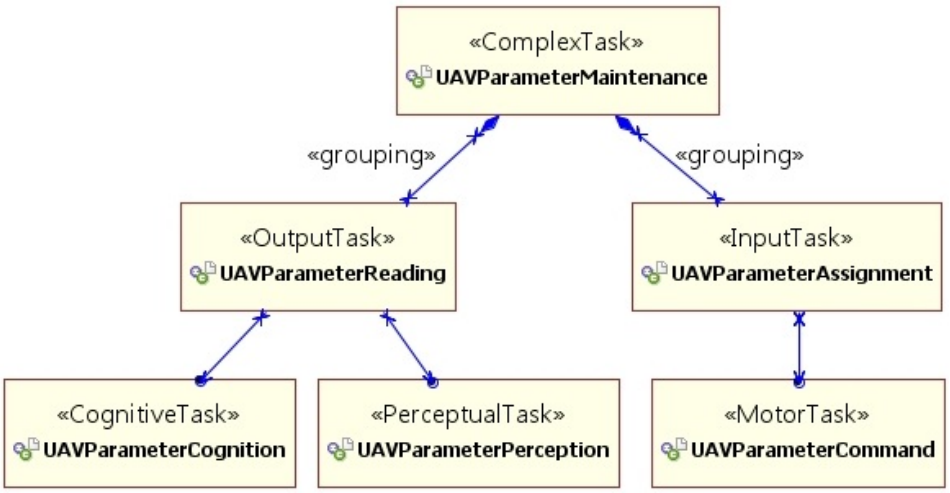
- Modul za komunikaciju sa letačkom platformom – zadužen za komunikaciju za letačkom platformom preko uređaja za prijem telemetrijskih podataka, obradu dobijenih podataka i njihovu distribuciju zainteresovanim modulima, kao i drugim sistemima.
- Modul za prezentaciju stanja letačke platforme – predstavlja softversku instrument tablu za prikaz letačkih podataka i podataka stanja pogona letelice.
- Modul za prikaz podataka misije – digitalna mapa sa vertikalnim pogledom i pogledom profila koja će prikazivati podatke vezane za misiju letelice (letelica sa planiranom i stvarnom putanjom leta, kontrolnim objektima i ciljevima).
- Modul za preuzimanje i prikazivanje slike sa letelice – preuzima, obrađuje i prikazuje audio i video tokove koje emituje sistem za video nadgledanje montiran na telu letelice. Ovi podaci se mogu svrstati u kategoriju podataka vezanih za misiju letelice.

Osnovni deo softverskog okruženja predstavlja modul za prijem telemetrijskih podataka. Nakon povezivanja sa uređajem za prijem telemetrijskih podataka, dobijeni podaci se transformišu u poruke specifičnih formata kako bi bili pogodni za dalju obradu. Moduli za prezentaciju stanja letačke platforme i prikaz podataka misije, preko odgovarajućeg programskog interfejsa, uspostavljaju komunikaciju sa modulom za prijem telemetrijskih podataka, čitaju generisane poruke i prikazuju ih na odgovarajući način. Slanje upravljačkih komandi ka leletici se vrši putem odvojene ručne konzole. Velike količine podataka letelice koje se prezentuju u realnom vremenu izazivaju velika naprezanja ljudskih senzorskih i percepcijskih aparata. U skladu sa tim, određeni delovi interfejsa aplikacije će se projektovati uzimajući u obzir karakteristike čoveka [Jovanović09a].

U opisu primene predloženog pristupa u razvoju korisničkog interfejsa sistema posmatraćemo dizajn softverske instrument table za praćenje letačkih podataka i parametara stanja pogona BL.

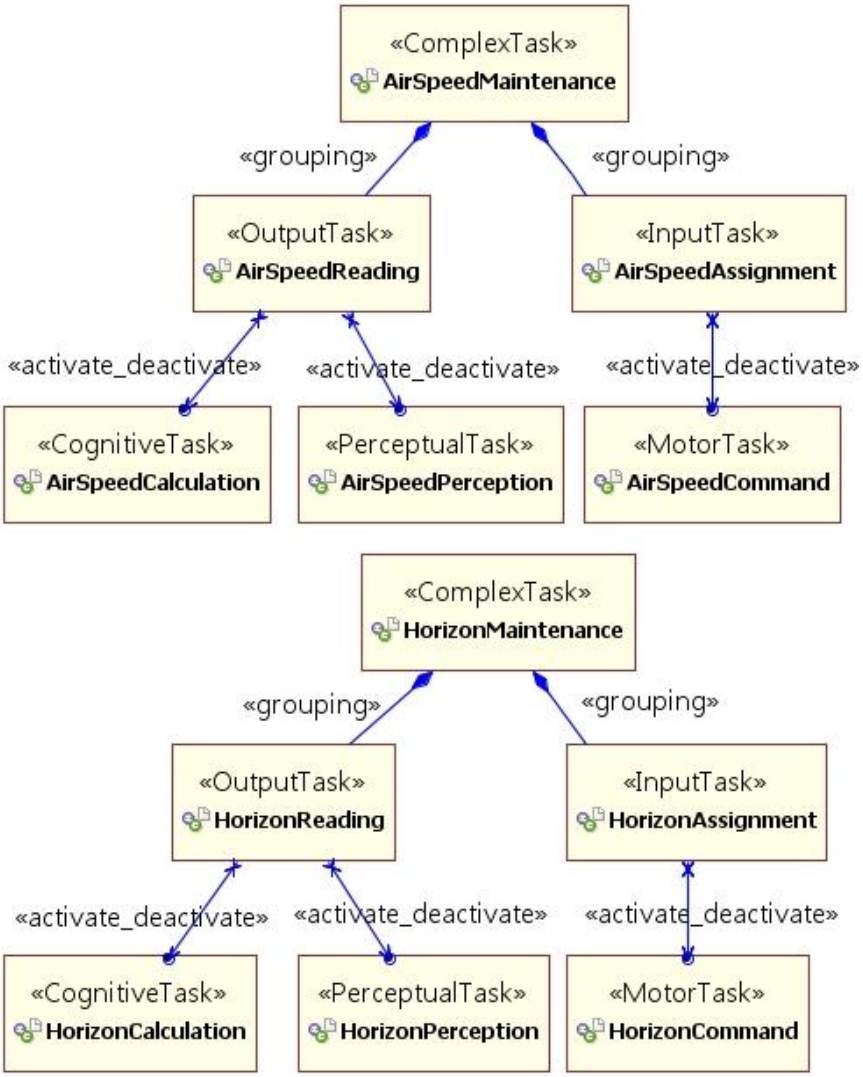
8.1.1. Model zadataka softverske instrument table

Na slici 8.1 prikazana je opšta struktura zadatka vezanog za praćenje i održavanje letačkog parametra.



Slika 8.1. *Generalizovana struktura zadatka vezanog za održavanje letačkog parametra BL od strane operatora.*

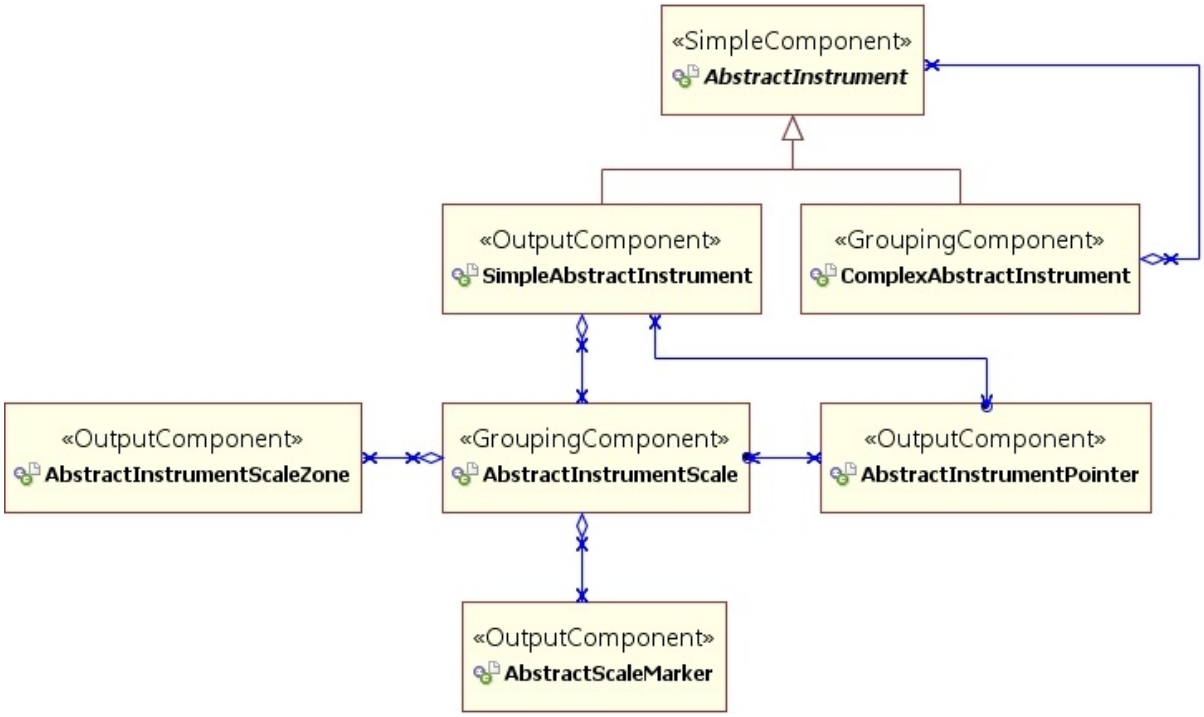
U ranoj fazi razvoja interfejsa instrument table zadaci su analizirani sa stanovišta angažovanja čoveka. Na taj način smo došli do strukture zadatka održavanja parametra BL (*UAVParameterMaintenance*) koji je modelovan kao stereotip složenog zadatka. On objedinjuje stereotip ulaznog zadatka čitanja vrednosti parametra (*UAVParameterReading*) i stereotip izlaznog zadatka zadavanja vrednosti parametra (*UAVParameterAssignment*). Zadatak čitanja vrednosti parametra dalje obuhvata percepcijski zadatak opažanja od strane čoveka (*UAVParameterPerception*) i kognitivni zadatak spoznaje opaženog parametra (*UAVParameterCognition*). Sa druge strane, zadatak zadavanja vrednosti parametra podrazumeva izvršavanje odgovarajuće motoričke radnje čoveka (*UAVParameterCommand*). Na osnovu generalizovane strukture zadatka, izvedeni su zadaci vezani za održavanje specifičnih letačkih parametara. Tako su na slici 8.2 prikazane strukture zadataka vezane za održavanje vazdušne brzine i horizonta. Odgovarajuće klase zadataka nasleđuju koncepte generalizovanog zadatka na slici 8.1. Zadaci vezani za čitanje letačkih parametara će biti realizovani putem specifičnih tipova instrumenata, dok se zadavanje vrednosti parametra vrši putem odvojene upravljačke konzole.



Slika 8.2. *Izvedene strukture zadataka za održavanje specifičnih letačkih parametara BL kao što su vazdušna brzina i horizont.*

8.1.2. Model apstraktnog interfejsa instrument table

Model zadataka vezan za praćenje letačkih parametara i parametara pogona se preslikava u model apstraktnog interfejsa instrument table. Instrument tabla se sastoji od skupa instrumenata koji prezentuju vrednosti letačkih i motoričkih parametara iz modela zadataka. U skladu sa tim smo predvideli apstraktni model instrumenta BL (slika 8.3).

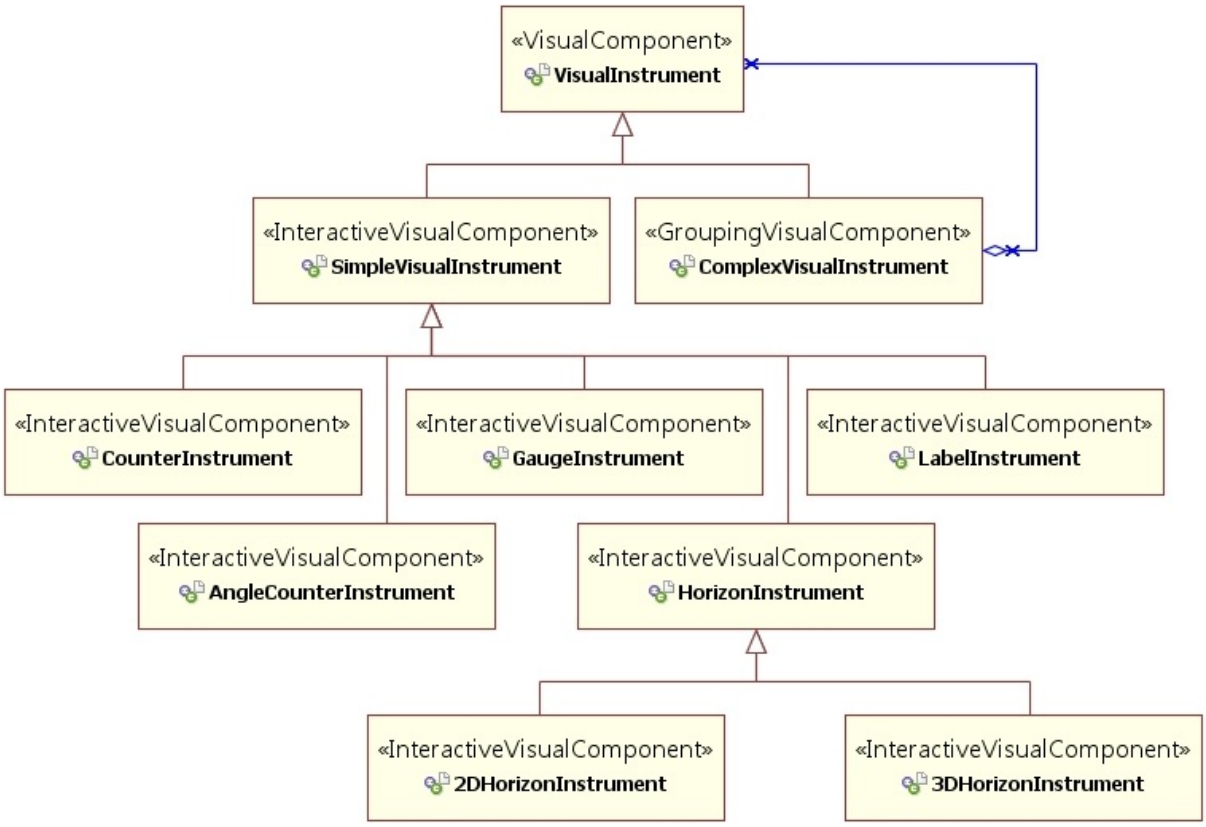


Slika 8.3. *Model apstraktnog instrumenta za prikaz letačkog parametra ili parametra pogona BL.*

Osnovna klasa instrumenta (*AbstractInstrument*) predstavlja stereotip jednostavne komponente apstraktnog interfejsa. Apstraktni instrument može biti jednostavan (*SimpleAbstractInstrument*) ili složen (*ComplexAbstractInstrument*). Jednostavan tip je modelovan kao stereotip izlazne komponente apstraktnog interfejsa, dok je složen tip opisan stereotipom komponente grupisanja. Uvođenje složenog tipa omogućava kreiranje instrumenata proizvoljne složenosti koji se često mogu naći u praksi. Svaki jednostavan instrument se sastoji od skale koja definiše opseg vrednosti parametra koji se prikazuje (*AbstractInstrumentScale*) i pokazivača na skalu (*AbstractInstrumentPointer*). Skala je modelovana kao stereotip komponente grupisanja jer je podeljena na jednu ili više zona (*AbstractInstrumentScaleZone*). Razlog za uvođenje zona jeste isticanje kritičnih opsega vrednosti odgovarajućih parametara. Ovakav pristup omogućava kreiranje generičkih instrument tabli sa proizvoljnim izgledom i razmeštajem instrumenata.

