RĪGAS TEHNISKĀ UNIVERSITĀTE

**Kristaps Babris**

# DIVPUSLOŽU MODEĻA TRANSFORMĀCIJĀ SAKŅOTA TĪMEKĻA LIETOTNES LIETOTĀJA SASKARNES PROTOTIPA ĢENERĒŠANA

Promocijas darbs


RTU Izdevniecība
Rīga 2025

**RĪGAS TEHNISKĀ UNIVERSITĀTE**
Datorzinātnes, informācijas tehnoloģijas un enerģētikas fakultāte

**Kristaps Babris**
doktora studiju programmas „Datorzinātne un informācijas tehnoloģija”
doktorants

# DIVPUSLOŽU MODEĻA TRANSFORMĀCIJĀ SAKŅOTA TĪMEKĻA LIETOTNES LIETOTĀJA SASKARNES PROTOTIPA ĢENERĒŠANA

**Promocijas darbs**

Zinātniskā vadītāja
profesore *Dr. sc. ing.*
OKSANA ŅIKIFOROVA

Rīga 2025

# ANOTĀCIJA

Atslēgvārdi: *Lietotāja saskarne, modeļvadāma izstrāde, divpusložu modelis, lietotāja saskarnes šabloni.*

Mūsdienu tirgus tendences un lietotāju prasības liecina, ka programmatūras inženierijas jomā lietotāja saskarnei ir jābūt ne tikai funkcionālai, bet arī intuitīvai, estētiski pievilcīgai un lietotājam draudzīgai. Arvien biežāk tiek uzsvērts, ka tieši lietotāja saskarnes uzlabošanā un pielāgošanā ir jāiegulda resursi, lai panāktu konkurētspēju un lietotāju apmierinātību. Lietotāja saskarnes kvalitāte un izstrādes ātrums var būt izšķiroša programmatūras risinājuma pieņemšanā tirgū. Īpaši tas viss ir svarīgs tīmekļa lietotņu izstrādes kontekstā, kas ir kļuvušas par būtisku uzņēmumu komunikācijas un pakalpojumu sniegšanas veidu, un lietotāja saskarne ir viens no galvenajiem faktoriem, kas nosaka lietotnes pieņemšanu un lietotāju apmierinātību. Ātri un pie tam kvalitatīvi izstrādāta tīmekļa lietotnes saskarne var ievērojami uzlabot lietošanas ērtumu, palielināt produktivitāti un nodrošināt pozitīvu lietotāju pieredzi, kas ir būtiska konkurētspējas uzturēšanai.

Viens no programmatūras inženierijas risinājumiem, kas dod iespēju paātrināt izstrādi, nezaudējot kvalitāti, ir izstrādes automatizācija. Arī lietotāja saskarnes automātiskā ģenerēšana ievērojami samazina laiku, kas nepieciešams programmatūras izstrādei. Turklāt, automātiskā ģenerēšana nodrošina vienotu izskatu un sajūtu visā lietotnē, jo visi saskarnes elementi tiek veidoti pēc vieniem un tiem pašiem standartiem. Tas samazina cilvēka kļūdu iespējamību un nodrošina augstāku kvalitāti. Papildus tam, izmantojot tipveida tīmekļa priekšgala komponentus, var vieglāk nodrošināt lietotnes atbilstību dizaina vadlīnijām un lietojamības principiem.

Promocijas darbā ir piedāvāts risinājums tīmekļa lietotnes lietotāja saskarnes prototipa pirmkoda ģenerēšanai no divpusložu modeļa, izmantojot modeļvadāmas programmatūras inženierijas principus. Modeļu transformācijas likumu kopas izstrāde balstās divpusložu modeļa, kā avota modeļa, metamodelī un lietotāja saskarnei definētajā sagaidāmo elementu kopas metamodelī, kas satur visus lietotāja saskarnes slāņus: konceptu, navigāciju un prezentāciju.

Promocijas darbs ietver ievadu, četras nodaļas un secinājumus. Tajā ir 196 lapaspuses, 76 attēli, 45 tabulas un 253 bibliogrāfijas avoti.

# ABSTRACT

Keywords: *User Interface, Model-Driven Development, Two-Hemisphere Model, User Interface Patterns.*

Current market's trends and user's needs suggest that in the software engineering field, it is important for the user interface (UI) to have functionality along with an intuitive nature, visually appealing aspects and easy operation. They stress on investing resources into enhancing or adjusting UI for staying competitive as well as keeping users satisfied – UI superiority combined with fast development is key factor determining how well a software solution will be received by market. This point is very important in web application development because it has become a considerable way for companies to communicate and provide services, where the user interface plays an essential role in deciding acceptance of the application and satisfaction of users. A quickly developed interface for a web application with high quality can make it more usable, productive and help secure good user experience which is crucial to remain competitive.

One of the software engineering solutions that lets to speed up development without risking quality is development automation; this includes automatically generating the user interface. This also notably decreases the time needed for software development by creating an automated generation of UIs. Another advantage of development automation is application consistency – every interface element will adhere to identical standards for creation. This minimizes chances for human mistakes and guarantees better quality control. Moreover, using predefined web application front-end components can more easily ensure the application's compliance with design guidelines and usability principles. The PhD thesis proposes a solution for generating a source code of web application user interface prototype from a two-hemisphere model using model-driven software engineering principles. The development of a set of model transformation rules is based on the metamodel of the two-hemisphere model as a source model and the metamodel of the expected elements defined for the user interface, which include all user interface layers: concept, navigation, and presentation.

The thesis includes an introduction, four chapters, and conclusion. It contains 196 pages, 76 figures, 45 tables and 253 bibliography sources.

# SATURA RĀDĪTĀJS

# IEVADS

Programmatūras izstrāde kā inženierijas nozare ir samēra jaunā, salīdzinot ar būvniecību, medicīnu, matemātiku, fiziku, utml., tai kopš 1969.gadā konstatētas programmēšanas krīzei (Aspray, Keil et al., 1999) un dibināšanai par inženierijas nozari promocijas darba izstrādes noslēguma gadā ir 55 gadi. Šajā laikā ir veikti vairāki pētījumi par programmatūras izstrādes metodoloģiju, realizācijas risinājumu, tehnoloģiju un arhitektūras veidu attīstīšanā. Mūsdienās ir nostabilizējies efektīvas programmatūras izstrādes pamatkoncepciju kopa, kas ir spējās metodoloģijas (Calvary, Coutaz et al., 2003; Al-Saqqa, Sawalha et al., 2020), automatizācija (Narang & Mittal, 2022; Yigitbas, Jovanovikj et al., 2020), modulāra un elastīga arhitektūra (Akiki, Bandara et al., 2014; Mbuga, Korongo et al., 2022), satvaru izmantošana, efektīva testēšana un ieguldījumi izstrādātāju rīkos (Akiki, Bandara et al., 2014; Mbuga, Korongo et al., 2022). Automatizācijas ieviešana atkārtotiem uzdevumiem, piemēram, testēšanai, būvēšanai un izvietošanai, ievērojami paātrina izstrādes procesu (Nikiforova, Babris et al., 2021).

Mūsdienu programmatūras risinājumi tiek veidoti no trīs aspektiem:

1) Biznesa loģikas komponenti, kuros ir realizēti lietojumprogrammas darbības noteikumi un procesu un kas veido sistēmas pamatu;
2) Dati nodrošina informāciju, kas ir būtiska gan analīzei, gan operatīvai darbībai, ļaujot pieņemt pamatotus lēmumus un uzlabot uzņēmuma procesus.
3) Lietotāja saskarne nodrošina lietotāju pieredzi un mijiedarbību ar sistēmu.

Un tieši lietotāja saskarne izceļas kā programmatūras kritisks komponents, jo darbojas kā tilts starp lietotāju un programmatūru, padarot sistēmas realizācijas iespējas pieejamas un izmantojamas gala lietotājiem. Lietotāja saskarne (angl. *User Interface, UI*) ir jebkura lietojumprogrammas galvenā sastāvdaļa (Nguyen, Vu et al., 2018), un tai ir izšķiroša nozīme lietojumprogrammas mijiedarbībai ar lietotāju. Tomēr lietotāja saskarne nav neatkarīga no tā lietošanas konteksta, kas definēts kā lietotājs, platforma un vide (Calvary, Coutaz et al., 2003; Yigitbas, Jovanovikj et al., 2020). Faktiski, lietojumprogrammas biznesa loģika var būt viena, bet lietotāja saskarne var būt realizēta un piemērota katrai atsevišķai platformai (mobilās ierīces, dažādas operētājsistēmas, utt.). Turklāt, visaptveroša dažādu jomu digitalizācija izvirza paaugstinātas prasības informācijas sistēmu veidošanas ātrumam, kad vairs nepastāv problēma

piedāvāt programmatūras inženierijas risinājumus jebkurā jomā, taču galvenais uzdevums ir piedāvāt to ātrāk par konkurentiem. Un tajā pašā laikā mūsdienu lietotāja saskarnes kļūst aizvien sarežģītākas, dēļ nepieciešamības atbalstīt dažādus heterogēnus pielietošanas kontekstus, jo vairs nav efektīvi nodrošināt tikai vienu, visiem gadījumiem, pielietojamu lietotāja saskarni. Tomēr, regulārā šāda kontekstu jeb platformu izmaiņu veikšana manuāli rada nozīmīgas izmaksas, variācijas un sarežģīti pārvaldāmus projektus (Pierre, Arosha et al, 2014).

**Risināšanai izvēlēta problēma**

Problēma, kas ir izvirzīta risināšanai šajā promocijas darbā ir saistīta ar nepieciešamību piedāvāt lietotāja saskarnes komponentu izstrādes pieeju, kas nodrošinātu automatizētu, un līdz ar to ātru, lietotāju saskarnes komponentu iegūšanu no zināšanām par problēmas vidi, kas, turklāt būs neatkarīgas no platformas un realizācijas detaļām, tas ir nodrošināta pietiekami augsta problēmvides zināšanu abstrakcija. Par šādām prasībām atbilstošu risinājumu var kalpot modeļvadāmā izstrāde (angl. *Model-Driven Development, MDD*), kas paredz, ka problēmvides zināšanas ir atspoguļotas modeļa veidā no platformas neatkarīgajā līmenī, un ir nodrošināts formālisms šī modeļa transformācijai platformai specifiskajā līmenī. Modeļvadamā izstrāde sāka attīstīties kopš 2001. gada (Lycett, Marcos et al., 2007) ar ideju pārveidot programmatūras izstrādes procesu par pilnīgi automatizētu, bet neskatoties uz acīmredzamajām modeļvadāmas izstrādes priekšrocībām (piemēram, modelējot var ieraudzīt sistēmas kopskatu un sākotnējo reprezentāciju, veikt tajā labojumus, panākot pilnīgu atbilstību problēmvides zināšanām, un tikai tad papildināt sistēmas modeli ar platformas detaļām), šīs izstrādes pieejas ieviešana līdz visu programmatūras izstrādes aspektu automatizācijai nav nonākusi.

Mūsdienās modeļvadāma izstrāde nodrošina sistēmas statisku artefaktu (piemēram klašu definīcijas no ER diagrammām) ģenerēšanu, kamēr pie dinamiskiem elementiem ir jāpiestrādā. Tas pats attiecas arī uz lietotāja saskarni, kur kvalitatīvi izveidotam modelim var daļēji uzģenerēt formas un to saturu, bet risinājumi pilnvērtīgus lietotāja saskarnes prototipus, kas būtu gatavi nodošanai realizācijā pašlaik vēl nav izstrādāti un ieviesti praksē.

Pēdējā laikā sāk attīstīties mēģinājumi lietot ģeneratīvo mākslīgo intelektu (Stige, Zamani et al., 2023; Silva, Martin et al., 2011) arī lietotāju saskarnes elementu ģenerēšanai. Lai gan ģeneratīvie rīki ir izcili dažos radošos uzdevumos, lietotāja saskarnes dizaina niansētais un ļoti

precīzais raksturs tiem rada grūtības. Ģenerētie lietotāja saskarnes varianti ir mākslīgā intelekta algoritmu lietošanas rezultāta "fantāzijas", kam vajag ļoti precīzi definēt prasības un zināšanu bāzi ar zināšanu likumiem, kas pēc būtības jau ir topošās lietotāja saskarnes skices un prasa specifiskās prasmes, lai darbotos ar ģeneratīvā mākslīgā intelekta rīkiem (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020). Tā rezultātā ģenerētā lietotāja skiču apraksta definēšana (Kompaniets, Lyz et al., 2020; Johnson, Gross et al., 2009) un iegūta rezultāta verificēšana prasa pārmērīgus resursu to veidošanai, testēšanai un atkļūdošanai (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020). Cilvēku dizaineri pašlaik joprojām ir būtiski, lai nodrošinātu precizitāti, funkcionalitāti un uz lietotāju vērstus aspektus, kas ir būtiski efektīvai lietotāja saskarnes izstrādei (Newmanm & Landay, 2000; Li, Cao et al., 2023) un lietotāja saskarnes skiču veidošana joprojām ir izaicinājums dizaineriem, lai pārvarētu plaisu starp koncepciju un radīšanu (Nikiforova, Zabiniako et al., 2021b).

Savā veidā ģeneratīva mākslīga intelekta izmantošanu lietotāja saskarnes ģenerēšanai arī var saukt par modeļos sakņotu lietotāja saskarnes veidošanu, jo ģeneratīvā mākslīga intelekta algoritmi par avota modeli izmanto zināšanas par problēmas vidi un rezultātā piedāvā variantus lietotāja saskarnes skicēm kā mērķa modeli (Sharp, Rogers et al., 2019; Planas, Daniel et al., 2021). Problēmvides modeļu izveide un transformācija, kas balstīta uz metamodelēšanas principiem, var dot iespēju realizēt lietotāja saskarnes satura automātisku ģenerēšanu no problēmvides apraksta, ja šis apraksts ir izveidots formāli (Joo, 2017; Beek & McIver, 2021), piemēram, ar modeli kas ir pilnīgs un konsekvents. Lai gala programmatūras produkts būtu kvalitatīvs, sākotnēji ir jāizveido kvalitatīvi modeļi, kas satur pilnīgu un visaptverošu informāciju par problēmvidi (Osis & Asnina, 2011).

**Pētījuma objekts, priekšmets un hipotēze**

**Pētījuma objekts** šajā promocijas darbā ir tīmekļa lietotnes lietotāja saskarne, jo īpaši tīmekļa lietotnes kontekstā, kur lietotāji sagaida ātru, intuitīvu un patīkamu pieredzi, saskarnes nozīme ir vēl lielāka. un **pētījuma priekšmets** ir lietotāja saskarnes prototipa automatizēta ģenerēšana, kur lietotāja saskarne ir veidota no problēmvides zināšanām, kas ir attēlotas modeļa veidā un satur pilnīgu un nepretrunīgu informāciju par topošas programmatūras lietošanas loģiku. Ir plānots, ka promocijas darbā piedāvātā risinājuma rezultātā no problēmvides modeļa ģenerētie

lietotāja saskarnes prototipi ir gan pietiekami vienkārši, gan atkārtoti lietojami un, balstoties modeļvadāmās izstrādes principos (Beltramelli, 2018; Hussmann, Meixner et al., 2011) ir iespējams ģenerēt jaunus prototipus, veicot izmaiņas problēmvides modelī. Lai šāds risinājums darbotos pareizi, ir nepieciešams izvēlēties notāciju avota modelim, kurš satur pilnīgas un nepretrunīgas zināšanas par problēmvides loģiku. Un, zinot sagaidāma mērķa modeļa saturu un notāciju, ir iespējams definēt transformācijas likumus, kas sakņosies avota un mērķa modeļa metamodeļos. Par avota modeli risinājuma izstrādē ir izvēlēts divpusložu modelis (Nikiforova, Kozacenko et al., 2015), ka pilnīgums, nepretrunīgums un lietojamība jau ir pierādīti iepriekšējos pētījumos un publikācijas (Nikiforova & Pavlova, 2011; Bajovs, Nikiforova et al., 2013; Kozačenko, 2014; Nikiforova & Gusarovs, 2020; Gusarovs, 2020) dažādu programmatūras artefaktu izgūšanā. (Gusarovs, 2020) ir pierādīts, ka:

1) divpusložu modelis var kalpot ka modelis, kas satur pietiekamu informāciju problēmvides atbalsta programmatūras sistēmas ģenerēšanai,
2) biznesa procesu modelēšana ir viena no ierastajām programmatūras inženierijas aktivitātēm, līdz ar to nepieciešamība veidot divpusložu modeli, kas sastāv no biznesa procesa modeļa un atbilstošā konceptu modeļa, neprasa papildus resursus un izmaiņas tradicionālajā izstrādē.

Attiecīgi darbā ir izvirzīta **hipotēze**, ka, ja biznesa loģiku atbalstošo pirmkodu var ģenerēt no divpusložu modeļa (Gusarovs, 2020), tad no divpusložu modeļa ir iespējams ģenerēt arī lietotāja saskarnes prototipu, kas pēc būtības ir biznesa loģikas darbības rezultāta attēlojums informācijas sistēmas lietotāja pusē. Tādējādi, promocijas darba ietvaros izstrādātais risinājums būtu domāts tikai tīmekļa lietojumiem, kuriem ir iespējams izšķirt netriviālus biznesa procesus un datu entitījas.

**Aizstāvēšanai izvirzītās tēzes**

Promocijas darba ietvaros aizstāvēšanai ir izvirzītas divas tēzes:

1) No RTU izstrādātā divpusložu modeļa ir iespējams ģenerēt tīmekļa lietotnes lietotāja saskarnes prototipus, izmantojot modeļvadāmas izstrādes pamatkoncepcijas, kas ir modeļi un modeļu transformācijas.
2) Tīmekļa lietotnes lietotāja saskarnes prototipa automātiskā iegūšana no biznesa līmeņa modeļiem paātrina tīmekļa lietotnes izstrādes procesu, nezaudējot rezultāta kvalitāti.

**Promocijas darba mērķis un uzdevumi**

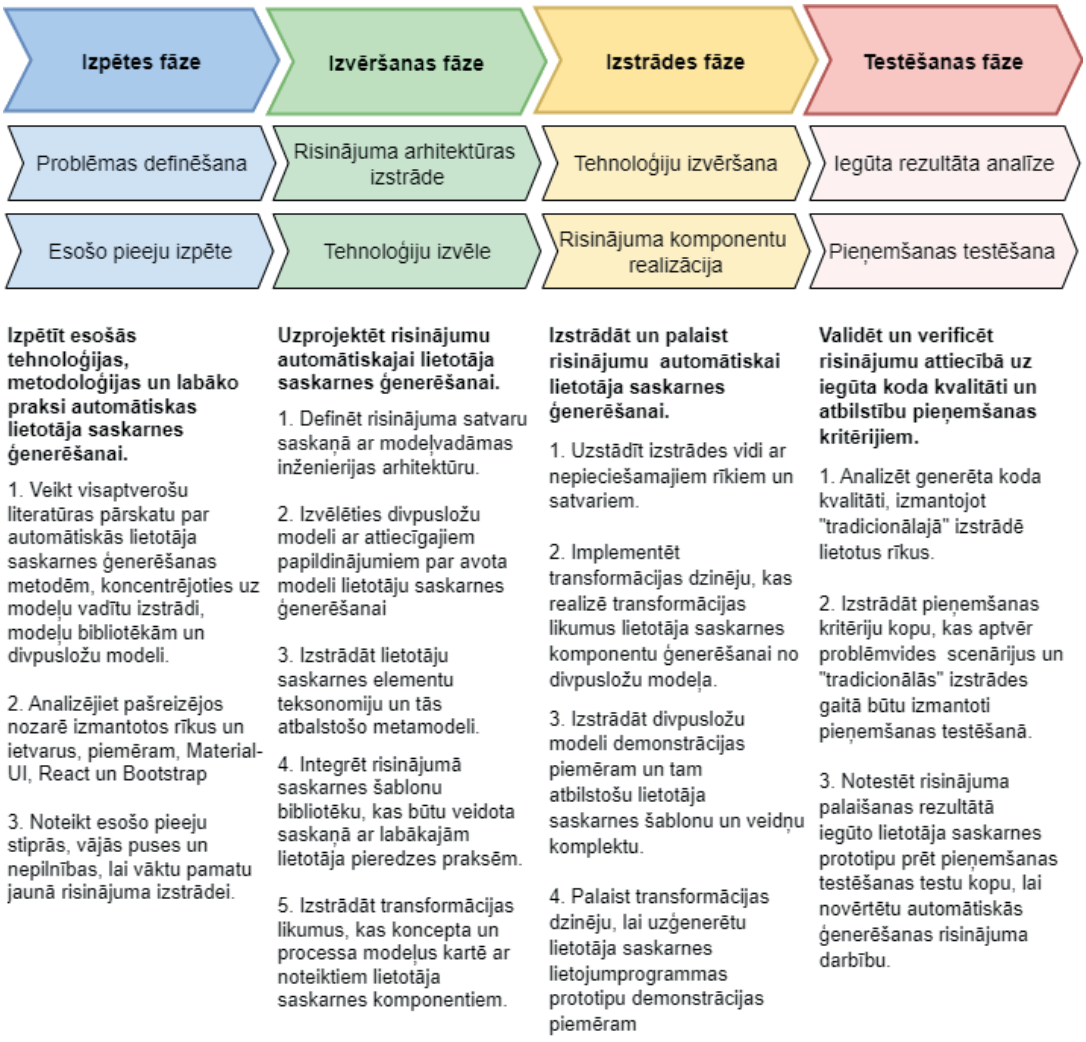
Promocijas darba **mērķis** ir izstrādāt pieeju lietotāja saskarnes prototipa automātiskajai ģenerēšanai no divpusložu modeļa, kurā ir definēti problēmvides loģikas scenāriji. Turklāt, ir plānots, ka iegūtais lietotāja saskarnes prototips tiek definēts modeļa veidā, kas turpmāk ir transformējams kādā no tīmekļa lietotnes priekšgala komponenšu satvariem.

Mērķa sasniegšanai tiek izvirzīti šādi **uzdevumi**:

1) Izpētīt esošos lietotāja saskarnes izstrādes metodes un paņēmienus ar nolūku identificēt potenciālus pamata elementus un formālismu, ko var izmantot lietotāja saskarnes automātiskajā ģenerēšanā.
2) Apkopot informāciju par lietotāja saskarnes elementu daudzveidību ar nolūku definēt lietotāja saskarnes elementu taksonomiju un šablonus, kas kalpos par pamatu mērķa metamodeļa izstrādē.
3) Bagātināt divpusložu modeļa notāciju ar transformācijas likumiem nepieciešamiem elementiem, kas ir procesu metadati, un izveidot divpusložu modeļa (kā avota modeļa) metamodeli
4) Izveidot formātu avota/mērķa modeļa saturam, ar nolūku definēt transformācijas likumus, kur par avota modeli tiek izmantots problēmvides divpusložu modelis un par mērķa modeli tiek sagaidīts lietotāja saskarnes prototips šīs problēmvides atbalsta tīmekļa lietotnei.
5) Demonstrēt izstrādāto risinājumu lietotāja saskarnes prototipa ģenerēšanai uz reālas problēmvides atbalsta tīmekļa lietotnes piemēra un pārbaudīt tā atbilstību sagaidāmajam rezultātam.
6) Novērtēt un secināt par pētījuma rezultātiem.

**Pētījuma metodes**

Uzdevumu izpildei doktora darba ietvaros veikta pētījuma metodoloģija ir definēta saskaņā ar programmatūras izstrādes labākās prakses fāzēm, kas ir izpēte, izvēršana, izstrāde un testēšana, un detalizēti ir parādīta 1.attēlā (Kruchten P., 2003).




**1.att. Promocijas darba ietvaros veikta pētījuma detalizācija**

Izpētes fāzē tiek izmantotas zinātniskās literatūras teorētiskā izpēte un saistīto pētījumu analīze, kas ietver esošo zinātnisko rakstu, grāmatu, monogrāfiju un citu avotu apkopošanu un analīzi, lai gūtu padziļinātas zināšanas par pētījuma objektu un esošajiem risinājumiem.

Izvēršanas fāzes rezultātā ir piedāvāts risinājumu balstīt modeļvadāmas programmatūras inženierijas pamatkoncepcijās: modeļos, metamodeļos un modeļu transformācijās. Par avota modeli transformāciju definēšanai ir izmantots divpusložu modelis un mērķa modelis ir veidots, pamatojoties uz darba autora veidotu tīmekļa lietotnes lietotāja saskarnes elementu taksonomiju. Transformācijas likumu izpildes gaitā ir izmantotas lietotāju saskarnes šablonu bibliotēkas veidnes un tajās definētais pirmkods, kas notiek saskaņā ar "*Model-ToText*" transformācijas nosacījumiem objektu vadības grupas definētajā kontekstā. Kā arī, pēc būtības, atbilstošo lietotāja saskarnes elementu ģenerēšana no divpusložu modeļa seko grafu transformācijas teorijai, kur gan divpusložu modelis, gan tā transformācijas rezultāts - lietotāja saskarnes prototips, abi var tikt apskatīti kā grafi ar attiecīgajiem elementiem (virsotnēm) un pārejām starp tiem (šķautnēm).

Piedāvātā risinājuma izstrādē ir izmantotas sistēmu analīzes, informācijas analīzes un sintēzes, sistēmu modelēšanas, grafu teorijā sakņotās modeļu transformācijas, noteiktas situācijas izpētes (angl. *Case study*) metodes un paņēmieni. No praktiskā skatupunkta risinājums tiek izstrādāts, izmantojot prototipēšanas metodi, un satur tādas aktivitātes kā problēmas un iespējamo risinājumu alternatīvu analīze, eksperimentu veikšana un to rezultātu analīze. Rezultātā izstrādes fāzē ir implementēts iepriekš definēto transformācijas likumu dzinējs, kas ir aprobēts uz nelielas problēmvides fragmenta.

Risinājuma validācija ir veikta divos aspektos. Pirmkārt, modeļu transformācijas rezultātā iegūta koda kvalitātes analīze ir veikta ar *JSLint*, *JSHint* un *ESLint* palīdzību, kas dod iespēju pārliecināties, ka iegūtais rezultāts ir atbilstošs programmēšanas praksēm un vadlīnijām. Otrkārt, ir veikta modeļu transformācijas rezultātā iegūta tīmekļa prototipa testēšana, lietojot standarta (ISO/IEC/IEEE 29119) pieņemšanas testēšanas metodi. Šīs validācijas metodes ietvaros ir izstrādāta potenciālās tīmekļa vietnes pieņemšanas (acceptance) kritēriju kopa pret ko būtu validēta "manuāli" izstrādāta tīmekļa vietne "tradicionālas" izstrādes procesā. Palaižot šo testa kopu modeļu transformācijas rezultātā iegūtam prototipam un pārbaudot, ka automātiski iegūti tīmekļa priekšgala (front-end) komponenti atbilst pieņemšanas kritērijiem, kam atbilstu arī "manuālas" izstrādes rezultāts, ir iespējams pārliecināties, kas ar piedāvāto automatizāciju ir panākts izstrādes ātrumu, nezaudējot rezultāta kvalitāti, jo tie paši kritēriji arī ir atbalstīti.

**Zinātniskais jaunieguvums un praktiskā nozīme**

Promocijas darba rezultātā ir iegūti šādi **jaunieguvumi** pētījuma jomā:

1) Ir veikta divpusložu modeļa analīze un vērtētas iespējas izgūt lietotāja saskarnes prototipus no tā. Veikta divpusložu modeļa priekšrocību un trūkumu analīze lietotāja saskarnes elementu ģenerēšanas kontekstā un ir piedāvāts divpusložu modeļa uzlabojums, lai būtu iespējams ģenerēt transformācijas likumu mērķa elementus, kas ir definēti lietotāja saskarnes taksonomijas veidā.
2) Ir izstrādāts risinājums un atbilstoša transformācijas likumu kopa, kas ļauj ģenerēt lietotāja saskarnes prototipus no divpusložu modeļa. Promocijas darba ietvaros izstrādātais risinājums ir aprobēta JavaScript bibliotēkas React.js prototipu ģenerēšanai Material Design satvara komponentos.
3) Ir definēti formāti avota (divpusložu) un mērķa (lietotāja saskarnes prototipa) modeļu uzglabāšanai un apstrādei, kā arī abu modeļu metamodeļi, kas turpmāk var tikt izmantoti citos pētījumos, kur būs nepieciešamība šos modeļu elementus lietot avota vai mērķa līmenī.
4) Ir izteiktas rekomendācijas Rīgas Tehniskajā universitātē radītā *BrainTool* rīka uzlabošanai un turpmākai attīstībai.

Promocijas darba **praktiskā nozīmība** ir izstrādāts risinājums realizācijas potenciālā ģenerēt lietotāja saskarnes prototipus no divpusložu modeļiem. Darbā ir definēta visa nepieciešama avota un mērķa modeļa specifikācija un transformācijas likumi, kas turpmāk ir lietojami piedāvāta risinājuma atbalsta prototipa realizācijā. Darbā piedāvāts risinājums var būt lietderīgs programmatūras izstrādātāju grupām, kas darbojas ar dažāda veida spraudņu / programmu izstrādi, kas atbalsta programmatūras izstrādes procesu, piemēram, integrētu izstrādes vižu, lietotāja saskarnes izstrādes rīku un sistēmu modelēšanas rīku izstrādātājiem. Kā arī izstrādāta risinājuma demonstrācija, divpusložu modeļa veidošanai izmantojot *BrainTool* rīku, kas šī promocijas darba ietvaros ir papildināts ar lietotāja saskarnes ģenerēšanu, parādītā rīka potenciālu turpmākai tā attīstībai un iespējams, komercializācijai.