8.1.3. Model konkretnog interfejsa instrument table

Model konkretnog interfejsa se dobija transformacijom modela apstraktne instrument table. Generisani model je prilagođen načinima komunikacije koji odgovaraju potrebama konkretnog operatera. Prilikom transformacije će biti konsultovan model fizioloških i kognitivnih funkcija čoveka. Na slici 8.4 je prikazan konkretni model instrumenta prilagođen vizuelnom načinu komunikacije.



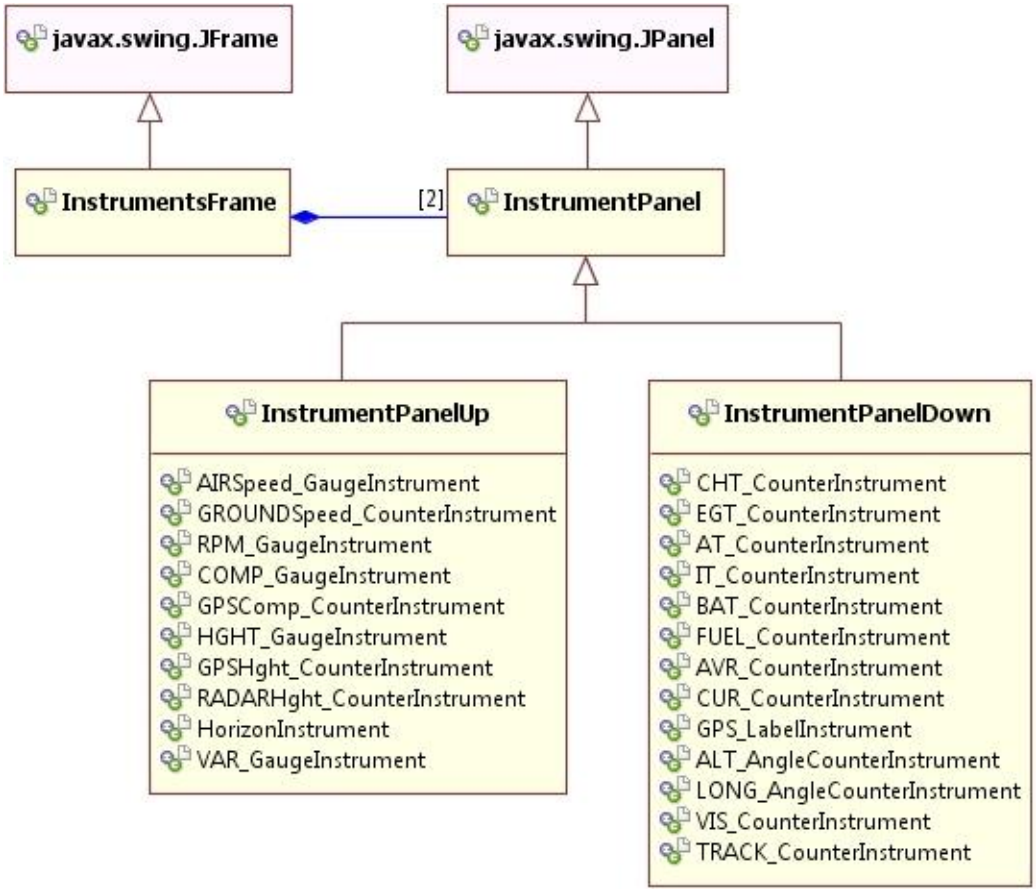
Slika 8.4. *Hijerarhija vizuelnih instrumenata instrument table BL.*

Analogno instrumentu apstraktnog modela, vizuelni instrument može biti jednostavan (*SimpleVisualInstrument*) ili složen (*ComplexVisualInstrument*). Jednostavan tip je modelovan kao stereotip izlazne interaktivne vizuelne komponente (pridodana vrednost *type* setovana na *output*). Složen instrument je označen kao stereotip vizuelne komponente grupisanja. Jednostavni vizuelni instrumenti dalje mogu biti tipa brojačkog instrumenta (*CounterInstrument*), ugaonog brojačkog instrumenta (*AngleCounterInstrument*), instrumenta pokazivača (*GaugeInstrument*), instrumenta horizonta (*HorizonInstrument*) i instrumenta labele (*LabelInstrument*). Svaki od izvedenih tipova nasleđuje strukturu jednostavnog instrumenta u pogledu postojanja odgovarajućih tipova skala, zona, pokazivača i markera. Zbog preglednosti ovi elementi su izostavljeni na dijagramu na slici 8.4.

Na dijagramu su prikazani izvedeni tipovi instrumenta horizonta – dvodimenzionalni (*2DHorizonInstrument*) i trodimenzionalni (*3DHorizonInstrument*) instrument horizonta. Na primeru ovog tipa instrumenta biće demonstrirana adaptacija interfejsa konkretnom korisniku na osnovu analize ograničenja i sklonosti ka korišćenju specifične funkcije čoveka, tj. vizuelno-prostorne percepcije (*eng. visuospatial perception*). Vizuelno-prostorna percepcija je mentalna funkcija koja na osnovu informacija dobijenih putem čula vida određuje relativnu poziciju objekta u odnosu na okruženje ili u odnosu na druge objekte u okruženju [ICF10]. U zavisnosti od stepena ograničenja i individualnih sklonosti ka korišćenju ove funkcije čoveka, operatoru će biti prezentovan odgovarajući tip instrumenta horizonta. Način određivanja i označavanja stepena ograničenja i sklonosti ka angažovanju funkcija čoveka je objašnjen u prethodnom poglavlju (odeljak 7.3.2).

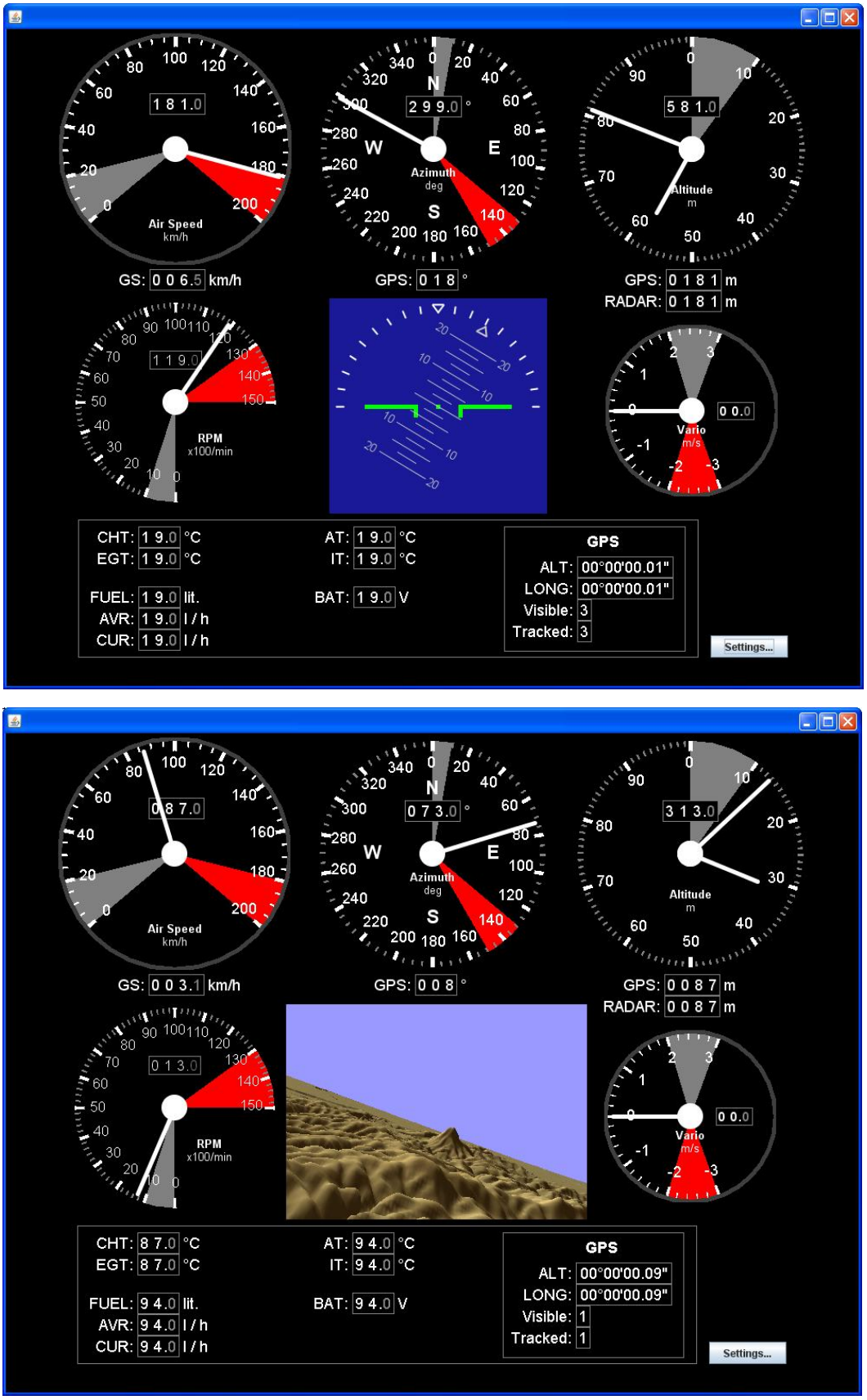
8.1.4. Model konačnog interfejsa instrument table

Na slici 8.5 je prikazan model instrument table BL prilagođen Java tehnologiji. Instrument tabla (*InstrumentsFrame*) se sastoji od dva panela – gornjeg (*InstrumentPanelUp*) i donjeg (*InstrumentPanelDown*). U gornjem panelu su smeštani instrumenti za čitanje letačkih parametara, dok se u donjem panelu nalaze instrumenti za prikaz parametara pogona BL. Klase okvira i panela instrument table nasleđuju odgovarajuće Java kontejnerske komponente iz modela Java Swing tehnologije.



Slika 8.5. *Model instrument table BL prilagođen Java platformi.*

Na slici 8.6 su prikazana dva slučaja izvršne verzije korisničkog interfejsa softverske instrument table. U oba primera instrumenti su grupisani na osnovu tipova parametara koje prezentuju. Tako su u gornjem delu smešteni instrumenti za prikaz letačkih parametara, dok su u donjem delu instrumenti za prezentaciju parametara pogona BL. Interfejsi se razlikuju u izgledu instrumenta horizont koji je smešten u gornjem panelu tako da zauzima centralno mesto interfejsa. Gornja instrument tabla sadrži dvodimenzionalni tip instrumenta i generisana je za korisnika koji ima nizak stepen sklonosti ka angažovanju funkcije vizuelno-prostorne percepcije. Donja instrument tabla sadrži trodimezionalnu varijantu horizonta koja obuhvata i prikaz reljefa terena. Ona je generisana za korisnika koji ima umeren ili visok stepen sklonosti ka angažovanju funkcije vizuelno-prostorne percepcije.



Slika 8.6. *Različiti slučajevi softverske intstrument table BL generisani na osnovu sklonosti korisnika ka korišćenju funkcije vizuelno-prostorne percepcije. Instrument table se razlikuju u izgledu instrumenta horizont.*

8.2. *Studija slučaja edukativnih igara*

U projektu vezanom za edukativne igre istraživali smo mogućnosti razvoja igara za učenje. Poslednjih godina primetna je široka upotreba video igara u edukativne svrhe. Rezultati istraživanja u ovoj oblasti se mogu naći i u većem broju specijalnih izdanja relevantnih časopisa i konferencija. Razlog za sve rasprostranjeniju upotrebu igara u oblasti edukacije pre svega leži u njihovim karakteristikama koje su značajne za sam proces učenja, od kojih se posebno izdvajaju:

- jasno definisani ciljevi;
- radnje (lekcije) koje se uvežbavaju (ponavljaju) dok se njima ne ovlada;
- praćenje progresa i prilagođavanje instrukcija nivou igrača;
- primena savladanih radnji;
- motivacija;
- razvoj apstraktnih kognitivnih veština kao što su rešavanje problema, analiza, planiranje i izvršavanje, i upravljanje resursima i dr.