**Darba autora publikācijas**

Pētījuma rezultātu aprobācija ir atspoguļota 8 publikācijās starptautiskos un Latvijas Zinātnes padomes atzītos zinātniskos izdevumos:

1. Babris K., Ņikiforova O., Sukovskis U. Brief Overview of Modelling Methods, Life-Cycle and Application Domains of Cyber-Physical Systems. Applied Computer Systems, 2019, Vol. 24, Issue 1, pp. 1.-8., https://doi.org/10.2478/acss-2019-0001 (Web of Science)

*Promocijas darba autora ieguldījums: saistīto darbu analīze, salīdzināšanas kritēriju izstrāde un vērtējumu apkopojuma veikšana.*

2. Ņikiforova O., Babris K., Kristapsons J. Survey on Risk Classification in Agile Software Development Projects in Latvia. Applied Computer Systems, 2020, Vol. 25, No. 2, pp. 105-116. ISSN 2255-8683. e-ISSN 2255-8691. Available from: https://doi.org/10.2478/acss-2020-0012 (Web of Science)

*Promocijas darba autora ieguldījums: saistīto darbu analīze, riska veidu identificēšana, secinājumu izstrāde.*

3. Ņikiforova, O., Babris K., Madelāne L. Expert Survey on Current Trends in Agile, Disciplined and Hybrid Practices for Software Development, Applied Computer Systems, vol.26, no.1, 2021, pp.38-43. https://doi.org/10.2478/acss-2021-0005 (Web of Science)

*Promocijas darba autora ieguldījums: saistīto darbu analīze, programmatūras izstrādes procesa prasību un īpašību definēšana, secinājumu izstrāde.*

4. Ņikiforova O., Zabiniako V., Kornienko J., Rizhko R., Babris K., Gasparoviča-Asīte M. Efficiency Monitoring of Engineering System Designer Work Based on Multi-System User Behavior Analysis with AI/ML Algorithms, 2021 IEEE 62nd International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), 2021, pp. 1-6, DOI: 10.1109/RTUCON53541.2021.9711720 (SCOPUS)

*Promocijas darba autora ieguldījums: eksperimentu veikšana, sistēmas modeļa sākotnējas versijas validēšana.*

5. Ņikiforova O., Zabiniako V., Kornienko J., Rizhko R., Babris K., Nikulsins V., Garkalns P., Gasparoviča-Asīte M. Solution to On-line vs On-site Work Efficiency Analysis on the Example of Engineering System Designer Work, Applied Computer Systems, vol.26, no.2, 2021, pp. 87-95, DOI: 10.2478/acss-2021-0011 (SCOPUS)

*Promocijas darba autora ieguldījums: eksperimentu veikšana, sistēmas modeļa gala versijas validēšana.*

6. Nikiforova O., Babris K., Mahmoudifar F. Automated Generation of Web Application Front-End Components from User Interface Mockups, Proceedings of International Conference on Software Technologies (ICSOFT 2024, July 8-10), SCITEPRESS Digital Library, 2024, pp. 100-111, DOI: 10.5220/0012759500003753 (SCOPUS)

*Promocijas darba autora ieguldījums: saistīto darbu analīze, avota un mērķa metamodeļa izstrāde.*

7. Ņikiforova O., Babris K., Guliyeva A. Definition of a Set of Use Case Patterns for Application Systems: A Prototype-Supported Development Approach, Applied Computer Systems, Applied Computer Systems, vol.29, no.1, 2024, pp.59-67. DOI: 10.2478/acss-2024-0008 (Web of Science)

*Promocijas darba autora ieguldījums: praktiskā piemēra izstrāde, eksperimentu veikšana, lietošanas gadījumu šablonu apkopojums un strukturēšana.*

8. Babris K., Nikiforova O. Towards Automated UI Mockup Generation from Two-Hemisphere Problem Domain Models: A Conceptual Framework and Approach, Proceedings of 19th Iberian Conference on Information Systems and Technologies (CISTI 2024, June 25-28), 2024, pp. 1-6 – pieņemts publicēšanai, būs iesniegts indeksēšanai SCOPUS

*Promocijas darba autora ieguldījums: saistīto darbu analīze, avota un mērķa metamodeļa izstrāde, risinājuma koncepcijas izstrāde.*

9. Babris K., Nikiforova O. From Models to Interfaces: Leveraging the Two-Hemisphere Model for Automated UI Generation, 2024 IEEE 65th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), Riga, Latvia, 2024, pp. 1-6, DOI: 10.1109/ITMS64072.2024.10741944 (SCOPUS)

*Promocijas darba autora ieguldījums: saistīto darbu analīze, risinājuma un aprobācijas piemēra izstrāde.*

Promocijas darba galvenie rezultāti prezentēti trijās starptautiskās zinātniskās konferencēs.

1. CISTI'2024 – 19th Iberian Conference on Information Systems and Technologies, – 2024, 25.–28.06.2024. – Salamanka, Spānija, "Towards Automated UI Mockup Generation from Two-Hemisphere Problem Domain Models: A Conceptual Framework and Approach".

2. ICSOFT 2024 – 19th International Conference on Software Technologies, 08.–10.07.2024. – Dižona, Francija, “Automated Generation of Web Application Front-end Components from User Interface Mockups”.
3. ITMS'2024 – The 65th International Scientific Conference on Information Technology and Management Science of Riga Technical University, 03.–04.10.2024. – Rīga, Latvija, “From Models to Interfaces: Leveraging the Two-Hemisphere Model for Automated UI Generation”.

**Promocijas darba struktūra**

Promocijas darbs ir strukturēts tā, ka pirmajā nodaļā ir apskatīti lietotāja saskarnes veidi un brieduma līmeņi pēc analoģijas ar spēju brieduma līmeni, un novirzīts fokuss uz promocijas darba pētījuma priekšmetu, proti lietotāja saskarnes prototipu. Kā arī ir ieskicēts lietotāja saskarnes prototipa izstrādes process no dažādā abstrakcijas līmenī attēlotiem ieejas datiem un modeļiem. Šajā nodaļā ir sniegts pilnīgs kopsavilkums par rīkiem un tehnoloģijām, kas veido pašreizējo lietotāja saskarnes attīstību kā arī uzsver nepieciešamību pēc viegli saprotamas un lietojamas saskarnes izveidi. Par formālismu lietotāja saskarnes prototipa ģenerēšanas risinājuma izstrādē ir izvēlēti modeļvadāmas inženierijas pamatprincipi, kas ir avota un mērķa modeļi un to metamodeļi, kas ir aprakstīti promocijas darba otrajā nodaļā. Kā arī otrajā nodaļā ir paskaidrota modeļvadāmās izstrādes būtība, kā, veidojot abstraktus modeļus, ir iespējams pārveidot modeli par citu modeli vai pirmkodu. Risinājumā izmantotie un otrajā nodaļā aprakstītie metamodeļi kalpo par pamatu modeļu transformācijas likumu veidošanai, kas ir definēti trešajā nodaļā. Nodaļā ir izskaidrots metodiskais veids, kā izveidot šos transformācijas likumus, lai garantētu, ka tie ir precīzi, elastīgi un pielāgojami dažādām lietotāja saskarnes veidošanas vajadzībām. Promocijas darba ceturtā nodaļa ir veltīta izstrādāta risinājuma aprobācijai un novērtēšanai, kas balstās uz sagaidāmā rezultāta, t. i., lietotāja saskarnes prototipa, izstrādi noteiktai problēmas videi un tā salīdzināšanu ar risinājuma pielietošanas ietvaros izveidoto prototipu. Darba nobeigumā tiek izteikti secinājumi par iegūtajiem rezultātiem un nākotnes pētījuma virzieni.

# 1. LIETOTĀJA SASKARNES IZSTRĀDE

Šajā nodaļā tiek apskatīts kopējais lietotāja saskarnes izstrādes process, tā elementi un cita informācija.

Zem lietotāja saskarnes (angl. *User Interface, UI*) tiek domāta vide, kurā lietotājs mijiedarbojas ar lietojumprogrammu vai sistēmu un otrādi – sistēma dod atgriezenisko saiti lietotājam saprotamā veidā – vai nu lietotājs darbina šo sistēmu, ievada datus vai jāizmanto šīs sistēmas saturs (Nguyen, Vu et al., 2018; Alfaridzi & Yulianti, 2020). Lietotāja saskarne ir būtiska lietojumprogrammas sastāvdaļa (Nguyen, Vu et al., 2018) – tā vienotā veidā apvieno vizuālo dizainu, mijiedarbības dizainu un informācijas arhitektūru (Mukthapuram, 2021).

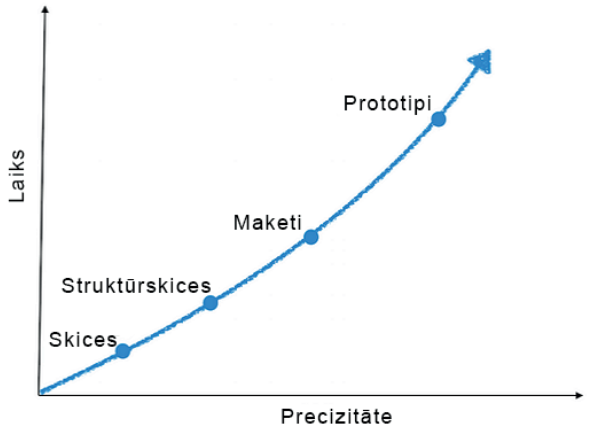
Zem lietotāja pieredzes (angl. *User Experience, UX*) tiek saprasta pieredze, kas cilvēkam rodas, lietojot produktu, lietotāja saskarni vai mijiedarbojoties ar to. Lietotāja pieredze ietver cilvēku izpratni par to, cik apmierinošs, patīkams un viegli lietojams ir produkts (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020; Sharp, Rogers et al., 2019).

Termins UX tiek izmantots IT jomā, izmantojot gan informācijas sistēmu kopumā, gan tās atsevišķo daļu (vietni, lietojumprogrammu utt.), Lai aprakstītu subjektīvo attieksmi, kāda ir lietotājam. Tomēr, neskatoties uz acīmredzamo termina vienkāršību, UX dizains ir sarežģīts un neskaidrs process, kas papildus analīzes un projektēšanas posmiem ietver arī slikti formalizētas procedūras, piemēram, aptaujas, lietotāju darba uzraudzību, veiktspējas testēšanu un darbību žurnālu analīzi, fokusa grupas, utt. (Diehl, Martins et al., 2022). Turklāt pastāv dažādas pieejas UX projektēšanai un šī procesa aizpildīšanai ar konkrētu saturu praksē. UX izstrāde sākotnējā formā ir balstīta uz ūdenskrituma izstrādes modeli. Izstrādes komanda cenšas pēc iespējas vairāk uzzināt par produktu, pirms tiek izveidots pat visvienkāršākais prototips. Šāda izpēte var aizņemt ilgu laiku (mēnešus vai pat gadus), bet tikai to pabeigšana aizsāk projektēšanas komandas darbu. Visas darbplūsmas tiek veidotas secīgā kaskādē, kas apgrūtina darbu paralēlu izpildi un atgriešanos pie iepriekšējiem posmiem, lai veiktu izmaiņas (Kompaniets, Lyz et al., 2020).

Lietotāja saskarne kā gala produkts, parasti iziet posmu no skices (zemas precizitātes produkts) līdz prototipam (augstas precizitātes produkts), kas nozīmē, ka šajā procesā tiek iesaistīti dažādi speciālisti, piemēram, lietotāja saskarnes izstrādātājam jāanalizē un jāapstrādā grafikas dizainera izstrādāts lietotāja saskarnes makets, utt. Šajā procesā tiek ražoti dažādi dizaina produkti,

lai sasniegtu gala produktu – lietotāja saskarni (1.1. att., ieguldītā laika un dizaina produkta precizitātes līmeņa attiecība), kas galu galā ir darbietilpīgs process un ietver sevī vairākas manuālas un laikietilpīgas darbības (Bajammal, Davood et al., 2018).

Katrs precizitātes līmenis nosaka arī prototipa brieduma pakāpi. Prototipēšanas process ietver koncepcijas pārņemšanu no aptuvena melnraksta jeb uzmetuma, lai to galu galā pārvērstu par kaut ko lietojamu, ko mērķa lietotāji var novērtēt. Šis process ir atkārtojams, un tāpēc tam ir vajadzīgs laiks, lai sasniegtu brieduma pakāpi. Tāpat kā jebkuram citam radošam procesam, arī dizaineriem bieži ir jāizstrādā un jāpārstrādā koncepcijas, līdz tiek panākta vienprātība. Katras precizitātes līmeņa un atkārtota dizaina apstrāde, kas saistīti ar vienas precizitātes pārveidošanu citā, var padarīt visu procesu ilgstošu un sarežģītu (Suleri, Pandian et al., 2019).




**1.1. att. Laika un precizitātes grafiks (adaptēts no (Fung, 2019))**

## 1.1. Lietotāja saskarnes veidošanas līmeņi

Lietotāja saskarnes izstrādē ir vairāki precizitātes līmeņi, no kuriem katrs atspoguļo atšķirīgu projektēšanas procesa posmu un piedāvā dažādas detalizācijas un pilnveidošanas pakāpes (Stompff & Smulders, 2015). Šie līmeņi palīdz dizaineriem sazināties un atkārtot dizaina koncepcijas dažādos izstrādes posmos. Tālāk ir sniegts pārskats par izplatītākajiem lietotāja saskarnes dizaina līmeņiem un to, kā tie atšķiras (Rivero, Rossi et al., 2011):

1. Skices – visvienkāršākais lietotāja saskarnes dizaina līmenis. Tie ir ātri, ar roku zīmēti vai digitāli izveidoti dizaina ideju attēlojumi, koncentrējoties uz izkārtojuma struktūru un pamatelementu izvietojuma tveršanu. Tām ir zema precizitāte (angl. *Low-Fidelity*). Skices ir apzināti aptuvenas un neformālas, izlaižot detalizētu vizuālo stilu, krāsas un precīzus mērījumus. Skices par prioritāti izvirza dizaina koncepciju ātru un neformālu nodošanu.
2. Struktūrskices – detalizētāki nekā skices un nodrošina saskarnes strukturālu izklāstu. Tās nosaka lietotāja saskarnes elementu izkārtojumu, satura hierarhiju un funkcionalitāti, nekoncentrējoties uz vizuālā dizaina detaļām. Tām ir vidēja precizitāte (angl. *Medium-Fidelity*). Lai gan struktūrskices ir precīzākas nekā skices, tās joprojām saglabā zemu vizuālās detalizācijas līmeni. Tās izmanto vienkāršas formas, vietturus (angl. *Placeholders*) un vienkāršu tekstu, lai attēlotu saskarnes elementus, uzsverot funkcionalitāti, nevis estētiku.
3. Maketi – statiski, vizuāli saskarnes dizaina attēlojumi. Tie ietver detalizētāku vizuālo stilu, piemēram, krāsas, tipogrāfiju un attēlus, nodrošinot skaidrāku lietotāja saskarnes galīgā izskata un izjūtas attēlojumu. Tie ir augsta precizitātes (angl. *High-Fidelity*). Maketi ir noslīpētāki un vizuāli izsmalcinātāki, salīdzinot ar skicēm un struktūrskicēm. Tās vizuālā izskata un estētikas ziņā ļoti līdzinās galaproduktam, ļaujot ieinteresētajām pusēm labāk iztēloties gala rezultātu.
4. Prototipi ir interaktīvs lietotnes saskarnes izstrādes līmenis, ar kuru lietotājs var mijiedarboties un izjust tā funkcionalitāti. Prototipi simulē gala lietotāja saskarni – pārejas, uzvedību, izskatu un nodrošina visaptverošāku un reālistiskāku pieredzi nekā citi līmeņi, pirms gala lietotāja saskarnes izstrādes. Precizitāte var atšķirties atkarībā no prototipa veida – prototipi arī dalās zemas precizitātes un augstas precizitātes prototipos (Rudd, Stern et al., 1996). Zemas precizitātes prototipi koncentrējas uz pamata mijiedarbību un navigāciju, kamēr augstas precizitātes prototipi piedāvā detalizētāku un reālistiskāku gala produkta attēlojumu, tostarp sarežģītas mijiedarbības un animācijas (Virzi, Sokolov et al., 1996).
5. Gala lietotāja saskarne – galīgā, pilnībā izstrādātā saskarnes versija, kas ir gatava ieviešanai. Šādā lietotāja saskarnē ir iekļauti visi nepieciešamie dizaina elementi,

līdzekļi un funkcionalitāte, protams – ievērojot noteiktās dizaina vadlīnijas un labāko praksi kā arī ar augstāko precizitāti (angl. *Highest-Fidelity*). Gatava lietotāja saskarne ir ideāla pikseļiem, un katra detaļa ir rūpīgi izstrādāta, lai nodrošinātu konsekvenci, lietojamību un vizuālo pievilcību. Tas atspoguļo dizaina procesa kulmināciju, atspoguļojot paredzēto lietotāja pieredzi un zīmola identitāti.

Apskatot skices, struktūrskices, maketus, prototipus un gala lietotāja saskarni var secināt, ka katrs lietotāja saskarnes dizaina precizitātes līmenis kalpo savam mērķim (Rudd, Stern et al., 1996). Šie koncepti var tikt sadalīti dizaina izstrādes procesā, sākot no idejas un koncepcijas izpētes, līdz gala lietotāja saskarnes ieviešanai. Katrs līmenis piedāvā unikālu skatījumu uz lietotāja saskarnes dizainu, ļaujot dizaineriem un izstrādātājiem atkārtot, pilnveidot un apstiprināt dizaina koncepcijas dažādos izstrādes posmos.

### 1.1.1. Skices (*Sketches*)

Skices ir brīvs zīmējums uz papīra vai digitālā rīkā, kas sniedz koncepcijas pamata redzējumu un sinhronizē sapratni starp pasūtītāju un izstrādātāju. Skices var būt ļoti noderīgas projektēšanas procesa konceptualizācijas un sākotnējās vizualizācijas posmā. Attiecīgi šis ir zemas precizitātes (angl. *Low-fidelity*) dokuments, kas vairāk kalpo kā komentārs vai tālākās izstrādes virziena vadītājs. Lietotāja saskarnes izstrādē skices attiecas uz aptuveniem, ar roku zīmētiem vai digitāli veidotiem saskarnes ideju un koncepciju attēlojumiem. Tie parasti ir pirmais solis projektēšanas procesā, kas kalpo kā ātras un neformālas dizaina koncepciju vizualizācijas pirms detalizētāka darba sākšanas. Skices ir būtiski rīki prāta vētrai, ideju izpētei un dizaina koncepciju saziņai ar ieinteresētajām personām (Riaz, Arshad et al., 2022).

Skices ir apzināti aptuvenas un neformālas, koncentrējoties uz pamata izkārtojuma struktūru, elementu izvietojuma un vispārējo dizaina koncepciju tveršanu, neiegrimstot detaļās. Tās ir ātri izveidojamas, ļaujot dizaineriem ātri ģenerēt vairākas idejas. Neskatoties uz to, ka ir pieejami daudzi rīki, lai veidotu digitālas skices, pārsvarā, dizaineri uzsāk darba ieskicēšanu uz papīra, tas ir pietiekami ērti un samazina nepieciešamo laiku, lai ieskicētu iespējamo lietotāja saskarnes skici, kā arī to ir viegli pārstrādāt, ja nepieciešams ir ieviest kādas korekcijas.

### 1.1.2. Struktūrskices (*Wireframes*)

Struktūrskice ir dokuments, kas iezīmē lietojumprogrammas pamatstruktūru – tā ir zemas precizitātes dizaina dokuments, jo tas nenosaka konkrētas detaļas, piemēram, krāsas. Pēc struktūrskices izveides tā tiek pārskatīta un pievienota sīkāka informācija, pēc detalizācijas papildināšanas tā kļūst par augstākas precizitātes (angl. *High-fidelity*) maketu (Silva, Martin et al., 2011).

Struktūrskice kalpo kā rasējums, kas iezīmē lietotāja saskarnes struktūru, izkārtojumu un funkcionalitāti, neiedziļinoties konkrētās dizaina detaļās, piemēram, krāsās, tipogrāfijā vai vizuālajā stilā. Struktūrskices parasti tiek izveidotas agrīnā projektēšanas procesā, lai palīdzētu vizualizēt lietojumprogrammas vai vietnes vispārējo izkārtojumu un plūsmu, pirms tiek ieguldīts laiks un resursi detalizētos dizaina elementos.

Struktūrskices koncentrējas uz lietotāja saskarnes skeleta struktūru, attēlojot tādu elementu izkārtojumu kā navigācijas izvēlnes, satura apgabali, pogas (angl. *Button*), formas un citi interaktīvi komponenti. Tie nodrošina skaidru vizuālu priekšstatu par to, kur katrs elements tiks novietots ekrānā (Riaz, Arshad et al., 2022).

Struktūrskices palīdz izveidot hierarhiskas attiecības starp dažādiem elementiem un ilustrē navigācijas plūsmu saskarnē. Tie nosaka ekrānu vai lapu secību, ar kurām lietotāji saskarsies, mijiedarbojoties ar lietojumprogrammu vai vietni.

Lai gan struktūrskices, galvenokārt, koncentrējas uz izkārtojumu un struktūru, tās var arī norādīt uz pamata funkcionalitāti un mijiedarbības modeļiem. Tas var ietvert noklikšķināmo apgabalu izcelšanu, nolaižamo izvēļņu norādīšanu vai parādīšanu, kā lietotāji var pārvietoties starp dažādām saskarnes sadaļām.

Struktūrskiču izveide agrīnā projektēšanas procesā palīdz identificēt iespējamās lietojamības problēmas un dizaina nepilnības, pirms tiek ieguldīts ievērojams laiks un resursi detalizētā projektēšanā un izstrādē. Šo problēmu risināšana agrīnā stadijā galu galā var ietaupīt laiku un naudu ilgtermiņā, samazinot vajadzību pēc plašas pārskatīšanas vēlākā procesā (Sutipitakwong & Jamsri, 2020).

Neskatoties uz to, ka šobrīd nav pieņemta standarta par struktūrskiču elementiem, tiek izmantots vispārpieņemts simbolu kopums, ko attēlo kā attēlus, pogas, rindkopas, ievades elementus, utt.

Kā sākotnējais ekrāna dizains, struktūrskices bieži ir nepilnīgas (Nguyen, Vu et al., 2018). Tās ir arī pārāk vispārīgas, jo struktūrskicē izstrādātājs bieži koncentrējas tikai uz elementu izkārtojumu. Izstrādātājam ir arī jāizstrādā ekrāna lietotāja saskarne no jauna, pamatojoties uz struktūrskices dizainu. Šis uzdevums varētu būt sarežģīts izstrādātājam, it īpaši, ja viņam nav iepriekšējas pieredzes lietotāja saskarnes izstrādē vai priekšgala programmēšanā (Nguyen, Vu et al., 2018).

Kopumā struktūrskicēm ir svarīga nozīme lietotāja saskarnes izstrādes procesā, nodrošinot pamata ietvaru saskarnes dizaina konceptualizēšanai, saziņai un uzlabošanai. Tās kalpo kā vērtīgs rīks, lai dizaina koncepcijas pārvērstu taustāmos attēlos, kas virza lietotājam draudzīgas un intuitīvas digitālās pieredzes izstrādi.

### 1.1.3. Maketi (*Mockups*)

Makets ir statisks augstas precizitātes dizaina dokuments (Newmanm & Landay, 2000). Atšķirībā no prototipa, tajā nav mijiedarbības. Tas piedāvā dizaina galīgo izskatu un parasti tiek veidots starp struktūrskiču un prototipu veidošanu. Struktūrskices ir izstrādātas, lai būtu iespējams redzēt struktūru un funkcionālās prasības, kuras pēc tam tiek attēlotas maketos. Tāpēc maketi būtībā ir papildinātas struktūrskices ar vizuālu noformējumu – attēliem, krāsām un tipogrāfiju, utt.

Lietotāja saskarnes izstrādē maketi ir vizuālie attēlojumi ar augstāku precizitāti salīdzinājumā ar struktūrskicēm. Kamēr struktūrskices koncentrējas uz lietotāja saskarnes strukturālo izkārtojumu un funkcionalitāti, maketi nodrošina detalizētāku un noslīpētāku attēlojumu, bieži ietverot vizuālā dizaina elementus, piemēram, krāsas, tipogrāfiju, attēlus un stilu. Maketu mērķis ir radīt lietotāja saskarnes galīgo izskatu un sajūtu, ļaujot ieinteresētajām personām precīzāk vizualizēt paredzēto dizainu un lietotāja pieredzi (Uzayr, 2022).

Atšķirībā no struktūrskicēm, kas parasti ir pelēku toņu un minimālistiski, maketos ir iekļauti vizuālā dizaina elementi, kas atspoguļo lietotāja saskarnes estētisko izskatu. Tas ietver krāsu, fontu,

ikonu, attēlu un citu grafisko līdzekļu izmantošanu, kas veicina saskarnes vispārējo vizuālo identitāti un zīmolu.

Maketi sniedz detalizētu lietotāja saskarnes attēlojumu, precīzi parādot, kā elementi parādīsies ekrānā izmēra, atstatuma, līdzinājuma un stila ziņā. Tie ietver reālistisku saturu un attēlus, lai simulētu faktisko lietotāja pieredzi un palīdzētu ieinteresētajām personām labāk izprast, kā saskarne izskatīsies un darbosies.

Kopumā maketiem ir svarīga nozīme lietotāja saskarnes izstrādes procesā, mazinot plaisu starp konceptuālām struktūrskicēm un galīgajiem lietotāja saskarnes dizainiem. Tie nodrošina vizuāli bagātīgu un detalizētu saskarnes attēlojumu, palīdzot dizaineriem, klientiem un ieinteresētajām personām vizualizēt un novērtēt piedāvāto dizainu pirms ieviešanas.

### 1.1.4. Prototipi (*Prototypes*)

Kaut arī termins prototips var attiekties uz visu, kas kalpo, lai attēlotu sistēmu kopumā, un tāpēc to laiku pa laikam lieto, lai apzīmētu shēmu kopumu vai maketu kopumu, to visbiežāk lieto, lai atsauktos uz interaktīvu prototipu. Interaktīvie prototipi parasti tiek veidoti HTML vai kādā citā, pietuvinātā gala produkta vidē, un ļauj dizainerim parādīt, kā lietotājs mijiedarbosies ar gatavo vietni (Newmanm & Landay, 2000). Prototipa galvenā ideja ir nodrošināt visas sistēmas vai noteiktas mijiedarbības simulāciju, ko nenodrošina skices vai maketi.

Literatūrā ir dažādas pieejas prototipu veidošanai, kurām katrai ir savas priekšrocības un trūkumi (Nacheva, 2017). Prototipu veidošanas process tiek veikts vairākos posmos, kas var ilgt līdz pat gala produkta realizācijai. Prototipus var izveidot gan programmatūras projekta sākuma stadijā, gan tieši pirms tā pabeigšanas. Tos var atkārtot tik ilgi, cik nepieciešams, kas nozīmē, ka "process ir iteratīvs" (Warfel, 2009; Nacheva, 2017).

Prototipi, atšķirībā no statiskajām struktūrskicēm vai augstas precizitātes maketiem, ļauj iesaistītajām personām interaktīvi mijiedarbojoties ar lietotāja saskarni – faktiski, pārvietoties pa dažādām lapām, ekrāniem un dinamiski izmantot galvenās iespējas un funkcijas. Zem prototipa var saprast interaktīvus digitālus lietotāja saskarnes attēlojumus, kas simulē galaprodukta funkcionalitāti un lietotāja pieredzi – lietotāji var noklikšķināt uz pogām, ievadīt tekstu,

pārvietoties starp ekrāniem un veikt citas darbības, kas ir līdzīgas tam, kā viņi mijiedarbotos ar faktisko produktu.

No tā var secināt, ka prototipi ir nozīmīgi lietotāja saskarnes projektēšanas un izstrādes rīki – tie ļauj dizaineriem un iesaistītajām personām pieņemt lēmumus un pārbaudīt lietotāja saskarnes lietojamību un lietotāja pieredzi vēl pirms gala lietotāja saskarnes izstrādes. Tomēr, augstas precizitātes prototipu izveide ir laikietilpīga un saistīti ar augstām izmaksām (Kolthoff, 2019). Izstrādes izmaksas var samazināt, savlaicīgi atklājot dizaina defektus, ja prototipi pirms ieviešanas tiek pārbaudīti, salīdzinot ar programmatūras prasībām (De Macedo, Fontão 2023). Tomēr, pāreja no scenārijiem uz formālām specifikācijām joprojām ir slikti definēta, un prototipu veidošana joprojām ir sarežģīta, savienojot lietojumprogrammas problēmvidi ar lietotāja saskarni. Vissvarīgākais ir tas, ka prototipu veidošanas un scenāriju pieejai trūkst integrācijas vispārējā prasību izstrādes procesā (Elkoutbi, Khriss et al., 1999).

Literatūrā tiek norādīs uz trīs galvenajām pieejām (1.1. tabula) prototipu izstrādē: izpētes (angl. *Exploratory*), eksperimentālā (angl. *Experimental*) un evolucionārā (angl. *Evolutionary*) (Nacheva, 2017).

Kad tiek runāts par prototipu veidošanu, katras pieejas galvenā uzmanība tiek pievērsta prototipa precizitātei, kas dažādām pieejām ir atšķirīga (Nacheva, 2017).

Izpētes prototipu veidošanas precizitāte ir zema, nevizualizējot produkta galvenās iezīmes. Šī pieeja nav piemērota lietotāju cerību izpētei. Prototipi ir nefunkcionāli (horizontāla orientācija) un nav galīgās sistēmas daļa (Nacheva, 2017).

Turpretim eksperimentālā prototipu veidošanas pieeja tiek izmantota daļēji realizētu risinājumu (sistēmas komponentu) lietojamības pārbaudei, kuri nevar pilnībā darboties. Tie bieži vien ir būtiski sistēmas komponenti. Tāpēc prototipu precizitāte ir vidēja – orientācija var būt gan vertikāla, gan horizontāla (Nacheva, 2017).

Trešā pieeja – evolucionārā prototipu veidošana, ir nepārtraukts process, lai pielāgotu lietojumprogrammu sistēmu strauji mainīgiem organizatoriskajiem ierobežojumiem (Nacheva, 2017; Bäumer, Bischofberger et al., 1996). Prototipus var izmantot galveno produktu īpašību prezentēšanai un lietojamības testēšanai un novērtēšanai.

**1.1. tabula**

**Trīs galvenās pieejas prototipu izstrādē (aizgūts no (Nacheva, 2017))**

| | **Izpētes** | **Eksperimentālā** | **Evolucionārā** |
|---|---|---|---|
| **Mērķis** | *Pētījums*: sistēmas prasību precizēšana un dažādu ieviešanas alternatīvu apspriešana | *Novērtējums*: veicot lietotāju testēšanu, mērķis ir novērtēt, vai tehniskie risinājumi atbilst sistēmas prasībām | *Izmaiņu pielāgošana*: pastāvīga sistēmas pielāgošana dinamiski mainīgajai videi |
| **Izpētes objekts** | Sistēmas prasības | Daļēji realizēts risinājums | Detalizētas sistēmas prasības |
| **Programmatūras izstrādes procesa posms** | Izstrādes procesa sākuma posmiem noteikt prasības | Katram posmam pēc sākotnējo prasību saņemšanas | Visā izstrādes procesā, arī lai realizētu galīgo sistēmu |
| **Precizitāte** | Zema | Vidēja | Augsta |
| **Orientācija** | Horizontāla | Horizontāla vai vertikāla | Vertikāla |
| **Rezultāts (mijiedarbība ar galīgo sistēmu)** | Ātrais (prezentācijas) prototips | Ātrais (prezentācijas) prototips vai komponenti (funkcionālais prototips) | Izmēģinājuma sistēma vai galīgā sistēma |

## 1.2. Lietotāja saskarnes ģenerēšanas veidi

Lietotāja saskarni var veidot pilnībā manuāli – sākot no skiču zīmēšanas, beidzot līdz gatavai saskarnei. Kā arī var izmantot pusautomātisku un automātisku pieeju, lai samazinātu izstrādes laiku un samazinātu izstrādes piepūli (Molina, Meliá et al., 2002).

Lēmumi, kas pieņemti pašā sākumā – analīzes fāzē – ir būtiska nozīmē tālākajā izstrādes gaitā, jo tie noteiks ierobežojumus nākošajām iterācijām. Pārejot uz izstrādes un ieviešanas fāzēm, var izmantot dažādas pieejas. 1.2. tabulā ir apkopotas trīs projektēšanai piedāvāto pieeju galvenās īpašības.

Šobrīd, ģeneratīvā mākslīgā intelekta laikā, dizaineriem ir nepieciešams piedalīties analīzes un projektēšanas fāzē, lai varētu noteikt kopējo izstrādes kontekstu un noteikt ierobežojumus vai akcentus. Dizaineriem ir jāpieņem lēmumi pamatojoties uz savām zināšanām un iegūto pieredzi (Molina, Meliá et al., 2002). Šajā gadījumā izstrādātājiem var noderēt uz šabloniem balstīta izstrāde kas ierobežotu plašu izvēles kopumu no kā izvēlēties. Izmantojot jau esošās pieredzes un labās prakses var samazināt izstrādes laiku un padarīt gala produktu vienkāršāk saprotamu gala lietotājam.