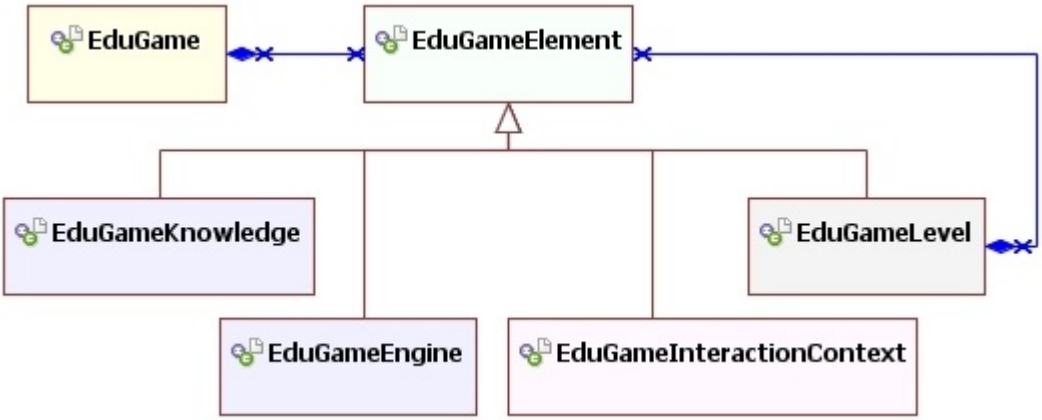
Kako bi se pomenute karakteristike igara iskoristile u edukativne svrhe, u njihovom razvoju je potrebno uzeti u obzir osobine čoveka relevantne za učenje.

Uopšteno posmatrano, edukativne igre se mogu posmatrati sa dva aspekta. Prvi je zajednički za većinu postojećih komercijalnih igara, a to je aspekt zabave koji treba da zainteresuje korisnika da igra igru. Drugi aspekt je iz domena edukacije i odnosi se na igre kao sredstvo za učenje i sticanje znanja. Ovde je ključni problem uspostavljanje formalnog mehanizma razvoja edukativne igre koji će integrisati pomenute aspekte. Pored toga, proces razvoja edukativne igre možemo razdvojiti na dve povezane aktivnosti. Jedna se odnosi na razvoj samog mehanizma funkcionisanja edukativne igre (*eng. game engine*), dok druga razmatra razvoj korisničkog interfejsa edukativne igre.

Ovde ćemo opisati istraživanje u oblasti interakcije čoveka i edukativne igre, tj. problem razvoja korisničkog interfejsa edukativne igre. Igru posmatramo kao sredstvo za edukaciju koje integriše modele igrača i koristi koncepte višenačinske komunikacije čoveka i računara u prenošenju edukativnog sadržaja. U razvoju korisničkog interfejsa edukativne igre naglasak je na modelima koji opisuju čoveka sa aspekta osobina relevantnih za proces učenja. U tom pogledu koristili smo kombinaciju tehnike modelovanja korisnika zasnovane na šablonima i tehnike zasnovane na osobinama (tehnike su opisane u odeljku 5.1.3). Kao kriterijum za kreiranje šablonskih modela korisnika smo uzeli teoriju višestruke inteligencije koja je našla širu primenu u domenu obrazovanja. Na osnovu ove teorije predviđeni su profili korisnika koji se razlikuju po tipu inteligencije čoveka. Svaki od profila je detaljnije opisan specifičnim skupom proširenja za modelovanje čoveka. Ovi profili su dopunjeni modelima koji opisuju domensko znanje korisnika. U nastavku ćemo opisati bitnije aspekte istraživanja sa primerima korišćenja predloženih proširenja za modelovanje kontekstno-osetljive interakcije čoveka i računara, pri čemu će naglasak biti na modelima čoveka. Više detalja se može naći u [Jovanović11].

8.2.1. Osnovni koncepti edukativne igre

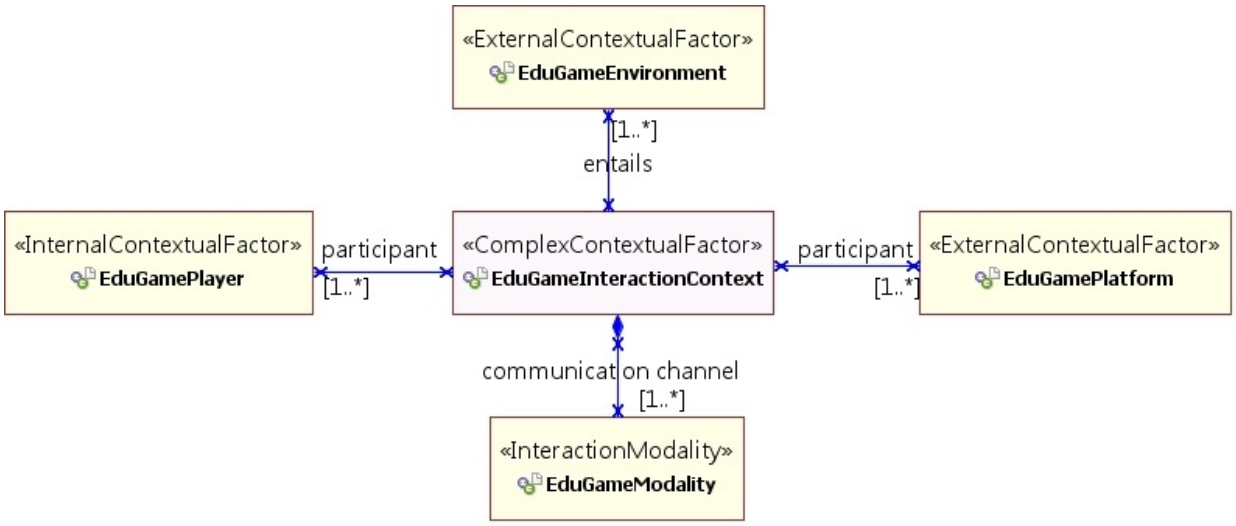
U procesu razvoja edukativne igre najpre smo definisali rečnik primitiva za modelovanje. Na slici 8.7 je prikazan pojednostavljen prikaz osnovnih koncepata edukativne igre [Jovanović08a].



Slika 8.7. *Osnovni koncepti modela edukativne igre.*

Koncept *EduGameElement* predstavlja osnovu za kreiranje drugih koncepata igre. Osnovni koncept za modelovanje domenskog znanja edukativne igre je *EduGameKnowledge*. Za modelovanje baze domenskog znanja korišćen je skup proširenja za modelovanje znanja čoveka opisan u odeljku 4.2.1.2. Mehanizam rada igre je opisan složenim konceptom *EduGameEngine*. U konkretnom slučaju to može biti, na primer, alat koji generiše testove iz određene oblasti i prikazuje ih studentu. Koncept konteksta interakcije igre i igrača je modelovan entitetom *EduGameInteractionContext*. *EduGameLevel* je složen koncept koji integriše prethodne primitive i predstavlja mehanizam evaluacije igrača u kojem se nivoi dovode u vezu sa ocenama. Na ovaj način je omogućeno kreiranje igara sa različitim nivoima potrebnog znanja i veština.

Na slici 8.8 su prikazane osnovne komponente koje određuju kontekst interakcije čoveka i edukativne igre [Jovanović08b].



Slika 8.8. *Osnovne komponente konteksno-osetljive interakcije čoveka i edukativne igre.*

Interakcija između igre i čoveka se odvija putem većeg broja načina komunikacije (*EduGameModality*). Oni se mogu posmatrati kao komunikacioni kanali za razmenu informacija između čoveka i igre i modelovani su kao stereotipovi načina komunikacije. Višenačinska komunikacija se može odvijati između većeg broja igrača (*EduGamePlayer*) i računarskih platformi na kojima se igra izvršava (*EduGamePlatform*). Sama komunikacija se odvija u okruženju (*EduGameEnvironment*) koje definiše skup spoljašnjih ograničenja. Osnovni elementi modelovanja čoveka, okruženja i platforme su modelovani kao odgovarajući stereotipovi konteksta interakcije.

8.2.2. Modelovanje profila čoveka u edukativnim igrama

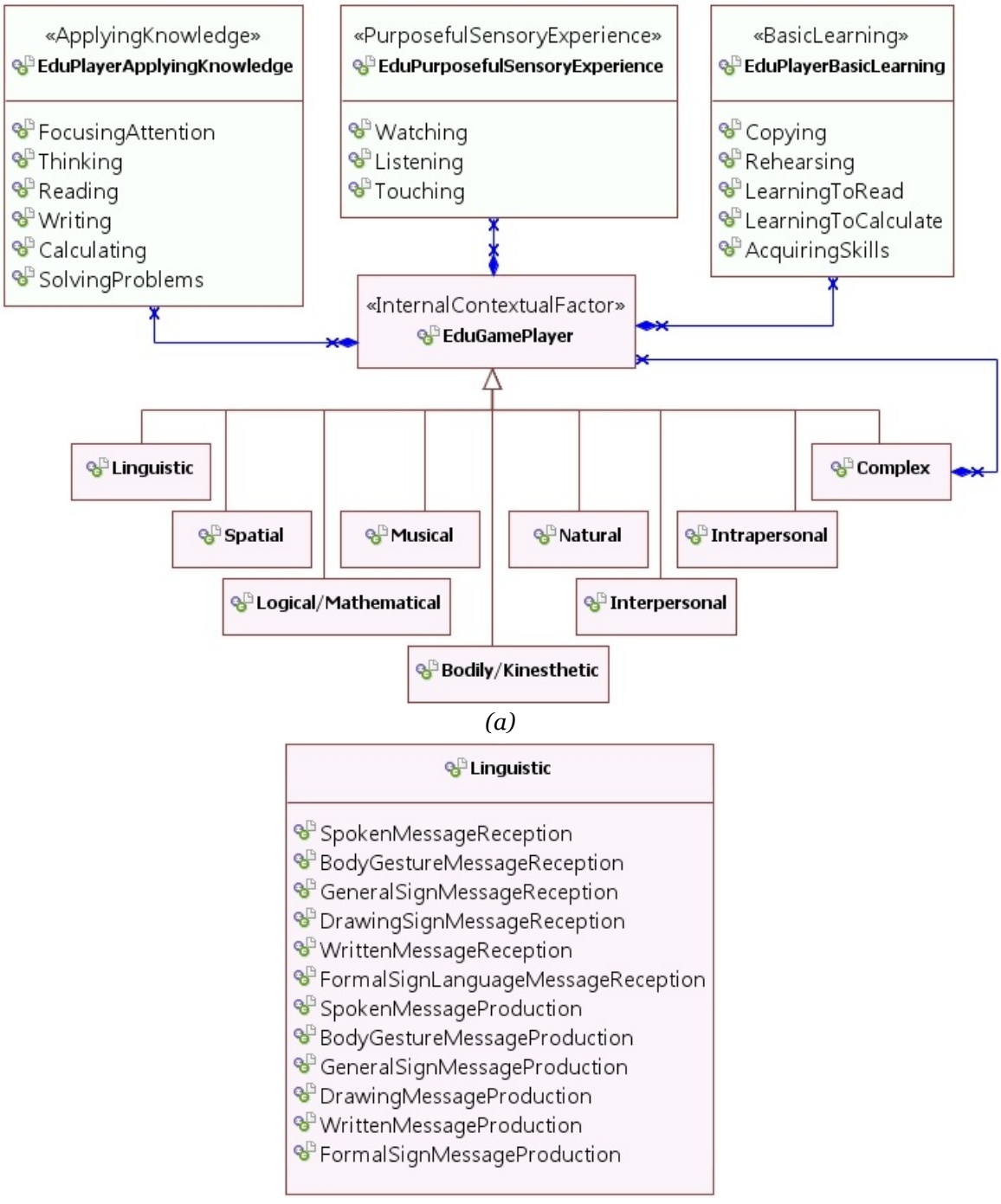
Klasifikacija tipova igrača edukativne igre je preuzeta iz teorije višestruke inteligencije (*eng. multiple intelligence theory*) [Gardner00]. Teorija višestruke inteligencije razmatra korišćenje specifičnih mogućnosti čoveka u procesu učenja i intelektualnog razvoja uopšte. Polazi od stanovišta da je tradicionalan način merenja inteligencije ograničen, jer ne uzima u obzir individualne karakteristike čoveka kao što su korišćenje specifičnih načina komunikacije i socijalna interakcija. Tvorac teorije, Hauard Gardner sa Univerziteta Harvard, je predložio osam tipova inteligencije čoveka:

- Lingvistički tip – uči i razvija sposobnosti korišćenjem reči jezika;
- Logički/matematički tip – uči i razvija sposobnosti korišćenjem brojeva i logike;
- Prostorni tip – uči i razvija sposobnosti korišćenjem vizuelnih i prostornih reprezentacija objekata;
- Muzički tip – uči i razvija sposobnosti korišćenjem melodija (muzike);
- Intrapersonalni tip – uči i razvija sposobnosti korišćenjem samoposmatranja;
- Kinestetički tip – uči i razvija sposobnosti korišćenjem fizičkog iskustva;
- Interpersonalni tip – uči i razvija sposobnosti na osnovu soci2jalnog iskustva;
- Naturalistički tip – uči i razvija sposobnosti na osnovu iskustva u prirodi.

Navedeni tipovi zapravo određuju načine na koji određena osoba uči. Na taj način teorija sugeriše različite oblike prezentovanja edukativnog materijala u cilju efektivnijeg učenja. Gardner ističe da je tradicionalni obrazovni proces uglavnom fokusiran na lingvistički i logički/metamatički tip i da ga je potrebno prilagoditi i ljudima koji ispoljavaju sklonosti u vezi sa ostalim tipovima inteligencija, kao što su na primer umetnici, arhitekte, muzičari, dizajneri i dr. Zbog toga smatramo da je u razvoju edukativne igre od ključnog značaja uzeti u obzir način na koji konkretan korisnik uči. Ova osobina čoveka je određena tipom inteligencije. Na osnovu tipa inteligencije čoveka biće generisan odgovarajući korisnički interfejs edukativne igre.