**1.2. tabula**

**Lietotāja saskarnes izstrādes pieeju salīdzinājums (aizgūts no (Molina, Meliá et al., 2002))**

| | **Manuāla izstrāde** | **Pusautomātiska izstrāde** | **Automātiska izstrāde** |
|---|---|---|---|
| **Izstrādes pūles** | Augstas | Vidējas | Zemas |
| **Izstrādes izvēles** | Daudzas | Daudzas | Dažas |
| **Kļūdu iespējamība** | Jā, augsta | Jā, vidēja | Nē |
| **Kartēšana no konceptuāliem šabloniem** | Atkarīgs no dizainera | Atkarīgs no dizainera, bet ar palīdzību | Vienmēr |
| **Prototipu izveides ātrums** | Zems | Vidējs | Ātrs |
| **Pielāgojamība** | Augsta | Vidēja | Zema |

### 1.2.1. Manuālā lietotāja saskarnes ģenerēšana

Manuālās lietotāja saskarnes veidošana ir detalizēts un iteratīvs process, kas parasti sākas ar izpratni par projekta prasībām un lietotāju vajadzībām. Šis posms ietver plašu izpēti un saziņu ar ieinteresētajām personām, lai apkopotu visu nepieciešamo informāciju par paredzēto

funkcionalitāti, lietotāju vēlmēm un biznesa mērķiem. Dizaineri bieži veido vai apstrādā personas un lietotāju stāstus, lai iegūtu dziļāku izpratni par mērķauditoriju. Šādā veidā tiek mēģināts panākt, ka uz lietotāju orientētā pieeja nodrošinātu, ka dizains atbilst lietotāju mērķiem un uzvedībai (Qing, Ibrahim et al., 2024). Pēc tam prasības tiek pārvērstas sākotnējās skicēs vai struktūrskicēs, kas kalpo kā pirmais lietotāja saskarnes struktūras un izkārtojuma vizuālais attēlojums. Struktūrskices koncentrējas uz elementu izvietojumu un kopējo plūsmu, neiedziļinoties vizuālās detaļās, piemēram, krāsās un tipogrāfijā.

Kad skices vai struktūrskices tiek apstiprinātas, nākamais solis ir izveidot detalizētus maketus vai prototipus. Šajā posmā tiek izvēlēti projektēšanas rīki, piemēram, *Adobe XD*, *Sketch* vai *Figma*, lai izstrādātu augstas precizitātes prototipus, kas ļoti līdzinās galaproduktam (Santoso, 2024). Dizaineri izvēlas krāsas, fontus, attēlus un citus vizuālos elementus, lai izveidotu estētiski patīkamu un funkcionālu saskarni. Viņi pievērš īpašu uzmanību tādiem dizaina principiem kā līdzsvars, kontrasts un hierarhija, lai nodrošinātu, ka lietotāja saskarne ir ne tikai vizuāli pievilcīga, bet arī intuitīva un viegli lietojama (Huddleston, 2017). Lietojamības dizains ir arī būtiska šīs fāzes sastāvdaļa, kurā dizaineri nosaka, kā lietotāji mijiedarbosies ar dažādiem elementiem, tostarp pogām, veidlapām un navigācijas izvēlnēm (Kimseng, Kurnia et al., 2024). Prototipu veidošanas rīki dizaineriem bieži vien ļauj simulēt šīs mijiedarbības, nodrošinot reālistisku pieredzi, ko ieinteresētās personas var pārskatīt un sniegt atsauksmes.

Manuālās lietotāja saskarnes ģenerēšanas pēdējais posms ir ieviešanas fāze, kurā apstiprinātais dizains tiek pārvērsts faktiskā kodā. Tas ietver priekšgala izstrādātājus, kuri izmanto HTML, CSS un JavaScript, lai izveidotu lietotāja skaskarni, kā paredzēts (Vadiyala, Baddam, 2017). Izstrādātāji cieši sadarbojas ar dizaineriem, lai nodrošinātu, ka lietotāja saskarnes vizuālie un interaktīvie aspekti tiek precīzi realizēti (Shobha, 2024). Viņiem arī jānodrošina, lai lietotāja saskarne būtu atsaucīga, kas nozīmē, ka tai ir labi jādarbojas dažādās ierīcēs un ekrāna izmēros. Šajā fāzē nepārtrauktai pārbaudei un iterācijai ir izšķiroša nozīme, lai identificētu un novērstu visas radušās problēmas (Jongmans, Jeanno et al., 2022). Lietotāju atsauksmes bieži tiek iekļautas, lai vēl vairāk uzlabotu saskarni, nodrošinot, ka tā atbilst vēlamajiem lietojamības un funkcionalitātes standartiem. Šis rūpīgais projektēšanas, prototipēšanas un ieviešanas process nodrošina, ka gala lietotāja saskarne ir labi izstrādāta, lietotājam draudzīga un atbilst projekta mērķiem un prasībām.

### 1.2.2. Pusautomātiska lietotāja saskarnes ģenerēšana

Pusautomātiskajā pieejā dizains netiek veikts tikai manuāli, jo procesam palīdz dažādi uz dizaina modeļiem balstīti rīki, kas palīdz iegūt nepieciešamos artefaktus. Pēc tam, iespējams iegūt vai nu prototipus vai jau gatavu lietotāja saskarni (Pelechano, Pastor et al., 2002). Daži autori piedāvā (Molina, Meliá et al., 2002; Bell, 1998) sava veida ģeneratorus, ko sauc par translāciju, kurā analīzes problēmvidei ir pievienotas dizaina veidnes. Ar šādu pieeju var iegūt gan strukturālus, gan dizaina modeļu kopumu noteiktai arhitektūrai, kuru dizainers var izvēlēties no atbilstoša šablona – dizaineriem ir iespēja jau projektēšanas fāzē izvēlēties šablonus, kas atbilst analīzes fāzē norādītajiem modeļiem. Pusautomātiski ģenerētas lietotāja saskarnes laika gaitā varētu kļūt grūtāk apkalpot un atjaunināt, īpaši, ja ir nepieciešamas izmaiņas dizainā vai parādās jaunas tehnoloģijas.

### 1.2.3. Automātiska lietotāja saskarnes ģenerēšana

Automātiskās pieejas galvenā priekšrocība ir tā ka jānodrošina ir tikai informācijas vākšana prasību un analīzes fāzēs, tādējādi ļoti agrīnā fāzē iegūstot faktiskus rezultātus, izmantojot automātisku kodu ģenerēšanu. Veicot izmaiņas projektējumā inženieris var mainīt sistēmas modeļus faktiskā koda vietā un pēc tam ģenerēt kodu no izmainītā modeļa (Sunitha & Samuel, 2019). Izmantojot automātisku pieeju var samazināt arī projekta izmaksas, jo iespējams agrās fāzēs saskaitīt funkciju punktus (Pastor, Abrahao et al., 2001). Ar to var pamatot, ka gala produktivitāte, kas iegūta ar šo pieeja ir labāka salīdzinājumā ar iepriekšējiem. Tomēr, tīra automātiska ģenerēšana var nebūt piemērojama vai pat nav vēlama visām sistēmām, it īpaši, ja problēmvide ir īpaša vai tai nav izstrādāta šabloniska pieeja, kā arī gadījumos, kad nepieciešama optimizācija. Šādos gadījumos labāk der vai nu pusautomātiska – kur iespējams piemērot automātiski ģenerētos artefaktus vai manuāla – gadījumā, kad ir sarežģīta vai īpaša problēma.

## 1.3. Lietotāja saskarnes prototipu ģenerēšanas risinājumi

Šajā nodaļā tiek apskatīti esoši lietotāja saskarnes ģenerēšanas rīki un pieejas. Sakarā ar to, ka lietotāja saskarnes izstrādes process tā klasiskajā izpratnē (no skices līdz prototipam) tiek veidots vairākos posmos, tad arī gala produktu (lietotāja saskarni) var iegūt no dažādiem ievades datiem.

Ir vairāki pieejami risinājumi, kuri mēģina samazināt dizaina procesa ietekmi uz lietotāja saskarnes izveidošanu automatizējot kādu no soļiem vai procesu kopumā. Neskatoties uz to, ka pastāv daudzi rīki, kas tieši vai netieši atbalsta programmatūras prototipu veidošanu vienā vai otrā aspektā, trūkst sistēmas "viss vienā", kas visaptveroši atbalstītu visu programmatūras prototipu veidošanas procesu (Suleri, Pandian et al., 2019).

UX joma pēdējo desmit gadu laikā ir ļoti mainījusies. No sarežģītas projektēšanas programmatūras izmantošanas kļuva vienkāršāki rīki, kas ļauj izveidot ātrus modeļus (prototipus). Šīs izmaiņas padarīja UX metodes populāras ne tikai Silīcija ielejā, bet arī visā pasaulē, jo uzņēmumi saprot, cik svarīgi ir sniegt lietotājiem lielisku pieredzi. Tomēr UX ekspertu pieņemšana darbā nenozīmē, ka organizācijā būs tūlītēja izpratne par dizaina ietekmi vai gatava vide pārmaiņām. Ar laiku dizains kļūst nobriedis, un pakāpeniski mainās arī organizatoriskie procesi. Šādā situācijā tādas sistēmas kā spēju brieduma modeļa integrācija (angl. *Capability Maturity Model Integration, CMMI*) ir ļoti svarīgas (Peres, Meira et al., 2015), lai vadītu grupas un piemērotu ietekmīgus procesus visos departamentos. Procesu uzlabošanas metodoloģijas nav paredzētas tikai programmatūras izstrādes komandām. Lai izveidotu spēcīgu UX programmu, ir nepieciešami labi definēti procesi, kas atbilst organizācijas mērķiem (Gilbert, Fischer et al., 2021). Organizācijas var uzlabot savu briedumu, koncentrējoties uz procesu izstrādi un slīpēšanu, kas uzlabos ne tikai UX komandu, bet arī visu organizāciju.

CMMI līmeņi, modeļa brieduma indeksa (angl. *Model Maturity Index, MMI*) un lietotāja saskarnes brieduma līmeņi piedāvā veidus, kā izmērīt un uzlabot procesus, modeļus vai saskarnes. Šeit var novilkt analoģiju un salīdzināt šos līmeņus (1.3. tabula). CMMI līmeņi ir daļa no CMMI modeļa, kas ir saistīts ar procesa uzlabošanu. Tie parāda, cik nobrieduši ir procesi organizācijā, sākot no sākotnējiem līdz optimizētiem. No otras puses, MMI sniedz skalu, lai novērtētu, cik nobrieduši ir modeļi projektā, parādot to specifiskos atribūtus dažādos posmos: sākotnējā, atkārtojamā, definētā un tā tālāk līdz pilnībā optimizētam līmenim, kurā viss ir pilnveidots konkrētā modeļa dzīves cikla ietvaros. Savukārt, lietotāja saskarnes brieduma līmeņi parāda attīstības stadiju, ko sasniedz lietotāja saskarne – tas var būt jebkas, sākot no pamata dizaina koncepcijām līdz sarežģītiem komponentiem, piemēram, grafiskiem elementiem programmatūras sistēmās.

**1.3. tabula**

**CMMI, MMI un lietotāju saskarnes līmeņu analoģija**

| CMMI līmeņi | MMI līmeņi | Lietotāja saskarnes izstrādes brieduma līmeņi |
|---|---|---|
| 0. līmenis. Nepabeigts (angl. *Incomplete*) | Bez specifikācijas | Bez iepriekšējas lietotāja saskarnes projektēšanas |
| 1. līmenis. Sākuma (angl. *Initial*) | Teksta veidā | Izmantojot skices |
| 2. līmenis. Pārvaldīts (angl. *Managed*) | Teksts papildināts ar modeļiem | Izmantojot struktūrskices |
| 3. līmenis. Definēts | Modeļi papildināti ar tekstiem | Izmantojot maketus |
| 4. līmenis. Kvantitatīvi pārvaldīts (angl. *Quantitatively managed*) | Modeļvadāma izstrāde | Izmantojot prototipu |
| 5. līmenis. Optimizēts | Tikai modeļi (koda programmēšanas valodā nav) | Gala lietotāja saskarnes (priekšgala komponenti) |

CMMI aplūko spēju uzlabot procesus programmatūras inženierijā, sistēmu inženierijā un līdzīgās jomās visai organizācijai (Hinderks, Mayo et al., 2022). CMMI mēra organizācijas procedūru briedumu, sākot no nejaušas prakses līdz skaidri izklāstītiem un pastāvīgi pilnveidotiem procesiem. MMI parāda, kā modelis laika gaitā ir mainījies un uzlabojies, parādot tā pieaugošo sarežģītību un praktiskumu. Lietotāja saskarnes brieduma līmeņi nozīmē, kā lietotāja saskarnes dizaina artefakti ir progresējuši vai pilnveidoti, piemēram, skices, struktūrskices, maketi un prototipi. Tā ir virzība no vienkāršām idejām līdz uzlabotiem un interaktīviem lietotāja saskarnes atveidojumiem un artefaktiem.

Lai gan katrs no šiem trim elementiem katrs atbild par savu jomu CMMI – procesi organizācijā, MMI – modeļi projektā un lietotāja saskarne ar kā projekta elements, var redzēt, ka katrā no šiem satvariem ir attīstības pakāpes, katra nodrošinot aizvien augstāku automatizāciju un optimizāciju. Tādejādi var novērot arī to, ka lietotāju saskarnes augstākie attīstības līmeņi ir prototipi un gala lietotāja saskarne un ko arī ir vērts tiekties.

### 1.3.1. No struktūrskicēm ģenerēta lietotāja saskarne

Struktūrskices nodrošina skaidru lietotāja saskarnes izkārtojuma un struktūras izklāstu, koncentrējoties uz funkcionalitāti un satura izvietojumu. Saskarņu ģenerēšana no struktūrskicēm nodrošina stabilu pamatu projektēšanas procesam, mazinot neskaidrības un veicinot skaidrību.

Kā struktūrskiču trūkumu var minēt to, ka nav izteiktas vizuālās detaļas, piemēram, krāsas, attēli un tipogrāfija. Struktūrskices koncentrējas uz izkārtojumu un struktūru nevis uz izskatu. Ja veido lietotāju saskarni tikai no struktūrskicēm, tad var zaudēt būtiskus vizuālus elementus, kas ietekmē vispārējo lietotāja pieredzi.

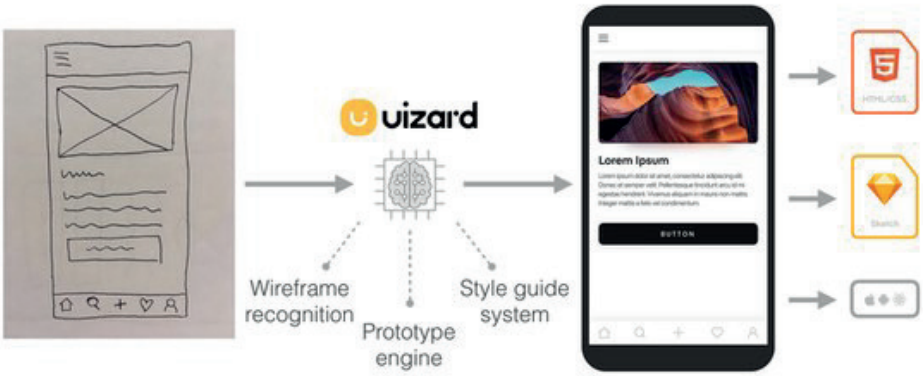
Lietotāja saskarnes ģenerēšana no struktūrskicēm piedāvā vairās priekšrocības – dizaina struktūras skaidrību, efektīvu darbplūsmu un dizaina konsekvenci. Tomēr ir arī savu trūkumi, kā piemēram, ierobežotas vizuālās detaļas, riski nepareizi tās interpretējot, kā arī ieviešanas sarežģītība (Nikiforova, Babris et al., 2020).

Tādi rīki kā *Adobe XD*, *Sketch* un *Figma* ļauj dizaineriem izveidot struktūrskices, kas atspoguļo lietotāja saskarnes izkārtojumu un struktūru. Pēc tam šīs struktūrskices var izmantot par pamatu faktiskā lietotāja saskarnes koda ģenerēšanai.

Viens no interesantiem rīkiem ir *Uizard* (https://uizard.io/), kas izmanto mašīnmācīšanos, lai ar roku zīmētās skices pārveidotu par strādājošiem prototipiem, kurus var eksportēt kā *Sketch* (https://www.sketch.com), un kodu formātos, piemēram, *HTML*, *React.js*, *Vue.js* un *Angular*. *Uizard* izmantot tādus risinājumus kā *pix2code* (Beltramelli, 2018) un *code2pix* (Gundotra, 2018) tā kopējā darbības shēma ir redzama 1.2. attēlā

*SketchiXML* (Coyette, Vanderdonckt et al., 2007) piedāvā iespēju zīmēt maketus programmatūrā (vai izmantot skenētas skices). *SketchiXML* semantikas ir veidotas vienotas modelēšanas valodas klašu diagrammā un katra klase, atribūts vai relācijas ir transformētas XML shēmā izmantojot *USiXML* (*User Interface eXtended Markup Language*) sintaksi (Coyette, Vanderdonckt et al., 2007; Sottet, Calvart et al., 2007). Līdzīgi arī *UsiSketch* izmanto attēlu atpazīšanu un *USiXML* eksportam (Medina, 2016).

*UsiXML* – ir lietotāja saskarnes dizaina specifikācijas valoda. Tas ļauj dizainerim aprakstīt lietotāja saskarni dažādos abstrakcijas līmeņos.




**1.2. att. Uizard kopējā darbības shēma (aizgūts no (Kaluarachchi & Wickramasinghe, 2023))**

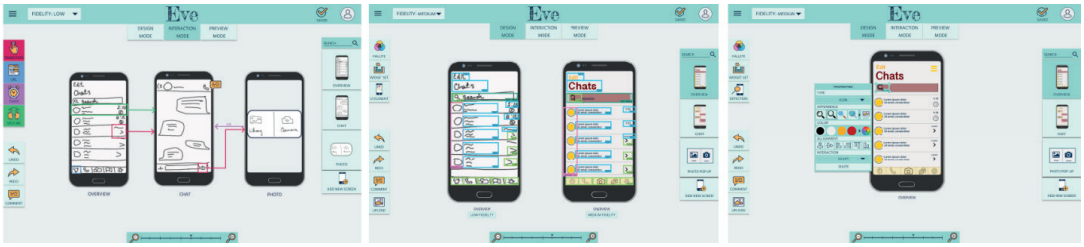
Arī *Microsoft AI Labs* piedāvā rīku sauktu par *Sketch2Code* (Robinson, 2019; Vanita, Piyush et al., 2019), kas izmanto mākslīgo intelektu (angl. *Artificial Inelligence, AI*) un datorredzi (angl. *Computer Vision*), kas ir publiski pieejams *Microsoft AI Lab* pētniecības prototips, kas nodrošina līdzīgu funkcionalitāti kā *Uizard.io*, ļaujot dizaineriem iesniegt savu lietotāja saskarnes skices attēlu, lai tos pārveidotu HTML kodā (Wimmer, Untertrifaller et al., 2020). *Sketch2Code* izmanto *Microsoft Azure Cognitive Services*, u.c. produktus.

Atšķirībā no *Uizard.io* un *Sketch2Code*, kur lietotājam nepieciešams augšuplādēt attēlu, kurš pēc tam tiks transformēts uz prototipu, *SketchingInterfaces* (Wimmer, Untertrifaller et al., 2020) piedāvā izmantot tiešraides kameras translācijas attēlus, ļaujot dizaineriem rediģēt un atjaunot savas skices (piemēram, uz tāfeles), reālajā laikā redzēt kā izskatīsies gatavais prototips (Wimmer, Untertrifaller et al., 2020).

Līdzīgi kā *SketchingInterfaces*, arī *Airbnb* ir izveidojis metodi kā izmantojot mākslīgo intelektu pārvērst skices strādājošā prototipā, faktiski no skices, kas ir uzzīmēta uz tāfeles un nofotografējot to, var iegūt strādājošu HTML (Wimmer, Untertrifaller et al., 2020). Izņemot demonstrācijas video, cita informācija nav pieejama (Wimmer, Untertrifaller et al., 2020), tāpat kā par pieeju, kura tika aprakstīta *XING* (Wimmer, Untertrifaller et al., 2020; Siering, 2017).

Interesanta pieeja ir *MetaMorph* projektam (Pandian, Suleri et al., 2021), kurš izmanto *TensorFlow Object Detection API* (Pang, Nijkamp et al., 2020; Huang, Rathod et al., 2017), kas piedāvā publiski pieejamu API ar kuru var mijiedarboties izstrādātāji, lai izstrādātu savus projektus, kas nosaka no zemas precizitātes shēmas, kādi ir iespējamie elementi.

Izmantojot *MetaMorph* ir izveidots *Eve* projekts (Suleri, Pandian et al., 2019), kura galvenā doma ir priekšlikums, kurā apvienot izstrādes visi, kura spēj gan zīmēt zemas precizitātes shēmas, gan arī tās konvertēt uz vidējas precizitātes maketiem un tālāk uz prototipiem, ar iespēju pārslēgties no viena izstrādes režīma uz otru (1.3. att.).




**1.3. att. Eve trīs režīmi (aizgūts no (Suleri, Pandian et al., 2019))**

### 1.3.2. No maketiem ģenerēta lietotāja saskarne

Lietotāja saskarnes ģenerēšana no maketiem ir kļuvusi par populāru metodi (Bajammal, Davood et al., 2018; Samir, Elsayed et al., 2024), starp dažādām lietotāja saskarnes izstrādes pieejām. Tāpat kā citām pieejām, šai ir vairākas priekšrocības un trūkumi. Maketi labi iekļaujas iteratīvajā projektēšanas procesā, katrā iterācijā ļaujot dizaineriem pilnveidot un atkārtot lietotāja saskarnes dizainu. Informāciju var iegūt gan no projektā iesaistītajām personām – atsauksmes veidā, gan izmantojot lietojamības testēšanu. Maketi pēc izskata ir tuvi gala lietotāja saskarnei, tādēļ tos ir viegli pārskatīt, lai risinātu lietojamības problēmas, uzlabotu estētiku vai iekļautu jaunas funkcijas. Tomēr, maketu interaktivitātes trūkums, rada sarežģītības simulēt lietotāju mijiedarbību un darbplūsmu. Lai to pārbaudītu, iespējams, nepieciešams veidot atsevišķus prototipus vai jāveic lietojamības testēšana, lai apstiprinātu lietotāja saskarnes dizaina lietojamību un funkcionalitāti.

Vizuālo attēlojumu, uzlabotu komunikāciju ar iesaistītajām personām, reālistisku lietotāja pieredzi, iteratīvu projektēšanas procesu un efektīvu izstrādi – to visu piedāvā lietotāja saskarnes

ģenerēšana no maketiem. Augstākminētajam ir vairākas priekšrocības, tomēr ir arī problēmas, kas saistītas ar ierobežotu mijiedarbību un maketu (Nikiforova, Babris et al., 2024a) statisku attēlojumu, laika un resursu ietilpību, kā arī iespējamo neatbilstības risku un ierobežotu elastību.

Maketu veidošanas rīki (piemēram, *Balsamiq*, *Axure RP*, *Moqups* u.c.), dod iespēju dizainerim izveidot lietotāja saskarnes vizuālos attēlojumus, kas būtu augstas precizitātes. Šādi izveidoti maketi ir vērtīgs resurss gala lietotāja saskarnes ģenerēšanai.

Eksistē vairākas pieejas kā ģenerēt lietotāja saskarni no maketiem. Šobrīd vienas no aprakstītākajām un populārākajām ir:

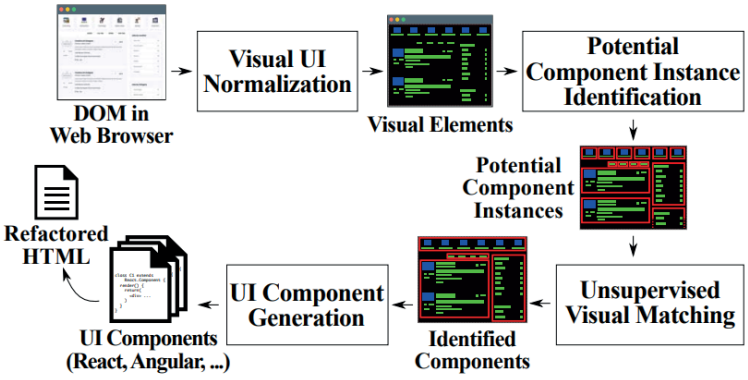
1. Transformācija Attēls-uz-Kodu (angl. *Image-to-Code Conversion*) – metode ietver datorredzes un mašīnmācīšanās paņēmienu izmantošanu, lai analizētu maketu attēlus un ģenerētu atbilstošu kodu vai iezīmēšanas valodu, piemēram, HTML/CSS vai XML.
2. Uz šabloniem sakņota ģenerēšana (angl. *Template-based Generation*) – šajā pieejā tiek izmantoti iepriekš noteikti šabloni vai komponenti, lai pārveidotu maketus interaktīvās lietotāja saskarnēs. Maketu elementi ir kartēti atbilstošiem veidnes komponentiem, ļaujot ātri ģenerēt lietotāja saskarnes.
3. Uz likumiem balstīta pieeja (angl. *Rule-based Systems*) – izmanto iepriekš noteiktu likumu vai heiristikas kopu, lai interpretētu un pārveidotu maketus funkcionālās lietotāja saskarnēs. Šie likumi nosaka, kā dažādi maketa elementi un mijiedarbības jāpārvērš kodā vai iezīmēšanas valodā.
4. Semantiskā analīze – metode ietver elementu semantiskās nozīmes analīzi maketā, lai ģenerētu atbilstošus lietotāja saskarnes komponentus. Dabiskās valodas apstrādes (angl. *Natural Language Processing, NLP*) un semantiskās analīzes metodes bieži tiek izmantotas, lai no maketa iegūtu jēgpilnu informāciju.
5. Mašīnmācīšanās metodes – mašīnmācīšanās modeļus var apmācīt, izmantojot maketu datu kopu un to atbilstošo lietotāja saskarnes ieviešanu, lai apgūtu modeļus un attiecības (Adefris, Habtie et al., 2022). Kad šie modeļi ir apmācīti, tie var automātiski ģenerēt lietotāja saskarnes kodu, pamatojoties uz jauniem maketiem (Samir, Elsayed et al., 2024).

6. Hibrīdās pieejas – apvieno dažādas metodes, piemēram, attēlu analīzi, uz likumiem balstītas sistēmas un mašīnmācīšanos, lai izmantotu katras metodes stiprās puses precīzākai un efektīvākai lietotāja saskarnes ģenerēšanai no maketiem.

Augstākminētās metodes atšķiras pēc to sarežģītības, precizitātes un piemērotības dažāda veida maketiem un lietotāja saskarnes prasībām. Atkarībā no konkrētā lietošanas gadījuma un vēlmēm vienu vai vairākas no šīm metodēm var izmantot automātiskai lietotāja saskarnes ģenerēšanai no maketiem.

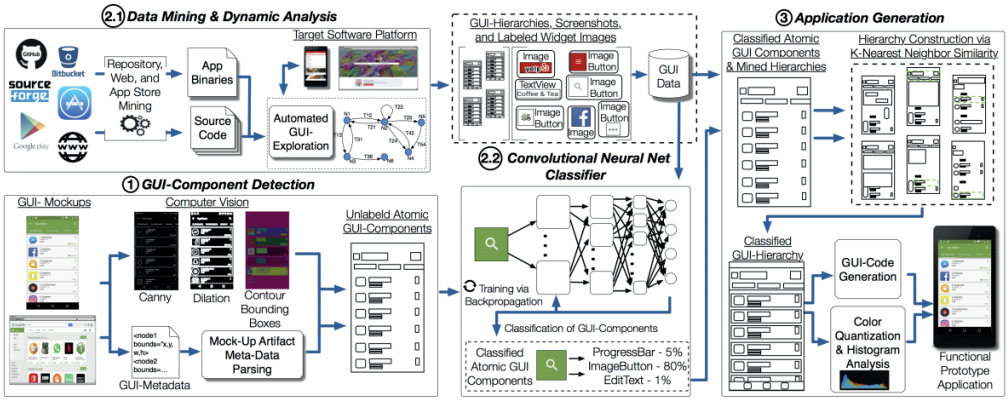
Viens no projektiem, kurš piedāvā ģenerēt lietotāja saskarni no maketiem ir *VizMod* (Bajammal, Davood et al., 2018). *VizMod* (saīsinājums no *Visual Modularizer*) autori piedāvā metodi ar kuru veikt maketa strukturālo analīzi un pārveidot to vairākkārt izmantojamus komponentus, izmantojot neuzraudzīto mašīnmācīšanās (angl. *Unsupervised Machine Learning*) pieeju (1.4. att.). *VizMod* autori uzsver, ka viņu rīks nodrošina iespēju izgūt *React*.js, *Angluar*, u.c. komponentus no maketa (Bajammal, Davood et al., 2018).

Vēl viena metode (Liu, Craft et al., 2018), kas izmanto hierarhisku skatījumu, lai analizētu lietotāja saskarnes semantiku ir centrēta uz mobilajām iekārtām. Šīs metodes autori ir izveidojuši dizaina semantikas leksikonu, kas kategorizē lietotāja saskarnes elementus. Izmantojot ekrānuzņēmumu kā ievades datus un norādot lietotāja saskarnes hierarhiju, metode spēj identificēt elementus izmantojot datorredzes pieeju (Liu, Craft et al., 2018).




**1.4. att. *VizMod* kopējā darbības shēma (aizgūts no (Bajammal, Davood et al., 2018))**

Kā datu vadītu (angl. *Data-driven*) pieeju demonstrē *ReDraw* (1.5. att.) metodes autori (Moran, Bernal-Cárdenas et al., 2020). Autori norāda ka viņu risinājums spēj automātiski atšķirt un klasificēt maketa lietotāja saskarnes elementus, ģenerēt elementu hierarhiju, kas būtu līdzīga tādai, kuru veidotu izstrādātājs, ģenerēt lietotnes, kuras būtu līdzīgas maketam un var pozitīvi ietekmēt kopējo izstrādes darba plūsmu (Moran, Bernal-Cárdenas et al., 2020). *ReDraw* autori uzsver, ka komponentu klasificētie attēli nav pietiekami, lai izveidotu efektīvu lietotāja saskarnes kodu. Lietotāja saskarne, parasti, tiek realizēta kā elementu hierarhiski koki, kur loģiskas grupas sastāvdaļas ir saliktas kopā konteineros (Moran, Bernal-Cárdenas et al., 2020).




**1.5. att. ReDraw kopējā darbības shēma (aizgūts no (Moran, Bernal-Cárdenas et al., 2020))**

Savukārt *Pix2Pix* pieejā (Ferreira, Restivo et al., 2021) autori piedāvā metodi kā ģenerēt no ar rokas zīmētiem maketiem (ievadei tiek izmantots attēls) gatavu HTML tīmekļa lapu. Darbība tiek sadalīta četros posmos (1.6. att.) – attēla iegūšana un pirmsapstrāde, objektu un konteineru izgūšana, objektu hierarhijas izveide un galu galā – HTML koda ģenerēšana. Ģenerētā HTML koda ģenerēšanai autori izmanto CSS režģi (*angl. Grid*), kas pēc pieejas autoru domām ir labāk piemērota nekā tradicionālais lineārais izkārtojums (Ferreira, Restivo et al., 2021).