Na slici 8.9a je prikazana klasifikacija tipova igrača edukativne igre [Jovanović11]. Za detaljniji opis profila čoveka koristili smo stereotipove ICF klasifikacije za opis aktivnosti čoveka. Na taj način su zajedničke osobine profila čoveka u edukativnim igrama predstavljene ICF entitetima koji opisuju aktivnosti vezane za učenje i sticanje znanja (kategorija *LearningAndApplyingKnowledge*). Entiteti su razdvojeni po grupama funkcija koje se odnose na osnovne aktivnosti učenja (*BasicLearning*),

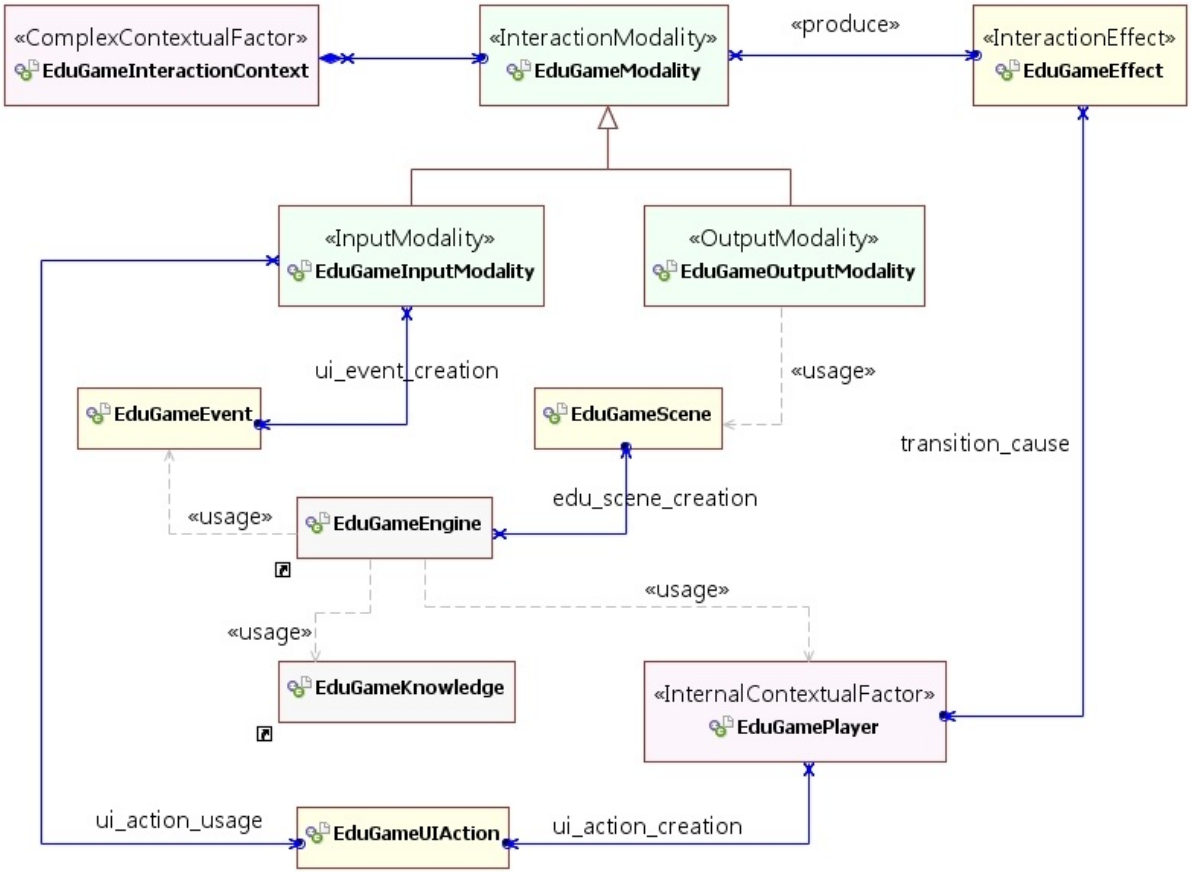
svrsishodno senzorsko iskustvo (*PurposefulSensoryExperience*) i primenu stečenog znanja (*ApplyingKnowledge*). Svaki od entiteta sadrži ugnježdene elemente koji su modelovani kao stereotipovi odgovarajućih podfunkcija. Uzimajući u obzir mogućnost da pojedinac može ispoljavati veći broj tipova inteligencije proširili smo postojeću klasifikaciju kompleksnim tipom (*Complex*) koji integriše jednostavne tipove. Na slici 8.9b prikazan je profil lingvističkog tipa koji je detaljnije opisan ICF proširenjima za modelovanje komunikacije čoveka putem prijema i slanja poruka korišćenjem određene vrste jezika.



Slika 8.9. *Predlog klasifikacije profila igrača edukativne igre (a). Profil lingvističkog tipa čoveka opisan ICF proširenjima za modelovanje komunikacije (b).*

8.2.3. Modelovanje interakcije čoveka i edukativne igre

Na slici 8.10 su prikazani osnovni elementi modela komunikacije čoveka i edukativne igre.



Slika 8.10. *Modelovanje komunikacije čoveka i edukativne igre.*

Interakcija čoveka i računara se odvija korišćenjem načina komunikacije specifičnih za edukativnu igru (*EduGameModality*). Načini komunikacije generišu efekte (*EduGameEffect*) koji mogu izazvati dovođenje igrača u odgovarajuće stanje poželjno za igru. Igrač inicira specifičnu akciju korisničkog interfejsa (*EduGameUIAction*) kao što je pokret miša ili korišćenje tastera tastature. Na osnovu ove akcije generiše se događaj edukativne igre (*EduGameEvent*). Nastanak događaja uzrokuje da mehanizam edukativne igre (*EduGameEngine*) generiše ili modifikuje određenu scenu (*EduGameScene*), pri čemu u obzir uzima znanje domenske oblasti i karakteristike čoveka. Scena je složen koncept i predstavlja korisnički interfejs edukativne igre. Model čoveka se formira pre početka sesije i ažurira se u toku igre praćenjem i analizom akcija igrača. Za inicijalno formiranje modela čoveka mogu se koristiti psihološki testovi.

Osnovne prednosti predloženog načina opisa interakcije jesu prilagođavanje interfejsa igre konkretnom korisniku i uspostavljanje formalnih relacija između kognitivnih efekata igre i stanja korisnika koja oni mogu izazvati. Razdvajanje i definisanje jasnih relacija između koncepata vezanih za logiku same igre sa jedne strane, i koncepata koji opisuju tok interakcije sa druge, povećava fleksibilnost razvoja. Izdvajanje parametara konteksta komunikacije čoveka i računara u posebne entitete omogućava očuvanje semantike interakcije pri modifikaciji edukativnog sadržaja igre.

Za opisivanje poželjnih stanja igrača u edukativnim igrama i kognitivnih efekata koji ih izazivaju iskoristili smo postojeća istraživanja u oblasti igara čiji se pregled može naći u [Jovanović11]. Tabela 8.1. daje primer kognitivnih efekata vezanih za stanje zadovoljstva, kao i predloženih načina komunikacije koji ih mogu izazvati.

Tabela 8.1. *Kognitivni efekti zadovoljstva i načini komunikacije koji ih mogu prouzrokovati.*

Kognitivni efekti karakteristični za dato stanje igrača edukativne igre		**Načini komunikacije**	
Stanje	***Efekat***	***Vizuelni***	***Audio***
Zadovoljstvo	**Fantazija**	**Opis scenarija igre korišćenjem slika, videa i animacija**	**Opis scenarija igre korišćenjem zvučnih efekata**
	Kontrola	**Navigacija korišćenjem miša i tastature**	**Navigacija prepoznavanjem glasa**
	Socijalna interakcija	**Vizuelni čet**	**Audio čet**
	Direktan odgovor sistema	**Vizuelna povratna sprega**	**Audio povratna sprega**
	Kreativnost	**Kreiranje vizuelnih elemenata i modifikacija postojećih**	**Kreiranje audio elemenata i modifikacija postojećih**

Na slici 8.11 je prikazan korisnički interfejs strategijske edukativne igre RIZIKO. U pitanju je eksperimentalni prototip čiji je razvoj vođen predloženim principima. Konkretna igra služi kao podrška kursu iz oblasti računarskih mreža. Znanje domenske oblasti je integrisano kroz pravila igre (prilikom zaposedanja odgovarajućih oblasti pojavljuju se dijalozi sa pitanjima iz domenske oblasti). Interfejs je prilagođen prostornom i interpersonalnom tipu korisnika. Vođenje korisnika kroz igru je realizovano vizuelnim elementima, dok je socijalna interakcija omogućena uvođenjem mehanizama komunikacije između korisnika (*chat room*). Igra je realizovana kao sastavni deo *Moodle* platforme za učenje na daljinu.

Slika 8.11. *Korisnički interfejs edukativne igre RIZIKO.*

8.3. *Rezime poglavlja*

U ovom odeljku demonstrirana je izvodljivost i praktična primena predloženog pristupa razvoju kontekstno-osetljivih korisničkih interfejsa. Primena predloženog pristupa je opisana na primerima dva projekta iz različitih domena. U projektu daljinskog nadzora korišćenjem bespilotne letelice opisan je proces projektovanja korisničkog interfejsa softverske instrument table. Sam proces razvoja je razmatran sa stanovišta modela korisničkih interfejsa, i sa stanovišta fizioloških i kognitivnih funkcija čoveka. U projektu vezanom za učenje putem igara opisan je razvoj korisničkog interfejsa igre sa stanovišta modela čoveka i interakcije sa računarom. Razvoj korisničkog interfejsa edukativne igre posmatra čoveka sa aspekta osobina vezanih za učenje, kao što su tipovi inteligencije i domensko znanje.

9. Zaključak

U ovom poglavlju dat je sažet pregled istraživanja. Najpre dajemo kratak osvrt na predloženo rešenje i hipotezu postavljenu na početku istraživanja. Nakon toga su istaknuti ostvareni rezultati i doprinosi teze. Na kraju navodimo mogućnosti korišćenja predloženog rešenja u drugim oblastima i pravce daljeg razvoja.

9.1. *Osvrt na rešenje i postavljene hipoteze*

U ovom radu smo istraživali problem unapređenja razvoja kontekstno-osetljivih korisničkih interfejsa. U skladu sa tim predložili smo rešenje koje omogućava automatizaciju razvoja korisničkog interfejsa prilagođenog kontekstu interakcije čoveka i računara. Rešenje se ogleda u proširenju jezika za modelovanje, standardne metodologije razvoja softverskih sistema i razvojnih alata elementima specifičnim za interakciju čoveka i računara.

U razvoju predloženog rešenja oslonili smo se na principe modelski zasnovanog razvoja softvera u kojem se platformski nezavisna specifikacija softvera postepeno ili automatizovano prevodi u izvršne aplikacije za različite softverske i hardverske platforme. Arhitektura upravljana modelima, koja se koristi za razvoj složenih programskih rešenja, hijerarhijski organizuje koncepte i modele u više nivoa, što je posebno bitno imajući u vidu da je razvoj kontekstno-osetljivih korisničkih interfejsa složen proces koji uključuje modelovanje velikog broja elemenata na različitim nivoima apstrakcije. Arhitektura upravljana modelima je zasnovana na standardnim i široko prihvaćenim tehnologijama čime je moguće ostvariti veću praktičnu upotrebljivost rešenja, jer se mogu koristiti postojeći alati za projektovanje sa kojima je upoznat veći broj analitičara, projektanata i programera. Osnovna prednost modelski zasnovanog pristupa je mogućnost opisa sistema korišćenjem koncepata koji su manje ograničeni mogućnostima implementacione platforme, a bliži su domenu problema koji se rešava. Ovo čini model lakšim za definisanje, razumevanje i održavanje, a otvorena je i mogućnost da čak i domenski ekspert može samostalno da kreira model. Na ovaj način moguće je povećati produktivnost rada analitičara, projektanata i programera, i obezbediti izradu kvalitetnijih korisničkih interfejsa. Sa druge strane, unapređenja korisničkih interfejsa mogu povećati i produktivnost rada krajnih korisnika softverskih sistema.

U skladu sa prethodnim, razvijen je model kontekstno-osetljive interakcije čoveka i računara i predloženi su modeli korisničkih interfejsa na različitim nivoima apstrakcije. Polazna hipoteza je da je pomoću predloženih modela moguće kreirati arhitekturu upravljanu modelima koja će unaprediti razvoj i upotrebljivost kontekstno-osetljivih korisničkih interfejsa.

9.2. *Ostvareni doprinosi*

U tezi smo izložili više rezultata do kojih smo došli tokom istraživanja, a koji se mogu koristiti u cilju unapređenja razvoja i korišćenja kontekstno-osetljivih korisničkih interfejsa.

Prvi doprinos je izvršena analiza sa kritičkim osvrtom na pogodnosti korišćenja postojećih rezultata u razvoju kontekstno-osetljivih korisničkih interfejsa. Za potrebe analize dat je sveobuhvatan i sistematizovan pregled postojećih rešenja u oblasti razvoja korisničkih interfejsa. Pored toga, izvršena je detaljna analiza i dat uporedni pregled jezika za razvoj, jezika za transformacije i alata za razvoj korisničkih interfejsa. Analiza je izvršena kroz glavne aspekte kontekstno-osetljivih interfejsa, tj. aspekt modelovanja računarske platforme, aspekt modelovanja okruženja, aspekt modelovanja korisnika i aspekt razvoja. Na osnovu izvršene analize uočili smo neke elemente koji se dalje mogu unaprediti i na taj način poboljšati kvalitet korisničkih interfejsa kako sa stanovišta korišćenja, tako i sa stanovišta razvoja. Baza znanja kreirana za potrebe analize se može koristiti kao osnova za buduća istraživanja u oblasti interakcije čoveka i računara, i kao materijal za samostalnu edukaciju ili držanje kurseva iz predmetne oblasti.

Drugi doprinos rada jeste generički model kontekstno-osetljive interakcije čoveka i računara. Ovaj model je rezultat primene interdisciplinarnog pristupa i objedinjavanja znanja iz više naučnih oblasti od interesa za kontekstno-osetljivu interakciju između čoveka i računara. Generički model predstavlja jedinstven i formalan opis osnovnih koncepata i relacija između faktora relevantnih za razvoj kontekstno-osetljivog korisničkog interfejsa. Faktori konteksta interakcije definisani su u okviru odgovarajućih podmodela - modela čoveka, modela okruženja, modela računarskih uređaja i modela višenačinske komunikacije. Model čoveka objedinjuje koncepte korisnika sa stanovišta anatomskih i fizioloških osobina čoveka sa jedne, i osobina čoveka vezanih za određenu oblast (znanje, interesovanja, kognitivni stil i dr.) sa druge strane. Na ovaj način se mogu modelovati osobine korisnika relevantne za konkretan domen korišćenja korisničkog interfejsa. Pri opisivanju koncepata okruženja i računarskog uređaja se polazi od modela čoveka, drugim rečima, koncepta fizičkog stimulansa ili draži koje čovek kreira ili detektuje. Model višenačinske komunikacije definiše relacije između faktora čoveka, uređaja i okruženja. Korišćenjem predloženog generičkog modela može se uspostaviti veza između postojećih istraživanja koja razmatraju kontekst sa različitih stanovišta. Jedan pravac istraživanja posmatra kontekst sa stanovišta korisnika i preovladava u oblastima koje se bave modelovanjem korisnika. Drugi pravac razmatra kontekst sa stanovišta računarskog uređaja i zastupljen je u oblastima mobilnog računarstva i sveprisutnog računarstva. Uopšteno posmatrano, model kontekstno-osetljive interakcije čoveka i računara se može koristiti i kao ontologija znanja potrebnih za razvoj korisničkih interfejsa, jer sadrži pregled najznačajnijih znanja i rezultata istraživanja iz predmetnih oblasti.

Treći doprinos se ogleda u primeni generičkog modela kontekstno-osetljive interakcije čoveka i računara u definisanju domenski specifičnog jezika za opis korisničkih interfejsa. Kako bi model bio upotrebljiv u razvoju softvera i u cilju definisanja semantičkih proširenja razvojnih okruženja, uvedene su primitive specifične za modelovanje kontekstno-osetljive interakcije čoveka i računara, i korisničkih interfejsa na različitim nivoima apstrakcije. Zbog standardizacije, široke prihvaćenosti, i dostupnosti razvojnih alata, mogućnost kreiranja domenski specifičnog jezika prvenstveno je razmatrana kroz proširenje UML jezika. Predložena proširenja UML-a omogućuju bitno unapređenje razvoja korisničkih interfejsa uopšte, jer postojeći specijalizovani alati problem interakcije između čoveka i računara rešavaju parcijalno i korišćenjem nestandardnih metodologija, što značajno umanjuje

njihovu praktičnu upotrebljivost, a usložnjava integraciju korisničkog interfejsa sa ostatkom softverskog sistema. Za proširivanje UML-a korišćen je mehanizam profila. Na taj način su kreirani profili za modelovanje konteksno-osetljive interakcije čoveka i računara i profili za modelovanje korisničkih interfejsa na različitim nivoima apstrakcije.

Sledeći doprinos predstavlja predlog proširenja standardne metodologije razvoja (*Unified* proces) sa konceptima iz modela kontekstno-osetljive interakcije između čoveka i računara i razvojnim modelima korisničkih interfejsa. U radu je opisana sistematična primena predloženih UML proširenja u procesu razvoja softvera. Upotreba uvedenih proširenja omogućava formalan i detaljniji opis kontekstno-osetljive interakcije između čoveka i računara u aktivnostima specifikacije zahteva, analize, dizajna i implementacije. Za svaku od razvojnih aktivnosti definisali smo odgovarajuća semantička proširenja. Tako je, na primer, uvođenjem proširenja za modelovanje zadataka u ranim fazama razvoja moguće sagledati kako funkcionalne, tako i nefunkcionalne zahteve sistema (faktori korisnika, okruženja, uređaja i interakcije), za razliku od standardnog procesa razvoja u kojem se korišćenjem slučajeva korišćenja u principu mogu sagledati prvenstveno funkcionalni zahtevi sistema. Proširenja UML jezika i standardnog procesa razvoja su implementirana u EMF okruženju za modelski zasnovan razvoj softvera.

Značajan doprinos jeste i predlog automatizacije procesa razvoja kontekstno-osetljivih korisničkih interfejsa. U skladu sa tim je razvijen integrisani sistem transformacija modela kontekstno-osetljivih korisničkih interfejsa na različitim nivoima apstrakcije. Osnovu za kreiranje transformacija modela predstavljaju UML proširenja za modelovanje kontekstno-osetljive interakcije čoveka i računara i kontekstno-osetljivih korisničkih interfejsa. Ovde je demonstrirana upotreba standardnih tehnologija za dizajn i transformacije modela kontekstno-osetljivih korisničkih interfejsa. Transformacije modela korisničkih interfejsa su implementirane u ATL tehnologiji u obliku *plug-in* komponenti i kao takve su prilagodljive, ponovno upotrebljive i proširive. Na ovaj način je moguće povećati produktivnost razvoja korisničkih interfejsa, a da pri tome bude očuvana semantika interakcije čoveka i računara, kao i mogućnost integracije sa ostatkom softverskog sistema. Pored toga, razvijeni alati za transformacije nisu vezani isključivo za razvoj korisničkih interfejsa, već su primenjivi i u drugim oblastima.

I na kraju, praktičan doprinos istraživanja predstavlja demonstracija izvodljivosti i praktične upotrebljivosti predloženog rešenja na primeru dve studije slučaja iz različitih domena. U projektu daljinskog nadzora korišćenjem bespilotne letelice opisan je proces projektovanja korisničkog interfejsa softverske instrument table, dok je u projektu vezanom za učenje putem igara opisan razvoj korisničkog interfejsa edukativne igre.

9.3. *Mogućnosti primene i daljeg istraživanja*

Rezultati izloženi u radu predstavljaju dobru osnovu za dalja istraživanja i nadgradnju. Pre svega, model kontekstno-osetljive interakcije se može proširiti i integrisati sa znanjima iz drugih disciplina koje se bave problemom modelovanja korisnika i integracije krajnjeg korisnika u proces razvoja

softvera, kao što su dizajn orijentisan ka korisniku (*eng. user-centered design*) i softversko inženjerstvo krajnjih korisnika (*eng. end-user software engineering*), respektivno.

Kada govorimo o kontekstno-osetljivom računarstvu generalno, predloženo rešenje se može integrisati sa postojećim softverskim arhitekturama kod kojih je naglasak na prikupljanju, obradi i apstrakciji podataka konteksta. Ovde do izražaja dolaze uvedena proširenja za modelovanje uređaja i okruženja interakcije.

U oblastima vezanim za razvoj interaktivnih softverskih sistema, predložena proširenja za modelovanje korisnika i korisničkih interfejsa se mogu koristiti u kombinaciji sa alatima za skiciranje (*eng. sketching tools*) i brzu izradu prototipova korisničkih interfejsa (*eng. rapid prototyping tools*).

Na kraju, moguća je i komercijalna primena izloženih doprinosa u oblasti razvoja softverskih sistema. Projektanti i programeri mogu da iskoriste razvijena proširenja jezika UML, procesa razvoja i alata u cilju unapređenja razvoja korisničkih interfejsa.

Reference

Abowd00 Abowd, G., Mynatt, E., “Charting past, present, and future research in ubiquitous computing”, ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1, March 2000, Pages 29–58.

Abrams99 Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., Shuster, J. E., “UIML: An appliance-independent XML user interface language”, Proceedings of the 8th International World Wide Web Conference, pp. 617-630.

Allanson02 Allanson, J., "Electrophysiologically Interactive Computer Systems", IEEE Computer, March 2002, pp. 60-65.

Allen83 Allen, J.F., “Maintaining Knowledge about Temporal Intervals,” Communications of the ACM, Vol. 26, No. 11, Nov. 1983, pp. 832-843.

Anhalt01 Anhalt, J., Smailagic, A., Siewiorek, D., Gemperle, F., Salber, D., Weber, S., Beck, J., Jennings, J., “Toward Context-Aware Computing: Experiences and Lessons”, IEEE Intelligent Systems, Vol. 16 , No. 3, pp. 38-46, May 2001.

Aquino09 Aquino, N., “Adding flexibility in the model-driven engineering of user interfaces”, Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems EICS’09, July 15–17, 2009, Pittsburgh, Pennsylvania, USA, pp. 329-332.

Aquino10 Aquino, N., Vanderdonckt, J., Pastor, O., “Transformation templates: adding flexibility to model-driven engineering of user interfaces”, Proceedings of the 2010 ACM Symposium on Applied Computing SAC '10, March 22–26, 2010, Sierre, Switzerland, pp. 1195-1202.

Aranda07 Aranda, A.H.M., Ibarra, O.M., “A Context Sensitive Public Display for Adaptive Multi-User Information Visualization”, Proceedings of the Third International Conference on Autonomic and Autonomous Systems ICAS '07, June 2007, IEEE CS

Atkinson03 Atkinson, C., Kohne, T., "Model-Driven Development: A metamodeling foundation", IEEE Software, Vol. 20, No. 5, Sep/Oct. 2003, pp. 36-41

ATL11 Atlas Transformation Language Home Page, http://www.eclipse.org/m2m/atl/

Baddeley03 Baddeley, A.,”WORKING MEMORY: LOOKING BACK AND LOOKING FORWARD”, Nature Reviews Neuroscience, Vol. 4, October 2003, pp. 829-839.

Ballagas06 Ballagas, R., Borchers, J., Rohs, M., Sheridan, J., “The Smart Phone: A Ubiquitous Input Device”, IEEE Pervasive Computing, January–March 2006, pp. 70-77.

Barboni10 Barboni, E., Ladry, J.F., Navarre, D., Palanque, P., Winckler, M., “Beyond modelling: an integrated environment supporting co-execution of tasks and systems models”, Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS’10, June 19–23, 2010, Berlin, Germany, pp. 165-174.

Bass92 Bass, L., “A metamodel for the runtime architecture of an interactive system: The UIMS developers workshop”, SIGCHI Bulletin, 24(1), pp. 32-37, 1992.

Bass98 Bass, L., Clements, P., Kazman, R., “Software Architecture in Practice”, Addison Wesley Publ., ISBN 0-201-19930-0 (1998).

Bellotti08 Bellotti, V., et al., “Activity-based serendipitous recommendations with the Magitti mobile leisure guide”, Proceeding of the ACM SIGCHI conference on Human factors in computing systems CHI '08, April 5–10, 2008, Florence, Italy, pp. 1157-1166.

Bergh05 Bergh, J. V. D., Coninx, K., “Towards Modeling Context-Sensitive Interactive Applications: the Context-Sensitive User Interface Profile (CUP)”, Proceedings of the 2005 ACM symposium on Software visualization SoftVis ’05, New York, NY, USA, pp. 87–94.

Bergh07 Bergh, J.V.D., Coninx, K., “From Task to Dialog Model in the UML”, Proceedings of the 6th international workshop on Task models and diagrams TAMODIA '07, Lecture Notes in Computer Science, Vol. 4849, pp. 98-111.

Bernsen94 Bernsen, N.O., "Foundations Of Multimodal Representations: A taxonomy of representational modalities", Interacting with Computers, 6, 1994, pp. 347-371

Berstel05 Berstel, J., Reghizzi, S., Roussel, G., Pietro, P., "A scalable formal method for design and automatic checking of user interfaces", ACM Transactions on Software Engineering and Methodology, Vol. 14, No. 2, April 2005, pp. 124–167.

Beyer98 Beyer, H., Holtzblatt K., 1998. Contextual Design. Morgan Kaufmann Publishers, San Francisco.

Bézivin03 Bézivin, J., Dupe, G., Jouault, F., Pitette, G., Rougui, J. E., "First experiments with the ATL model transformation language: Transforming XSLT into XQuery", 2nd OOPSLA Workshop on Generative Techniques in the context of MDA, Anaheim, CA, USA, 2003.

Bezivin06 Bezivin, J., Buttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A., "Model transformations? Transformation models!", Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, LNCS 4199, Genoa, Italy, Springer, Berlin, 2006; pp 440–453.

Bishop06 Bishop, J., "Multi-platform user interface construction: a challenge for software engineering-in-the-small", Proceedings of the 28th international conference on Software engineering, ICSE '06, May 20–28, 2006, Shanghai, China, pp. 751-760.

Blumendorf10 Blumendorf, M., Lehmann, G., Albayrak, S., "Bridging models and systems at runtime to build adaptive user interface", Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS'10, June 19–23, 2010, Berlin, Germany, pp.9-18.

Booch99 Booch, G., Rumbaugh, J., Jacobson, I., "The Unified Modeling Language", Addison-Wesley, 1999, ISBN 0-201-57168-4

Breiner09 Breiner, K., Maschino, O., Görlich, D., Meixner, G., "Towards automatically interfacing application services integrated into an automated, model-based user interface generation process", Proceedings of the IUI'09 Workshop on Model Driven Development of Advanced User Interfaces MDDAUI '09, 2009.

Brown04 Brown, S.S., "Conversion of notations", Technical report, University of Cambridge, 2004.

Bruck08 Bruck, J., Hussey, K., "Customizing UML: Which Technique is Right for You?", IBM Technical Paper, June 2008, http://www.eclipse.org/modeling/mdt/uml2/docs/articles

Brusilovsky07 Brusilovsky, P., Millán, E., "User Models for Adaptive Hypermedia and Adaptive Educational Systems", In P. Brusilovsky, A. Kobsa, and W. Nejdl (Eds.): The Adaptive Web, LNCS 4321, pp. 3 – 53, 2007.

Bruyère05 Bruyère, S., VanLooy, S., Peterson, D., "The international classification of functioning, disability and health: contemporary literature overview", Rehabil. Psychol. 50(2), 113–121 (2005)

Butter07 Butter, T., Aleksy, M., Bostan, P., Schader, M., "Context-aware User Interface Framework for Mobile Applications", Proceedings of the 27th International Conference on Distributed Computing Systems Workshops ICDCSW '07, IEEE Computer Society

Calvary03 Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., "A Unifying Reference Framework for Multi-Target User Interfaces", Interacting with Computers, Vol. 15, No. 3, June 2003, pp. 289–308.