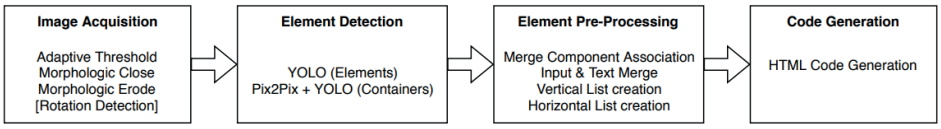
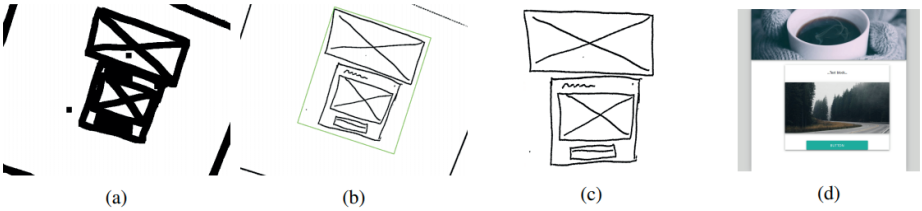
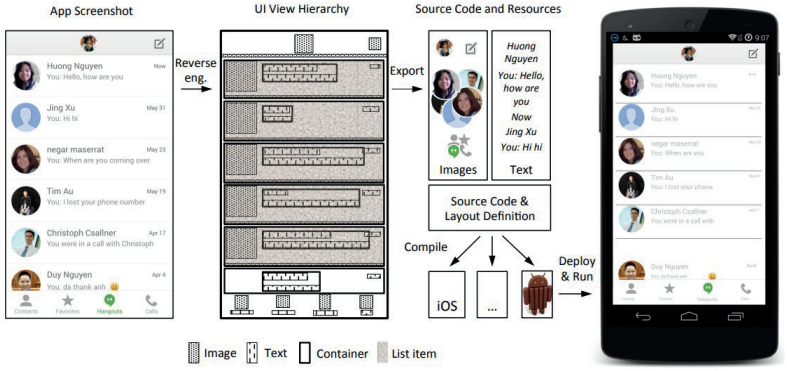



Figure 1: Pipeline processing steps executed sequentially.




**1.6. att. *Pix2pix* kopējā darbības shēma (aizgūts no (Ferreira, Restivo et al., 2021))**

*REMAUI* (Nguyen & Csallner, 2015) piedāvā no ekrānuzņēmuma vai uzzīmēta prototipa (skices) izmantojot datorredzi, izgūt elementus (tekstu, attēlus) un sakārtot tos atbilstošā hierarhijā (1.7. att.).




**1.7. att. *REMAUI* kopējā darbības shēma (aizgūts no (Nguyen & Csallner, 2015))**

Šo elementu hierarhiju tālāk var eksportēt kā strādājošu lietotāja saskarnes prototipu (eksportē izejas kodu, resursu datnes, pēc tam kompilē un palaiž uz telefona) (Nguyen & Csallner, 2015).

### 1.3.3. No prototipiem ģenerēta lietotāja saskarne

Gala lietotāja saskarnes izveidošana no prototipiem (piemēram, JavaScript prototipiem), nozīmē prototipu, pārveidošanu par pilnīgiem tīmekļa lietotnes vai lietojumprogrammas saskarnēm. Sākuma fāzē dizaineri un izstrādātāji strādā kopā, lai uzlabotu un veiktu galīgās izmaiņas prototipu projektos. Tiek pārbaudīts, vai prototipi precīzi attēlo, kāda veida lietotāja pieredze un funkcijas būs reālajā lietošanā. Pēc pārskatīšanas procesa, kad prototipi saņem apstiprinājumu no projektēšanas komandas, ir pienācis laiks izstrādes stadijai, kurā sāk pārveidot prototipus rezultējošā kodā. Veidojot interaktīvu prototipu ar rīku vai ietvaru, kura pamatā ir JavaScript, tajā var nebūt visu galaprodukta darbības aspektu. Piemēram, ja tiek izmantots *Sketch* kopā ar *InVision* (populāru prototipu veidošanas rīku), var novērot statiskus ekrāna izkārtojumus bez dinamiskām darbībām, piemēram, pogu klikšķiem vai lapu pārejām – šīm lietām ir nepieciešama kodēšana, lai tās kļūtu par aktīviem mijiedarbības elementiem. Lietotāja saskarnes prototipu izveidei JavaScript balstītu rīku izmantošanas mērķis ir nodrošināt elastību, veidojot interaktīvas pieredzes, kas atdarina faktisko programmatūras darbību, pirms ideju vizualizācijas iepriekš neprasot sarežģītas kodēšanas prasmes.

Šī metode ļauj dizaineriem ātri izveidot savu dizaina koncepciju funkcionālos modeļus, mazāk paļaujoties uz tehniskajiem programmētājiem sākotnējos posmos, kur pirmreizējas izpētes veikšanai var būt nepieciešamas iterācijas vairākos posmos, kas ietver plašus kodēšanas pasākumus, ko parasti veic specializēti programmētāji.

Izveidot gala lietotāja saskarni no prototipa nozīmē pārveidot, izmantojot rīkus vai ietvarus interaktīvu modeli, kas sastāv no komponentiem, par tīmekļa lietotnes saskarni, kurā visas vizuālās funkcijas ir pareizi kodētas kopā ar nepieciešamajiem mijiedarbības elementiem. Tiek uzskatīts, ka šie elementi darbojas, kā paredzēts, saskaņā ar iepriekš definētiem loģisku noteikumu kopumiem, kas regulē tā uzvedības modeli dažādos scenārijos.

Tas parasti nozīmē JavaScript loģikas, notikumu apstrādes un lietotāju mijiedarbības pārveidošanu no prototipiem organizētā kodā ar priekšgala izstrādes tehnoloģijām. Tie ietver HTML (strukturēšanai), CSS (stila veidošanai) un JavaScript uzvedībai. Izstrādātāji izmanto tādus ietvarus kā *React.js*, *Vue.js* vai *Angular*, lai atvieglotu komponentu izveidi, ko var atkārtoti izmantot viņu lietotāja saskarnes dizainā. Ievieš arī stila un izkārtojuma plānus, kas tiek noteikti

prototipu veidošanas fāzē, izmantojot, piemēram, CSS. Tas palīdz saglabāt vizuālo konsekvenci un pielāgojamību dažādās ierīcēs ar dažādu ekrāna izmēru. Lietotāja saskarnes ģenerēšanas metodē nepieciešams veikt rūpīgu testēšanu un pārbaudi, lai pārliecinātos, vai lietotāja saskarne darbojas pareizi, cik labi tā darbojas un vai tā var darboties nevainojami ar dažādām pārlūkprogrammām un ierīcēm. Tāpat lietotāju atsauksmes vai lietojamības testus var izmantot, lai precīzāk uzlabotu lietotāja saskarnes informāciju, kā arī uzlabotu visu lietotāja pieredzi. Pēc izstrādes un testēšanas posmu pabeigšanas izveidotā lietotāja saskarne tiek ievietota ražošanas vidēs, kur lietotāji var tai piekļūt, lai mijiedarbotos reālās dzīves situācijās.

Ātra ideju tulkošana no prototipiem uz lietotāja saskarnēm ir ļoti noderīga projektēšanā un izstrādē. Lietotāja saskarnes ģenerēšana no prototipiem uzlabo dizaina iterācijas procesu, veicina komandas darbu un rada efektīvākas saskarnes lietotājiem. Tas ļauj ātri iterēt un apstiprināt dizaina koncepcijas, palīdzot agrīni atpazīt un novērst lietojamības problēmas.

### 1.3.4. Zema koda un bez koda platformas

Perspektīvā ir parādījusies jaunā paradigma, ko sauc par zema koda (angl. *Low-code*) sistēmu izstrādi (angl. *Low-code Systems Development, LCSD*) vai bez koda (angl. *No-code*) sistēmu izstrādi, kas pārvērš programmatūras lietojumprogrammu izstrādi uz mazāk manuālu programmēšanu, izmantojot vizuālās programmēšanas pieejas, ko atbalsta grafiskās lietotāja saskarnes un modeļu vadīti dizaini un parasti tiek atbalstīti izmantojot tehnoloģiskās platformas (Alamin, Malakar et al., 2021; Martins, Branco et al., 2023).

Šie zema koda programmatūras izstrādes rīki izmanto grafisko lietotāja saskarni un iepriekš izveidotus elementus, piemēram, lietotāja saskarnes, biznesa objektus, datu objektus tādus kā tabulas un citus komponentus. Lietotājiem ir jāievieto datu lauki, lietotāja saskarnes elementi un biznesa loģika, lai apvienotu lietotāja saskarni ar visiem iepriekš iebūvētajiem komponentiem, lai izveidotu pilnīgu risinājumu (Chen, Lin et al., 2022; Martins., Branco et al., 2023).

Tomēr, kā norādīts esošajā literatūrā, zema koda sistēmu izstrādes platformu izmantošana var sniegt arī nepatīkamas pieredzes, kas no izstrādātāju viedokļa var negatīvi ietekmēt iespējamos ieguvumus. Šīs problēmas ir tieši saistītas ar iespējamiem darba ierobežojumiem, ierobežotu brīvību un radošumu, nepietiekamu dokumentāciju un pārskatu, kā arī vājām komandas sadarbības

iespējām (Conchúir, Ågerfalk et al., 2009; Beranic, Rek et al., 2020; Gao, 2022; Martins, Branco et al., 2023).

Lai gan sistēma, kurā zema koda apvienošana ar lieliem valodu modeļiem (angl. *Large-Language Model, LLM*) sola vairāk kontrolējamu un lietotājam draudzīgāku mijiedarbību ar lieliem valodu modeļiem, tam ir daži ierobežojumi (Cai, Mao et al., 2023). Viens no šādiem ierobežojumiem ir kognitīvās slodzes palielināšanās lietotājiem, kuriem nepieciešams saprast un veikt izmaiņas ģenerētās darbplūsmās (Rokis, Kirikova, 2022). Turklāt precīza un efektīva strukturēta plānošana lielo valodu modeļu ietvaros var būt sarežģīta, un slikti strukturēta plānošana var veicināt lielu lietotāja rediģēšanas slogu (Cai, Mao et al., 2023).

Pastāv uzskats, ka zema koda izstrādei un modeļvadītai inženierijai ir līdzības koncepcijās un mērķos, tomēr, tas nav viens un tas pats. Modeļvadāmas izstrādes tehnoloģijas var uzlabot zema koda izstrādes platformu elastību (mērogojamību) un lietošanas ērtību (Hebig, Khelladi et al., 2017; Gómez, Mendialdua et al., 2020). Šāda starpdisciplināra sinerģētiska metode var uzlabot programmatūras izstrādes vidi, atvieglojot cilvēkiem, kuri nav programmēšanas valodu eksperti, ātri un efektīvi izstrādāt stabilas lietojumprogrammas (Ruscio, Kolovos et al., 2022).

Zema koda programmatūras izstrāde uzņem apgriezienus, un pieaug zinātniskā interese par šo tēmu. Šobrīd pieejamājā literatūrā galvenā uzmanība pievērsta zema koda platformu tehniskajiem aspektiem, bet nav ņemti vērā organizatoriski un sociālie aspekti, kas saistīti ar izstrādes praksi, kas nepieciešama tās ieviešanai (Martinez, Pfister, 2023).

Modeļvadāmai inženierijai ir vairākas priekšrocības, kas nav atrodamas zema koda izstrādēs platformās (Cabot, 2020). Šīs priekšrocības var padarīt modeļvadāmu inženieriju par labāku risinājumu noteiktās situācijās. Modeļvadītā inženierija koncentrējas uz augsta līmeņa modeļu izmantošanu, lai aprakstītu un izprastu sarežģītas programmatūras sistēmas, kas var radīt spēcīgākus, paplašināmus un pārvaldāmākus risinājumus. Atšķirībā no zema koda platformām, kuru mērķis ir vienkāršot izveides procesu neprofesionāliem programmētājiem, izmantojot vizuālās saskarnes un gatavus elementus, modeļvadāmā inženierija piedāvā metodisku un formālu paņēmienu programmatūras projektēšanai. Šis veids ir labs, lai sistēmas sarežģījumus pārvērstu vispārīgākā formā un atvieglotu darbu, kas palīdz izstrādātājiem tikt galā ar programmatūras izmaiņām un tās attīstības raksturu (Gómez, Mendialdua et al., 2020).

Automātiski ģenerētas lietotāja saskarnes kontekstā modeļvadāmai izstrādei ir vairākas priekšrocības salīdzinājumā ar zema koda izstrādes platformām. Izmantojot modeļvadāmo pieeju, ir iespējams izveidot precīzus un formālus modeļus gan lietotāja saskarnei, gan biznesa loģikai – tas nodrošina vienmērīgu kombināciju, kas atbilst iepriekš noteiktiem modeļiem un arhitektūras normām (Cabot, 2020). Šablonu bibliotēkas izmantošana modeļvadāmas izstrādes ietvarā palīdz izstrādātājiem sistemātiski izmantot atkārtojamās un standarta lietotāja saskarnes daļas, tādējādi uzlabojot ģenerēto saskarņu atbilstību vēlamajam, palīdz saglabāt konsekvenci un samazina kļūdu iespējamību. Papildus tam modeļvadāmā izstrāde nodrošina atbalstu sarežģītākām iespējām, piemēram, modeļu transformācijām un programmatūras pirmkoda ģenerēšanai. Šīs funkcijas palīdz automatizēt lietotāja saskarnes izveides procesu saskaņā ar dizaineru vai izstrādātāju sniegtajām augsta līmeņa specifikācijām. Automatizācijas un detalizācijas nodrošināšana parasti nav iespējama ar zema koda platformām un tās var nepiedāvāt rūpīgu pielāgošanu un pārvaldību, kas nepieciešama sarežģītiem lietotāja saskarnes izveides uzdevumiem (Cabot, 2020). Modeļvadāmās izstrādes formālisms atbalstīts ar šablonu bibliotēku un bagātināts ar automātiskajām transformācijām padara to par lielisku iespēju izstrādāt augstākās klases lietotāja saskarnes, kuras var viegli uzturēt sarežģītās lietojumprogrammās.

### 1.3.5. Modeļvadāmas izstrādes pielietošana lietotāja saskarnes prototipu ģenerēšanā

Kā risinājums programmatūras izstrādes izmaksu samazināšanai ir piedāvātas modeļvadāmas metodikas, kuras atbalsta automātiskā ģenerēšana, kuras realizēšanai ir dažādas problēmas. No vienas puses, augstam automatizācijas līmenim ir nepieciešams izmantot detalizētus modeļus, kas ir pretrunā ar iteratīvo izstrādes procesu, kas balstīts uz lietotāja saskarnes maketu pakāpenisku pilnveidošanu, kas raksturīga uz lietotāju vērstajiem attīstības procesiem. No otras puses, slāņveida programmatūras arhitektūra nozīmē atšķirību starp biznesa loģikā un lietotāja saskarnē izmantotajiem modeļiem, kas katrā līmenī rada modeļu konsekvences problēmas (Machado, Couto et al., 2017).

Modeļvadāmās izstrādes paradigmā no augstas abstrakcijas līmeņa modeļiem tiek secīgi izgūti modeļi ar zemāku abstrakcijas līmeni, ideālā variantā, galu galā, izgūstot izpildāmu sistēmu (Stahl, Voelter et al., 2006; Rech & Bunse, 2008). Šis transformācijas process var būt manuāls vai

automatizēts. Abos gadījumos mērķis ir nodrošināt saskaņotu programmatūras izstrādes procesu, nodrošinot gala rezultāta kvalitāti un korekciju. Modeļu automātiska pārveidošana par izpildāmo kodu noved pie izstrādes laika samazināšanās (Sottet, Calvart et al., 2007).

#### *1.3.5.1. CRF pieeja lietotāja saskarnes projektēšanā*

Modeļvadāmā izstrāde parasti tiek izmantota biznesa loģikas un sistēmu datu slāņu izveidei, bet ne tik daudz lietotāja saskarnei (Hailpern & Tarr, 2006). Līdzīgi centieni samazināt lietotāja saskarnes izstrādē un attīstībā iztērēto laiku un pūles, vienlaikus garantējot to kvalitāti, noved pie pieejas, kuru definē kā modeļvadāmas lietotāja saskarnes izstrādes (angl. *Model-based User Interface Development, MBUID*) pieejas (Popp, Raneburger et al., 2013).