Calvary09 Calvary, G., Demeure, A., "Context-aware and mobile interactive systems: the future of user interfaces plasticity", Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems EICS'09, July 15–17, 2009, Pittsburgh, Pennsylvania, USA.

Cao09 Cao, Y., Theune, M., Nijholt, A., "Modality effects on cognitive load and performance in high-load information presentation", Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI'09, Sanibel Island, Florida, USA February 08 - 11, 2009, pp. 335-344.

Card83 Card, S.K., Moran, T.P., Newell, A., 1983. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Ceri07 Ceri, S., Daniel, F., Matera, M., Facca, F., "Model-Driven Development of Context-Aware Web Applications", ACM Transactions on Internet Technology, Vol. 7, No. 1, Article 2, 33 pages, February 2007.

Chang09 Chang, C.K., Jiang, H., Ming, H., Oyama, K., "Situ: A Situation-Theoretic Approach to Context-Aware Service Evolution", IEEE Transactions on Services Computing, Vol. 2, No. 3, July-September 2009, pp. 261-275.

Chen02 Chen, S.Y., Macredie, R.D.: Cognitive styles and hypermedia navigation: Development of a learning model. Journal of the American Society for Information Science and Technology 53, 1 (2002) 3-15

Chen04 Chen, H., Perich, F., Finin, T., Joshi, A., "Soupa: standard ontology for ubiquitous and pervasive applications", In Proceedings of the 1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, pages 258–267, 2004.

Cheng09 Cheng, W., Gotz, D., "Context-based page unit recommendation for web-based sensemaking tasks", Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI'09, Sanibel Island, Florida, USA February 08 - 11, 2009, pp. 107-116.

Chittaro11 Chittaro. L., Carchietti, E., De Marco, L., Zampa, A., "Personalized emergency medical assistance for disabled people", User Model User-Adap Inter., Springer Science, pp. 1-34., 2011.

Cockton95 Cockton, G., Clarke S., Gray, P., Johnson, C., 1995. Literate Development: Weaving Human Context into Design Specifications. In: Palanque, P., Benyon, D. (Eds), Critical Issues in User Interface Engineering, Springer-Verlag, London, pp. 227–248.

Collignon08 Collignon, B., Vanderdonckt, J., Calvary, G., "Model-Driven Engineering of Multi-target Plastic User Interfaces", Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems ICAS '08, March 2008, pp. 7-14

Congleton08 Congleton, B., Ackerman, M., Newman, M., "The ProD Framework for Proactive Displays", Symposium on User Interface Software and Technology archive Proceedings of the 21st annual ACM symposium on User interface software and technology UIST'08, Monterey, CA, USA October 19 - 22, pp. 221-230.

Conraets04 Coenraets, C., 2004. An overview of MXML: The Flex markup language. http://www.adobe.com/devnet/flex/articles/paradigm.html

Constantine03 Constantine, L., "Canonical Abstract Prototypes for Abstract Visual and Interaction Design", Proceedings of DSV-IS, Springer Vol. 2844 (2003), pp. 1-15.

Constantine09 Constantine, L., "Interaction Design and Model-Driven Development", Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems MODELS '09, October 2009, LNCS 5795, p. 377.

Cordy06 Cordy, J.R., "The TXL Source Transformation Language", Science of Computer Programming, Vol. 61, 2006, pp. 190–210.

Coutaz02 Coutaz, J., Rey, G., Foundations for a Theory of Contextors. Proceedings of 4th International Conference of Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002). Kluwer Academics, Dordrecht, pp. 13–33.

Coutaz05 Coutaz, J., Crowley, J., Dobson, S., Garlan, D., "Context is key", Communications of the ACM, Vol. 48, No. 3, May 2005, pp. 49-53.

Coutaz10 Coutaz, J., "User interface plasticity: model driven engineering to the limit!", Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS'10, June 19–23, 2010, Berlin, Germany, pp. 1-8.

Coutaz87 Coutaz, J., "PAC An Implemention Model for Dialog Design", Proceedings of Interact'87, pp. 431-436, 1987.

Crowley00 Crowley, J., Coutaz, J., Bérard, F., "Things that See", Communications of the ACM, vol 43, no. 3, pp. 54-64 (2000).

Crowley02 Crowley, J., Coutaz, J., Rey, G., Reignier, P., 2002. Perceptual Components for Context- Aware Computing. Proceedings of International Conference on Ubiquitous Computing UbiComp'2002 (Göteborg, 29 September-October 1 2002). Lecture Notes in Computer Science Vol. 2498, Springer Verlag, Berlin, pp. 117–134.

Czarnecki03 Czarnecki, K., Helsen, S., "Classification of Model Transformation Approaches", ACM OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, pp. 1-17.

Czarnecki06 Czarnecki, K., Helsen, S., "Feature-Based Survey of Model Transformation Approaches", IBM Systems Journal, 45(3),2006, pp. 621-645.

da Silva03 da Silva, P.P., Paton, N., “User Interface Modeling in UMLi”, IEEE Software, Vol. 20, No. 4, July 2003, pp. 62-69.

Demeure05 Demeure, A., Calvary, G., Sottet, J.S., Vanderdonkt, J., “A Reference Model for Distributed User Interfaces”, Proceedings of the 4th international workshop on Task models and diagrams TAMODIA '05, September 26–27, 2005, Gdansk, Poland, pp. 79-86.

Dey01a Dey, A., Abowd, G., Salber, D., 2001. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction 16 (2-4), 97–166.

Dey01b Dey, A.K., “Understanding and Using Context”, Personal and Ubiquitous Computing (2001) 5: 4–7, Springer-Verlag London.

Dey05 Dey, A., Mankoff, J., “Designing mediation for context-aware applications”, ACM Transactions on Computer-Human Interaction (TOCHI) , Vol. 12, No. 1, May 2005, pp. 53-80.

Dey09 Dey, A., Newberger, A., “Support for context-aware intelligibility and control”, Proceedings of the 27th international conference on Human factors in computing systems CHI '09, April 4–9, 2009, Boston, MA, USA, pp. 859-868.

Dourish04 Dourish, P., “What we talk about when we talk about context”, Personal and Ubiquitous Computing (2004) 8: 19–30.

Eisenstein01 Eisenstein, J., Vanderdonckt, J., Puerta, A., “Model-Based User-Interface Development Techniques for Mobile Computing”, Proceedings of ACM International Conference on Intelligent User Interfaces IUI’2001 (Santa Fe, 14-17 January 2001). ACM Press, New York, pp. 69–76.

Ellis94 Ellis, C., Wainer, J., “A Conceptual Model of Groupware”, in Proceedings CSCW’94, ACM Conference on Computer Supported Cooperative Work, Furuta, R., Neuwirth, C. eds., 79-88 (1994).

Engelbart68 Engelbart, D., English, W. “A Research Center for Augmenting Human Intellect.”, ACM SIGGRAPH Video Review 106 (1994), originalan vidoe zapis napravljen 1968.

English67 English, W.K., Engelbart, D.C., Berman, M.L. “Display selection techniques for text manipulation.” IEEE Transactions on Human Factors in Electronics HFE-8, 1 (1967), pp. 5–15.

Falb09 Falb, J., Kavaldjian, S., Popp, R., Raneburger, D., Arnautovic, E., Kaindl, H., “Fully automatic user interface generation from discourse models”, Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI’09, Sanibel Island, Florida, USA February 08 - 11, 2009, Demonstration Session.

Farmer03 Farmer, E., Brownson, A., “Review of workload measurement, analysis and interpretation methods - physiological measures, performance-based measures, and self-assessment measures”, CARE-Integra-TRS-130-02-WP2, 2003.

Fiala05 Fiala, Z., Houben, G.-J., “A Generic Transcoding Tool for Making Web Applications Adaptive”, Proceedings of the CAiSE’05 Forum. CEUR Workshop Proceedings, 2005.

Fitts54 Fitts, P. M. 1954. "The information capacity of the human motor system controlling theamplitude of movement". J. Exper. Psych. 47, 6, 381–391.

Foley84 Foley, D., Wallace, L., Chan, P., “The Human Factors of Computer Graphics Interaction Techniques,” *IEEE Computer Graphics and Applications*, vol. 4, no. 11, 1984, pp. 13–48.

Foley94 Foley, D., Noi Sukaviriya, P., “History, results, and bibliography of the user interface design environment (UIDE), an early model-based system for user interface design and implementation”, Proceedings of Design, Verification and Specification of Interactive Systems (DSVIS’94), 1994, pp. 3–14.

France07 France, R., Rumpe, B., “Model-driven Development of Complex Software: A Research Roadmap”, Future of Software Engineering FOSE '07, IEEE CS, May 2007.

Frey10 Frey A.G., Calvary, G., Chesa, S.D., “Xplain: an editor for building self-explanatory user interfaces by model-driven engineering”, Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS’10, June 19–23, 2010, Berlin, Germany, pp. 41-46.

Gajos06 Gajos, K., Czerwinski, M., Tan, D., Weld, D.,"Exploring the design space for adaptive graphical user interfaces", Proceedings of the Conference on Advanced Visual Interfaces AVI'06, ACM Press, New York, NY, USA, 2006, pp. 201–208.

Gajos07 Gajos, K., Wobbrock, J., Weld, D., "Automatically generating user interfaces adapted to users' motor and vision capabilities", Proceedings of the 20th annual ACM symposium on User interface software and technology UIST'07, Newport, Rhode Island, USA, October 07 - 10, 2007, pp. 231 – 240.

Gajos10 Gajos, K., Weld, D., Wobbrock, J., "Automatically generating personalized user interfaces with Supple", Artificial Intelligence, Vol. 174, No. 12, Elsevier Science Publishers, 2010, pp. 910–950.

Gardner00 Gardner, H., *Intelligence Reframed: Multiple Intelligences for the 21st Century*. New York: Basic Books, 2000.

Gardner02 Gardner, H., "Multiple intelligences: The theory in practice", 2002.

Gardner93 Gardner, H., "Frames of mind: The theory of multiple intelligences", 1993.

Gasevic06 Gasevic, D., Djuric, D., Devedzic, V., Selic, B., "Model Driven Architecture and Ontology Development", Springer-Verlag New York, Inc., July 2006.

Geary97 Geary, D.M., McClellan, A.L.,"Graphic Java: mastering the AWT", SunSoft Press, 1997.

Geary99 Geary, D.M.,"Graphic Java 2, Swing, 3/e", Prentice Hall PTR, 1999.

Gentner90 Gentner R., Grudin, J., "Why good engineers (sometimes) create bad interfaces", Conference on Human Factors and Computing Systems, 1990 , Seattle, Washington, United States, Publisher ACM Press New York, NY, USA

Gonzales00 Gonzales, R., "Disciplining Multimedia", IEEE MultiMedia, July-September 2000, pp. 72-78.

Goth10 Goth, G., "The eyes have it", Communications of the ACM, Vol.53 , No. 12, December 2010, pp. 13-15.

Grudin90 Grudin, J., "The computer reaches out: the historical continuity of interface design", Conference on Human Factors and Computing Systems, 1990 , Seattle, Washington, United States, Publisher ACM Press New York, NY, USA

Gui09 Gui, F., Adjouadi, M., Rishe, N., "A Contextualized and Personalized Approach for Mobile Search", Proceedings of the 2009 International Conference on Advanced Information Networking and Applications Workshops, IEEE CS, pp.966-971.

Haan10 Haan, G., Piguillet, H., Post, F., "Spatial Navigation for Context-Aware Video Surveillance", IEEE Computer Graphics and Applications, Vol. 30, No. 5, pp. 20-31, September 2010.

Häkkilä06 Häkkilä, J., Mäntyjärvi, J., "Developing design guidelines for context-aware mobile applications", Proceedings of the ACM international conference on Mobile technology, applications & systems Mobility '06, Oct. 25–27, 2006, Bangkok, Thailand, pp. 1-7.

Hartmann08 Hartmann, M., Zesch, T., Mühlhäuser, M., Gurevych, I., "Using Similarity Measures for Context-Aware User Interfaces", Proceedings of the IEEE International Conference on Semantic Computing ICSC '08, pp. 190-197.

Hartmann09a Hartmann, M., Schreiber, D., Mühlhäuser, M., "AUGUR: providing context-aware interaction support", Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems EICS'09, July 15–17, 2009, Pittsburgh, Pennsylvania, USA, pp. 123-132.

Hartmann09b Hartmann, M., Mühlhäuser, M., "Context-Aware Form Filling for Web Applications", Proceedings of the 2009 IEEE International Conference on Semantic Computing ICSC '09, September 2009, pp. 221-227.

Hatala05 Hatala, M., Wakkary, R., "Ontology-Based User Modeling in an Augmented Audio Reality System for Museums", User Modeling and User-Adapted Interaction , Vol. 15 , No. 3-4, Kluwer Academic Publishers, August 2005, pp. 339–380.

Heinrich07 Heinrich, M., Winkler, M., Steidelmuller, H., Zabelt, M., Behring, A., Neumerkel, R., Strunk, A., "MDA Applied: A Task-Model Driven Tool Chain for Multimodal Applications", Proceedings of the 6th international workshop on Task models and diagrams TAMODIA '07, Lecture Notes in Computer Science, Vol. 4849, pp. 15-27.

Helms08 Helms, J., Abrams, M., “Retrospective on UI description languages based on eight years’ experience with the user interface markup language (UIML)”, International Journal on Web Enginering Technology, Vol. 4, No. 2, pp. 138–162, 2008.

Ho05 Ho, J., Intille, S., “Using context-aware computing to reduce the perceived burden of interruptions from mobile devices”, Proceedings of the SIGCHI conference on Human factors in computing systems CHI '05, April 2–7, 2005, Portland, Oregon, USA, pp. 909-918.

Holman05 Holman, D., Vertegaal, R., Troje, N., “Paperwindows: Interaction techniques for digital paper”, Proceedings of the ACM SIGCHI conference on Human factors in computing systems CHI '05, Portland, OR, USA, pp. 591-599.

Holman08 Holman, D., Vertegaal, R., “Organic user interfaces: designing computers in any way, shape or form”, Communications of the ACM, Vol. 51, No. 6, pp. 48-55, June 2008.

Holsapple02 Holsapple, C.W., Joshi, K. D., "A collaborative approach to ontology design", Comm. of the ACM,Vol. 45, No. 2, February 2002, pp. 42-47.

Hong07 Hong, D., Dickson, C., Shen, V., Cheung, S. C., Kafeza, E., “Ubiquitous enterprise service adaptations based on contextual user behavior”, Information Systems Frontiers , Vol. 9 , No. 4, Kluwer Academic Publishers, September 2007, pp. 343–358.

Hong09 Hong, J., Suh, E., Kim, S., “Context-aware systems: A literature review and classification”, Expert Systems with Applications 36 (2009) 8509–8522.

ICF10 WORLD HEALTH ORGANIZATION, International Classification of Functioning, Disability and Health (ICF). WHO Press, Geneva, Switzerland
http://www.who.int/classifications/icf/icfapptraining/en/index.html

IEEE00 IEEE Standard 1471-2000, IEEE standard office, PO 1331, Piscataway, NJ 08855- 1331, http://standards.ieee.org , 2000.

Ishii08 Ishii, H., “The Tangible UserInterface and It’s Evolution”, Communications of the ACM, Vol. 51, No. 6, June 2008, pp. 32-37

Jacob86 Jacob, R. J. K.,”A specification language for direct-manipulation user interfaces”, ACM Trans. Graph. vol. 5, no. 4 (Oct. 1986),pp. 283–317.

Jacob91 Jacob, R.J.K., “The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At is What You Get”, ACM Transactions on Information Systems, Vol. 9, No. 2, April 1991, pp. 152-169.

Jacobson99 Jacobson, I., Booch, G., Rumbaugh, J., “The unified software development process”, Addison-Wesley object technology series, Reading, Mass: Addison-Wesley, 1999.

Jaquero09 Jaquero, V., Montero, F., Real, F., “Designing user interface adaptation rules with T: XML”, Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI’09, Sanibel Island, Florida, USA February 08 - 11, 2009, pp. 383-388.

John96 John, B.E., Kieras, D.E., “Using GOMS for user interface design and evaluation: which technique?”, ACM Transactions on Computer-Human Interaction, Vol. 3, No. 4, December 1996, pp. 287 – 319.

Jouault05 Jouault, F., Kurtev, I., “Transforming Models with ATL”, Proceedings of MoDELS, LNCS, vol. 3844, 2005, pp. 128–138.

Jouault08 Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., “ATL: A model transformation tool”, Science of Computer Programming 72 (2008), Elsevier Publishing, pp. 31–39.

Jovanović07 Jovanović, M., Starčević, D., Obrenović, Ž., “Improving Aircraft Cockpit Environment Using Multimodal User Interfaces”, In Extended Proceedings of the 12th International Conference on Human-Computer Interaction, HCI International 2007, Springer-Verlag LNCS, Beijing, P.R. China, 22-27. July 2007.

Jovanović08a Jovanović, M., Starčević, D., Minović, M., Štavljanin, V., “Surviving the Design of Educational Games: Borrowing from Motivation and Multimodal Interaction”, In Proceedings of the IEEE Conference on Human-System Interaction, HSI’08, May 25-27, 2008, Krakow, Poland.

Jovanović08b Jovanović, M., Starčević, D., Minović, M., Štavljanin, V., "Educational Games Design Issues: Motivation and Multimodal Interaction", WSKS 2008, Springer-Verlag, Springer-Verlag LNAI 5288, pp. 215–224.

Jovanović09a Jovanović, M., Starčević, D., Obrenović, Ž., "Designing Aircraft Cockpit Displays: Borrowing from Multimodal User Interfaces", Transactions on Computational Science III, Springer-Verlag LNCS 5300, pp. 55-65, 2009.

Jovanović09b Jovanović Mlađan, Softverska Podrška Zemaljskoj Kontrolnoj Stanici za Bespilotnu Letelicu, Magistarski rad, Elektrotehnički Fakultet u Beogradu, Beograd, April 2009.

Jovanović10 Jovanović, M., Starčević, D., Jovanović, Z., "Improving Design of Ground Control Station for Unmanned Aerial Vehicle: Borrowing from Design Patterns", 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010), IEEE Computer Society, Lille, France, Sep. 2010, pp. 65-73.

Jovanović11 Jovanović, M., Starčević, D., Minović, M., Štavljanin, V., "Motivation and Multimodal Interaction in Model-Driven Educational Games Design", IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, Vol. 40, No. 4, pp. 817-824.

Jovanović12 Jovanović, M., Starčević, D., Jovanović, Z., "Formal Specification of Usability Measures in Model-driven Development of Context-sensitive User Interfaces", In Proceedings of the 11th ACM International Working Conference on Advanced Visual Interfaces (AVI2012), ACM Press, Island of Capri (Naples), Italy, May 21-25.

Kamisaka09 Kamisaka, D., Muramatsu, S., Yokoyama, H., Iwamoto, T., "Operation Prediction for Context-Aware User Interfaces of Mobile Phones", Proceedings of the 2009 Ninth Annual International Symposium on Applications and the Internet SAINT '09, IEEE Computer Society.

Kawsar10 Kawsar, F., Fujinami, K., Nakajima, T., Park, J.H., Yeo, S., "A portable toolkit for supporting end-user personalization and control in context-aware applications", Multimedia Tools and Applications , Vol. 47 , No. 3, Kluwer Academic Publishers, May 2010, pp. 409–432.

Kleek10 Kleek, M., Moore, B., Karger, D., André, P., Schraefel, M., "Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web", Proceedings of the 19th international conference on World wide web, WWW '10, April 26–30, 2010, Raleigh, North Carolina, USA, pp. 951-960.

Kleppe03 Kleppe, A., Warmer, S., Bast, W., "MDA explained: The model-driven architecture: Practice and promise", p. 192. Addison-Wesley, Reading (2003)

Klug05 Klug, T., Kangasharju, J., "Executable task models", Proceedings of the 4th international workshop on Task models and diagrams TAMODIA '05, September 26–27, 2005, Gdansk, Poland, pp. 119-122.

Kobsa07 Kobsa, A., "Generic User Modeling Systems", In P. Brusilovsky, A. Kobsa, and W. Nejdl (Eds.): The Adaptive Web, LNCS 4321, 2007.

Korhonen07 Korhonen, J., Ojala, T., Ristola, A., Kesti, M., Kilpelänaho, V., Koskinen, M., Viippola, E., "Mobile Fair Diary: hybrid interface for taking, browsing and sharing context-aware notes", Personal and Ubiquitous Computing , Vol. 11, No. 7, October 2007, pp. 577–589, Springer, London, UK.

Krasner88 Krasner, G.E., Pope, S.T., "A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80". JOOP, vol. 1, no. 3, pp. 26-49, 1988.

Kulkarni10 Kulkarni, D., Tripathi, A., "A Framework for Programming Robust Context-Aware Applications", IEEE Transactions on Software Engineering, Vol. 36, No. 2, pp. 184-197, March/April 2010.

Larson06 Larson, J., *Common Sense Suggestions for Developing Multimodal User Interafces*, W3C Working Group Note 11 September, 2006, http://www.w3.org/TR/mmi-suggestions

Lee08 Lee, J., Seo, D., Rhee, G., "Visualization and interaction of pervasive services using context-aware augmented reality", Expert Systems with Applications: An International Journal , Vol.35, No. 4, Elsevier Press Inc., November 2008, pp. 1873–1882.

Lee09 Lee, Y., Shin, C., Woo, W., "Context-Aware Cognitive Agent Architecture for Ambient User Interfaces", Proceedings of the 13th International Conference on Human-Computer Interaction, HCI09, LNCS 5612, pp. 456–463, 2009.

Leveque09 Leveque, T., Estublier, J., Vega, G., “Extensibility and Modularity for Model Driven Engineering Environments”, 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 305-314.

Limbourg04 Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V., “UsiXML: A language supporting multi-path development of user interfaces”, Proceedings of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th International Workshop on Design, Specification, and Verification of Interactive Systems (EHCIDSVIS’ 04), pp. 200–220.

Lohmann06 Lohmann, S., Kaltz, W., Ziegler, J., “Model-driven dynamic generation of context-adaptive web user interfaces”, Proceedings of the 2006 international conference on Models in software engineering MoDELS'06

Lu09 Lu, J., Zhou, M., “An interactive, smart notepad for context-sensitive information seeking”, Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI’09, Sanibel Island, Florida, USA February 08 - 11, 2009, pp. 127-136.

Maalej09 Maalej, W., Happel, H., Rashid, A., “When users become collaborators: towards continuous and context-aware user input”, Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications OOPSLA '09, October 25–29, 2009, Orlando, Florida, USA, pp. 981-989.

MacKenzie92 MacKenzie, I. Scott, "Fitts' law as a research and design tool in human computer interaction", Human -Computer Interaction 7, (1992), pp. 91-139

Medina07 Medina, J. L. P., Chessa, S. D., Front , A., “A Survey of Model Driven Engineering Tools for User Interface Design”, Proceedings of the 6th international conference on Task models and diagrams for users interface design TAMODIA’07, LNCS 4849, pp. 84 – 97.

Meera96 Meera, M. B., Glinter, E.P., "Multimodal Integration", IEEE Multimedia, Winter 1996, pp. 14-24

Michotte08 Michotte, B., Vanderdonckt, J., “GrafiXML, a Multi-target User Interface Builder Based on UsiXML”, Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems ICAS '08, pp. 15-22, March 2008.

Microsoft06 Microsoft. 2006. XAML. http://windowssdk.msdn.microsoft.com/en-us/library/ms747122.aspx

Milanovic09 Milanovic, M., Gasevic, D., Giurca, A., Wagner, G., Devedzic, V., “Bridging concrete and abstract syntaxes in model-driven engineering: a case of rule languages”, Softw. Pract. Exper. 2009; 39:1313–1346.

Millán03 Millán, J., “Adaptive Brain Interfaces”, Communications of the ACM, Vol. 46, No. 3, March 2003, pp. 75-80.

Miluzzo10 Miluzzo, E., Wang, T., Campbell, A.T., “EyePhone: activating mobile phones with your eyes”, Proceedings of the 2nd ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds, New Delhi, India, August 30, 2010.

Moguffin05 Mcguffin, M.J., Balakrishnan, R., “Fitts’ Law and Expanding Targets: Experimental Studies and Designs for User Interfaces”, " ACM Transactions on Computer-Human Interaction, Vol. 12, No. 4, December 2005, Pages 388-422.

Mori02 Mori, G., Paterno, F., Santoro, C., “CTTE: Support for Developing and Analyzing Task Models for Interactive System Design”, IEEE Transactions on Software Engineering, Vol. 28, No. 8, August 2002, pp. 797-813.

Mori04 Mori, G., Paterno, F., Santoro, C., “Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions”, IEEE Transactions on Software Engineering, Vol. 30, No. 8, August 2004, pp. 507-520.

Mühlhäuser09 Mühlhäuser, M., Hartmann, M., “Interacting with context”, Proceedings of the 1st international conference on Quality of context QuaCon'09, LNCS 5786, pp. 1–14, 2009.

Myers00 Myers, B., Hudson, S., Pausch, R., “Past, present, and future of user interface software tools”, ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1, March 2000, Pages 3–28.

Myers98 Myers, B., "A Brief History of Human-Computer Interaction Technology", Interactions, ACM Press, March-April 1998, pp. 44-54.

Newman68 Newman, W.M. “A System for Interactive Graphical Programming.” Proceedings of AFIPS Spring Joint Computer Conference 28 (1968), pp. 47–54.

Nichols09 Nichols, J., Myers, B., “Creating a Lightweight User Interface Description Language: An Overview and Analysis of the Personal Universal Controller Project”, ACM Transactions on Computer-Human Interaction, Vol. 16, No. 4, Article 17, 37 pages, November 2009

Nunes00a Nunes, N.J., Cuhna, J.F., “Wisdom: a software engineering method for small companies”, IEEE Software, Vol. 17, no. 5, pp. 113-119, Sep/Oct 2000.

Nunes00b Nunes, N.J.., Cuhna, J.F.,”Towards a UML profile for interaction design: the Wisdom approach”, A. Evans, S. Kent and B. Selic (Eds.): UML 2000, LNCS 1939, pp. 101-126, 2000.

Obrenovic04 Obrenovic, Z., Starcevic, D., “Modeling Multimodal Human-Computer Interaction”, IEEE Multimedia, Vol. 11, No. 1, January-March 2004, pp. 65-72

Oviatt00 Oviatt, S., Cohen, P., “Mutlimodal systems that process what comes naturally”, Communications of the ACM, vol. 43, no. 3, pp. 45-53 (2000).

Oviatt06 Oviatt, S., “Human-centered design meets cognitive load theory: designing interfaces that help people think”, Proceedings of the 14th annual ACM international conference on Multimedia MULTIMEDIA '06, October 23–27, 2006, Santa Barbara, California, USA, pp. 871-880.

Park10 Park, J., Kim, K., Jo, S., “A POMDP approach to P300-based brain-computer interfaces”, Proceeding of the 14th ACM international conference on Intelligent user interfaces IUI’10, Hong Kong, China , February 07 - 10, 2010, pp. 1-10.

Pask72 Pask, G.: A fresh look at cognition and the individual. International Journal on the Man-Machine Studies 4 (1972) 211-216

Paterno00 Paterno, F., Mancini, C., “Model-Based Design of Interactive Applications”, ACM Intelligence, Winter 2000, pp. 27-37

Paterno08 Paternò, F., Santoro, C., “UIDLs for Ubiquitous Environments”, ACM CHI 2008 Workshop Proceedings on User Interface Description Languages for Next Generation User Interfaces, April 6tt, 2008, Florence, Italy.

Paterno08b Paterno, F., Santoro, C., Mantyjarvi, J., Mori, G., Sansone, S., “Authoring pervasive multimodal user interfaces”, Int. J. Web Eng. Technol., vol. 4, no.2, 2008, pp. 235–261.

Paterno09 Paterno, F., Santoro, C., Spano, L. D., “MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments”, ACM Transactions on Computer-Human Interaction (TOCHI) Vol. 16, No. 4, Article 19 (November 2009), 30 pages.

Paterno99 Paterno, F., “Model Based Design and Evaluation of Interactive Applications”, Applied Computing, London: Springer-Verlag, 1999.

Pawar08 Pawar, P., Wac, K., Van Beijnum, B.J., Maret, P., Van Halteren, A., Hermens, H., “Context-aware middleware architecture for vertical handover support to multi-homed nomadic mobile services”, Proceedings of the ACM symposium on Applied computing SAC’08, March 16-20, 2008, Fortaleza, Ceará, Brazil, pp. 481-488.

Pentland00 Pentland, A., "Perceptual user interfaces: perceptual intelligence", Communications of the ACM, Vol. 43, No. 3, March 2000, pp. 35 – 44.

Pfaff85 Pfaff, G, Haguen, P.J.W.T., “User Interface Management Systems”, Springer-Verlag: Berlin, 1985.

Piper02 Piper, B., Ratti, C., Ishii, H., “Illuminating clay: A 3-D tangible interface for landscape analysis”, Proceedings of the ACM SIGCHI conference on Human factors in computing systems CHI '02, Mineapolis, MN, USA, pp. 355-362.

Poupyrev07 Poupyrev, I., Nashida, T., Okabe, M., “Actuation and tangible user inrterfaces: The Vaucanson Duck, robots, and shape displays”, Proceedings of the ACM SIGCHI conference on Tangible, Embedded and Embodied Interaction TEI '07, Baton Rouge, CA, USA, pp. 205-217.

Puerta02 Puerta, A., Eisenstein, J., “XIML: A Common Representation for Interaction Data”, Proceedings of the ACM 6th International Conference on Intelligent User Interfaces IUI’02, January 13-16, 2002, San Francisco, California, USA, pp. 214-215.

Puerta09 Puerta, A., Hu, M., “UI Fin: A Process-Oriented Interface Design Tool”, ”, Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI’09, Sanibel Island, Florida, USA February 08 - 11, 2009, pp. 345-354.

Raento05 Raento, M., Oulasvirta, A., Petit, R., Toivonen, H., “ContextPhone: A prototyping platform for contextaware mobile applications,” IEEE Pervasive Computing, Vol. 4, No. 2, pp. 51-59, June 2005.

Rehman07 Rehman, K., Stajano, F., Coulouris, G., “An architecture for interactive context-aware applications”, IEEE Pervasive Computing, Vol. 6, No. 1, March 2007. 73–80.

Rekimoto97 Rekimoto. J., ”Pick-and-Drop: A Direct Manipulation technique for Multiple Computer Environments “, In Proc. UIST97, ACM Publ., pp. 31-39, 1997.

Ribeiro10 Ribeiro, F. R., José, R., “Timely and Keyword-Based Dynamic Content Selection for Public Displays”, Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems, IEEE CS, pp. 665-660.

Rich79 Rich, E., “User Modeling via Stereotypes”, COGNITIVE SCIENCE 3, pp. 329-354, 1979.

Riding98 Riding, R., Rayner, S., “Cognitive Styles and Learning Strategies: Understanding Style Differences in Learning and Behavior”, David Fulton Publisher, London (1998)

Riva07 Riva, O., Nadeem, T., Borcea, C., Iftode, L., “Context-aware migratory services in ad hoc networks”, IEEE Mobile Computing, Vol. 6, No. 12, 2007, pp. 1313–1328.

Samaan05 Samaan, N., Karmouch, A., “A mobility prediction architecture based on contextual knowledge and spatial conceptual maps”, IEEE Transactions on Mobile Computing, vol. 4, no. 6, 2005, pp. 537–551.

Sandor05 Sandor, C., Klinker, G., “A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality”, Personal Ubiquitous Computing (2005) 9: 169–185, Springer-Verlag London.

Savidis10 Savidis, A., Stephanidis, C., ‘Software refactoring process for adaptive user-interface composition”, Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS’10, June 19–23, 2010, Berlin, Germany, pp.19-28.

Schaefer02 Schaefer, R., Mueller, W., Dangberg, A., “RDL/TT, A: Description Language for the Profile-Dependent Transcoding of XML Documents”, Proceedings of the first International ITEA Workshop on Virtual Home Environments, 2002.

Schaefer06 Schaefer, R., Bleul, S., Mueller, W., “Dialog modelling for multiple devices and multiple interaction modalities”, Proceedings of the 5th International Workshop on Task Models and Diagrams for User Interface Design, pp. 39–53.

Scheifler86 Scheifler, R.W., Gettys, J. “The X Window System.” ACM Transactions on Graphics 5, 2 (1986), pp. 79–109.

Schilit94 Schilit, B., Theimer, M., “Disseminating active map information to mobile hosts”, IEEE Network 1994; 8: 22–32.

Schmidt00 Schmidt A, Gellersen H-W, Beigl M, Thate O (2000) Developing user interfaces for wearable computers—don’t stop to point and click. In: Proceedings of the international workshop on interactive applications of mobile computing (IMC 2000), Rostock, Germany, November 2000

Seidewitz03 Seidewitz, E., “What Models Mean”, IEEE Software, Vol. 20, No. 5, September/October 2003, pp. 26-33.

Selic03 Selic, B., “The Pragmatics of Model-Driven Development”, IEEE Software, Vol. 20, No. 5, September/October 2003, pp. 19-25.

Sendall03 Sendall, S., Kozaczynski, W., "Model Transformation The Heart and Soul of Model-Driven Software Development", IEEE Software, Vol. 20, No. 5, September / October 2003, pp. 42-45.

Serral10 Serral, E., Valderas, P., Pelechano, V., “Towards the Model Driven Development of context-aware pervasive systems”, Pervasive and Mobile Computing , Vol. 6, No. 2, Elsevier Science Publishers B. V., April 2010, pp. 254-280.

Shaefer07 Shaefer, R.,"A Survey on Transformation Tools for Model Based User Interface Development", J. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2007, LNCS 4550, pp. 1178-1187, 2007.

Shaer09 Shaer, O., Jacob, R. J. K.,"A specification paradigm for the design and implementation of tangible user interfaces", ACM Transactions on Computer-Human Interaction (TOCHI) Vol. 16, No. 4, Article 20 (November 2009), 39 pages.

Shneiderman00 Shneiderman, B., "Universal Usability", Communications of the ACM, 43(5), 17-22 (2000).

Shneiderman86 Shneiderman, B., 1986. "Seven plus or minus two central issues in human-computer interaction". Proc. CHI'86 Human Factors in Computing Systems

Simon04 Simon, R., Jank, M., Wegscheider, F., "A generic UIML vocabulary for device- and modality independent user interfaces", Proceedings of the World Wide Web Conference *WWW 2004*, May 17–22, 2004, New York, New York, USA, pp. 434–435.

Sinnig10 Sinnig, D., Mizouni, R., Khendek, F., "Bridging the gap: empowering use cases with task models", Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, EICS'10, June 19–23, 2010, Berlin, Germany, pp. 291-296.

Smith77 Smith, D.C. Pygmalion, "A Computer Program to Model and Stimulate Creative Thought", Birkhauser Verlag, Stuttgart, 1977. Originally doctoral dissertation, Computer Science Department, Stanford University, 1975.

Sottet05 Sottet, J-S., Calvary, G., Favre, J-M., Coutaz, J., Demeure, A., Balme, L., "Towards Model Driven Engineering of Plastic User Interfaces", Satellite Proceedings of the ACM/IEEE 8th International Conference on Models Driven Engineering Languages and Systems, MoDELS/UML 2005. LNCS, pp. 191–200.

Sottet07 Sottet, J., Calvary, G., Coutaz, J., Favfre, J,. "A model-driven engineering approach for the usability of plastic user interfaces", Proceedings of the Working Conference on Engineering Interactive Systems, EIS 2007, LNCS 4940, pp. 140–157.

Stanciulescu08 Stanciulescu, A., Vanderdonckt, J., Mens, T., "Colored graph transformation rules for model-driven engineering of multi-target systems", Proceedings of the third ACM international workshop on Graph and model transformations GRaMoT '08, May 12, 2008, Leipzig, Germany, pp. 37-44.

Sutherland63 Sutherland, I.E. "SketchPad: A Man-Machine Graphical Communication System." Proceedings of AFIPS Spring Joint Computer, Conference 23 (1963), pp. 329–346.

TourDeFlex10 TourDeFlex, 2010,

Turk00 Turk, M., Robertson, G., "Perceptual user interfaces (introduction)", Communications of the ACM, Vol. 43, No. 3, March 2000, pp. 33-35.

UML10 OMG Unified Modeling Language™ (OMG UML), Infrastructure Version 2.3, May 2010 http://www.omg.org/spec/UML/2.3/Infrastructure

Vale09 Vale, S., Hammoudi, S., "COMODE: A Framework for the Development of Context-Aware Applications in the Context of MDE", Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services, ICIW '09, pp. 261-266.

Vanderdonckt08 Vanderdonckt, J., Calvary, G., Coutaz, J., Multimodality for Plastic User Interfaces: Models, Methods and Principles, D. Tzovaras (Ed.) Multimodal User Interfaces. Signals and Communication Technology, Springer-Verlag Berlin Heidelberg 2008, DOI: 10.1007/978-3-540-78345-9, pp.75-101.

Vertegaal03 Vertegaal, R., "Attentive User Interfaces", Communications of the ACM, Vol. 46, No. 3, March 2003, pp. 31-33.

Weiser94 Weiser, M., "The world is not a desktop", ACM Interactions 1(1): 7-8, 1994.

Wexelblat78 Wexelblat, R.L. (Ed.), 1978. Proceedings of ACMSIGPLAN History of Programming LanguagesConference. SIGPLAN Notices, 13, 8.

Witkin77 Witkin, H.A., Moore, C.A., Goodenough, D.R., Cox, P.W.: Field-dependent and fieldindependent cognitive styles and their educational implications. Review of Educational Research 47, 1 (1977) 1-64

Wobbrock11 Wobbrock, J. O., Kane, S. K., Gajos, K. Z., Harada, S., and Froehlich, J., “Ability-based design: Concept, principles and examples”, ACM Trans. Access. Comput. 3, 3, Article 9 (April 2011), 27 pages.

Wolff09 Wolff, A., Forbrig, P., “Deriving User Interfaces from Task Models”, Proceedings of the IUI'09 Workshop on Model Driven Development of Advanced User Interfaces MDDAUI '09, 2009.

Wolff10 Wolff, A., Forbrig, P., “Model-driven User Interface Development with the Eclipse Modeling Project”, Proceedings of the ACM CHI 2010 5th International Workshop on Model Driven Development of Advanced User Interfaces MDDAUI’10, 2010.

Wong09 Lin, J., Wong, J., Nichols, J., Cypher, A., Lau, T., “End-user programming of mashups with vegemite”, Proceedings of the 13th ACM international conference on Intelligent user interfaces IUI’09, Sanibel Island, Florida, USA February 08 - 11, 2009, pp. 97-106.

Yu06 Yu, Z., Zhou, X., Zhang, D., Chin, C., Wang, X., Men, J., “Supporting contextaware media recommendations for smart phones”, IEEE Pervasive Computing, vol. 5, no. 3, 2006, pp.68–75

Zhou09 Zhou, L., Xiong, N., Shu, L., Vasilakos, A., Yeo, S., “Context-Aware Middleware for Multimedia Services in Heterogeneous Networks”, IEEE Intelligent Systems, 2009.

Spisak slika i tabela

Biografija autora

Mlađan Jovanović je rođen 25. maja 1981. godine u Novom Sadu. Diplomirao je 2005. godine na Vojnotehničkoj Akademiji, smer Službe informatike, sa prosečnom ocenom studija 9.19 (konačna ocena 10). Nakon završetka studija radi u Centru za komandno-informacione sisteme i informatičku podršku Vojske Srbije na poziciji razvojnog inženjera. Magistarske studije na Elektrotehničkom fakultetu je upisao 2005. godine. Studije je završio sa prosečnom ocenom 10. Magistrirao je 2009. godine sa temom "Softverska podrška zemaljskoj kontrolnoj stanici za bespilotnu letelicu". Angažovan je i kao spoljni saradnik na Fakultetu Organizacionih Nauka gde izvodi vežbe iz predmeta Multimediji, Interakcija Čoveka i Računara, Multimedijalne komunikacije i Multimedijalne baze podataka. Mlađan Jovanović je recenzent na međunarodnim konferencijama ACM Conference on Human Factors in Computing Systems, ACM Designing Interactive Systems i ACM International Conference on Intelligent User Interfaces. Pored toga, vrši recenziju radova i za časopise IEEE Computer i IEEE Transactions on Systems, Man and Cybernetics (Part A, B, C). Autor je većeg broja radova na domaćim i međunarodnim konferencijama i časopisima.

Прилог 1.

Изјава о ауторству

Потписани-а Милан Јовановић

број уписа /

Изјављујем

да је докторска дисертација под насловом

Развој контекстно-осетљивих корисничких интерфејса

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 03.09.2012.

Милан Јовановић

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Мирјан Јовановић

Број уписа /

Студијски програм /

Наслов рада Развој контекстно-осетљивих корисничких интерфејса

Ментор Проф. др Зоран Јовановић

Потписани Мирјан Јовановић

изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 03.09.2012.

Мирјан Јовановић

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић" да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Развој контекстно-осетљивих мрежних интерфејса

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

(4.) Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 03.09.2012.

Млађан Јовановић

1. Ауторство - Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. Ауторство – некомерцијално. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. Ауторство - некомерцијално – без прераде. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. Ауторство - некомерцијално – делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. Ауторство – без прераде. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. Ауторство - делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.