Lietotāja saskarnes jomā tiek plaši izmantots un bieži atsaucas uz *Cameleon Reference Framework* (*angl. Cameleon Reference Framework, CRF*) (Coyette, Vanderdonckt et al., 2007; Calvary, Coutaz et al., 2003; Vanderdonckt, 2005), kurā tiek definēti četri (1.4. tabula.) modelēšanas līmeņi (Mbaki, 2013). Faktiski šis satvars vai pieeja, atbilst MDA modeļu transformēšanas pieejai, kad no augstākiem, nespecifiskiem līmeņiem transformē līdz konkrētai programmatūrai.

**1.4. tabula**

**CRF četri konteksta līmeņi (aizgūts no (Coyette, Vanderdonckt et al., 2007; Sottet, Calvart et al., 2007))**

| Pielietojuma konteksts | Procesi un koncepti |
|---|---|
| | Abstrakta lietotāja saskarne (AUI) |
| | Noteikta lietotāja saskarne (CUI) |
| | Gala lietotāja saskarne (FUI) |

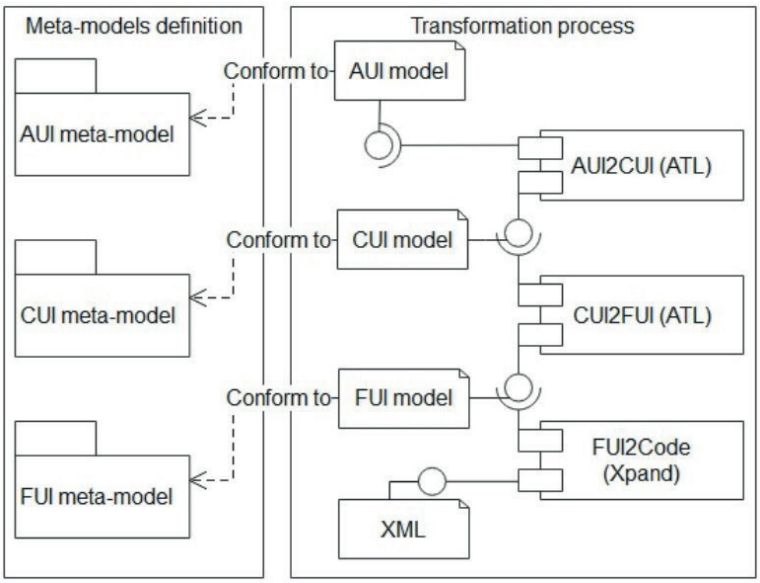
CRF sastāv no četriem līmeņiem un tie ir šādi (Coyette, Vanderdonckt et al., 2007):

1) Uzdevumi un koncepti (angl. *Task & Concept*) – apraksta dažādus veicamos lietotāju uzdevumus un problēmvidē orientētos jēdzienus, kas nepieciešami šo uzdevumu veikšanai.
2) Abstrakta lietotāja saskarne (angl. *Abstract User Interface, AUI*) – definē divas abstraktu mijiedarbības objektu (angl. *Abstract Interaction Objects, AIO*) formas – abstraktus konteinerus (*angl. Abstract Containers, AC*) un atsevišķus komponentus (angl. *Abstract*

*Individual Containers, AIC*), grupējot apakšuzdevumus pēc dažādiem kritērijiem (piemēram, uzdevuma modeļa strukturālie modeļi, kognitīvās slodzes analīze un semantisko attiecību identificēšana) (Coyette, Vanderdonckt et al., 2007).

3) Konkrēta lietotāja saskarne (angl. *Concrete User Interface, CUI*) – pārveido abstrakto lietotāja saskarni nepieciešamajā kontekstā, kurā pielieto konkrētos mijiedarbības objektus (angl. *Concrete Interaction Objects, CIO*), lai izdalītu ekrānvadīklu (angl. *Widget*) izkārtojumu un saskarnes navigāciju. Šajā līmenī tiek abstrahēta gala lietotāja saskarne no saskarnes, kura it neatkarīga no skaitļošanas platformas.
4) Gala lietotāja saskarne (angl. *Final User Interface, FUI*) ir darbspējīga lietotāja saskarne, kura darbojas uz izvēlētās skaitļošanas platformas vai nu caur interpretatoru (piemēram, tīmekļa pārlūku) vai izpildot kodu, pēc koda kompilēšanas.

Vienā no pētījumiem (Lassaad & Fadwa, 2021) autori izmanto līdzīgu pieeju – modeļos balstīto lietotāja saskarnes izstrādi (angl. *Model Based User Interface Development, MBUID*). Autori savā pētījumā piedāvā uz standartu kopumu un atbilstošām tehnoloģijām *(EMF*, *GMF*, *ATL* un *Xpand*) balstītu pieeju (1.8. att.).




**1.8. att. Autoru piedāvātās pieejas kopējā shēma (aizgūts no (Lassaad & Fadwa, 2021))**

Pieeja ir balstīta uz 3 metamodeļiem, kas atbilst dažādiem MBUID abstrakcijas līmeņiem. Saskaņā ar MBUID paradigmu izstrādes process sākas ar lietotāja saskarnes abstraktu specifikāciju (AUI modeli), kas tiek pārveidota par konkrētu specifikāciju (CUI modeli). Tālāk tā tiek pārveidota arī par strādājušu lietotāja saskarni (FUI modeli) tehnoloģiskā telpā (piemēram, skaitļošanas platformā, programmēšanas vai iezīmēšanas valodā). Šīs 2 transformācijas tiek piešķirtas, pateicoties M2M transformācijas likumu kopumam, kas tiek ieviesti, izmantojot ATL (angl. *ATLAS Transformation Language, ATL*) valodu un izmantojot *Eclipse* platformu. Visbeidzot, FUI tiek pārveidota par mobilās lietojumprogrammas pirmkodu. Šī transformācija tiek nodrošināta, pateicoties M2T transformācijas likumiem, kas ieviesti, izmantojot Xpand valodu un Eclipse platformu. Lai novērtētu piedāvātās pieejas iespējamību un lietderību, rakstā ir sniegts reāls piemērs ar *AlMubasher* mobilo lietojumprogrammu (Lassaad & Fadwa, 2021).

Neskatoties uz to, ka CRF ir viens no izplatītiem lietotāja saskarnes modelēšanas veidiem, kā trūkumu var izcelt procesa sarežģītību un apmācības nepieciešamību. Bez pietiekamas apmācības un atbalsta, komandām var rasties grūtības efektīvi izmantot CRF izklāstītos principus un metodes.

Ņemot vērā, ka CRF ir lietotāja saskarnes visaptveroša un uz lietotāju orientēta pieeja, CRF ieviešanai var būt nepieciešams ievērojams laiks un pūles – galvenā sarežģītība ir tā izprašanas un efektīvas ieviešanas aspekts. Piemēram, sarežģītības var radīt komandām ar ierobežotu pieredzi lietotāja saskarnes izstrādē vai saspringtu termiņu projektos.

Neskatoties uz to, ka CRF nodrošina strukturētu metodoloģiju un vadlīniju kopumu lietotāja saskarnes izstrādei, CRF var tiks izveidots tā, ka tas ne vienmēr var pilnībā atbilst katra projekta unikālajām vajadzībām un ierobežojumiem, proti, tas var radīt nepieciešamību veikt izmaiņas izstrādājamā sistēmā un pielāgot to priekš CRF, radot papildus sarežģītību, izmaksas un piepūli.

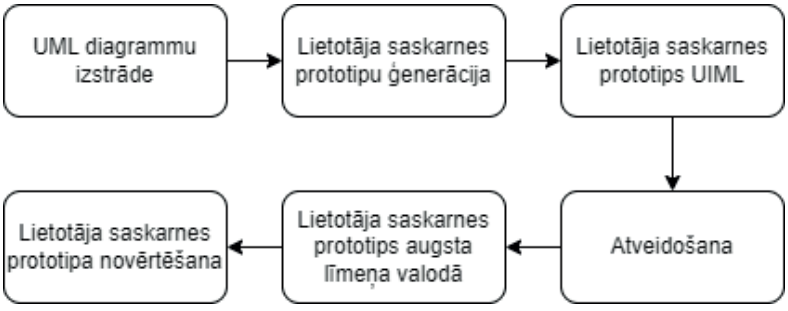
#### *1.3.5.2. No UML diagrammām ģenerēti prototipi*

Vairākos rakstos ir apspriestas dažādas metodes lietotāja saskarnes prototipu ģenerēšanai no vienotās modelēšanas valodas (angl. *Unified Modeling Language* – UML) diagrammām (Almendros-Jiménez & Iribarne, 2008; Almendros-Jimenez & Iribarne, 2005; Elkoutbi & Keller, 2000; Elkoutbi, Khriss et al., 2009; Huzarr & Loniewskil, 2006; Shatnawi & Shatnawi, 2016).

Pārsvarā šajos rakstos tika izvēlētas noteiktas valodas un izmantota šī valodām un vidēm specifiska ģenerēšana. Konkrētas valodas izmantošana lietotāja saskarņu ieviešanai var radīt daudzas problēmas ar dažāda izmēra ekrāna perifērijas ierīcēm, piemēram, rokas ierīcēm (Cimino & Marcelloni, 2012).

Ir ierosināti dažādi veidi, kā izveidot lietotāja saskarnes no lietojumprogrammu problēmvides specifikācijām (Babris, Nikiforova et al., 2019). Vienā no šādām metodēm (Nichols, 2004) ir ierosināta grafisko un runas saskarņu sistēmas automātiskai izveidošanai, izmantojot abstraktas specifikācijas īpaši paredzēts mobilajām ierīcēm. Citā pieejā (Almendros-Jiménez & Iribarne, 2008) tiek ietverta lietošanas gadījumu modeļu kartēšana grafiskos lietotāja saskarnēs – šī metode izmanto aktivitāšu diagrammas, lai aprakstītu lietošanas gadījumus. Tika piedāvāts arī lietotāja saskarnes modelēšana, izmantojot īpašas UML diagrammas, kas tika izveidotas uz lietošanas gadījumu diagrammām.

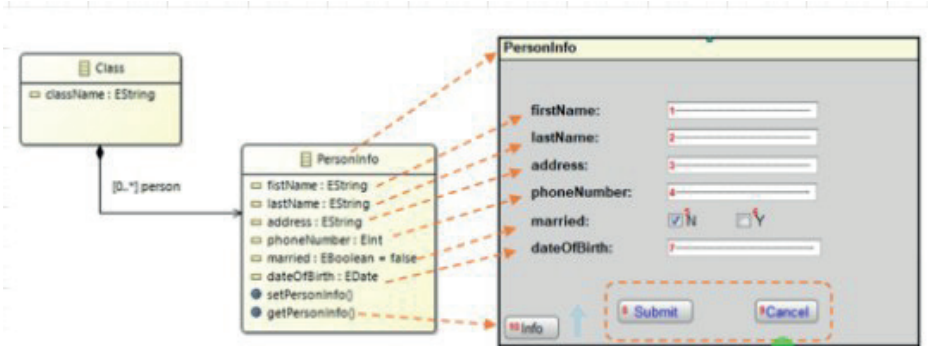
Izpētītajās metodēs (1.9. att.) tiek izmantotas UML diagrammas un lai sasniegtu kvalitatīvu rezultātu šiem modeļiem jābūt pabeigtiem un precīziem, lai izgatavotu prototipus (Shatnawi & Shatnawi, 2016).




**1.10. att. Lietotāja saskarnes ģenerēšanas process no UML (aizgūts no (Shatnawi & Shatnawi, 2016))**

Tika arī ierosināts (Elkoutbi, Khriss et al., 2009) prasību inženierijas procesu lietotāja saskarnes prototipu izveidei no lietojumprogrammu scenārijiem un formālām specifikācijām. Turklāt pētījuma autori (Elkoutbi, Khriss et al., 2009) ir iesnieguši veidu, kā izveidot lietotāja saskarnes prototipus no scenārijiem, kuru rezultātā tiek izveidoti augsta līmeņa Petri tīkli kā

formāla sistēmas specifikācija. Citā pētījumā (Bingbing, 2012) tika ieviesta lietotāja saskarnes ģenerēšana, pamatojoties tikai uz lietošanas gadījumiem, kas apvieno lietošanas gadījumu modeli ar datu plūsmas diagrammām. Taču tie nedarbojas labi, jo ir izmaiņas UML valodā vai ģenerēto lietotāja saskarņu slikta transportējamība starp dažādām platformām un valodām. No otras puses, šajā darbā parādītā metode rada lietotāja saskarnes prototipus valodā, kuras pamatā ir XML. Tas nodrošina lietotāja saskarnes, kas nav piesaistītas nevienai konkrētai platformai, un tās var integrēt ar pašreizējiem CASE rīkiem bez izmaiņām. Vairākos pētījumos tiek kombinētas tehnoloģijas (MACAO, MDE, SEF, utt.) un sasniegti labāki rezultāti, tomēr pētījumos aplūkotie gadījumi ir vairāk paredzēti formu lietotnēm (1.10. att.) un var neaptvert visas problēmvides, kurās tiek izmantotas lietojumprogrammas (Mahatody, Ilie et al., 2021).




**1.10. att. SEF makets, kas ģenerēts no UML klases diagrammas (aizgūts no (Mahatody, Ilie et al., 2021))**

Viens no pētījumiem (Roubi, Erramdani, et al., 2016) ietver lietotāja saskarnes modelēšanu, izmantojot UML diagrammas, īpaši klašu diagrammas, stāvokļa diagrammas un secību diagrammas. Šīs diagrammas atspoguļo attiecīgi lietotāja saskarnes komponentu struktūru, uzvedību un mijiedarbību. Autori veido kartēšanu no aktivitātes nosaukuma un papildus parametriem uz lietotāja saskarnes elementiem konkrētā implementācijā (1.11. att.). Pēc tam modelis tiek pārveidots izpildāmā kodā, izmantojot modeļa transformēšanas metodes. Ģenerētais kods atbilst MVC (angl. *Model-View-Controller, MVC*) modelim ar atsevišķiem komponentiem – datiem (modelis), prezentācijas loģika (skats) un lietotnes loģikai (kontrolleris).

| *PIM GUI elements* | | | *PSM GUI Elements* |
|---|---|---|---|
| Activity | Type | Property | |
| | Input | isPassword | passwordField |
| | | isTextArea | TextAreaField |
| | | N/A | TextField |
| | Selection | isSingleChoice | CheckBox |
| | | isDeployable | ComboBox |
| | | N/A | RadioButton |
| | Click | IsLink | Link |
| | | N/A | Button |
| | Label | N/A | Label |

**1.11. att. Transformācijas likumi no PIM uz PSM (aizgūts no (Roubi, Erramdani, et al., 2016))**

#### *1.3.5.3. No mijiedarbības plūsmām ģenerētas struktūrskices*

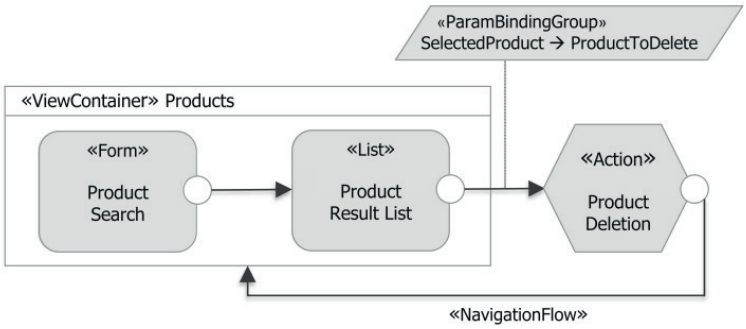
Mijiedarbības plūsmas modelēšanas valoda (angl. *Interaction Flow Modelling Language, IFML*) ir paredzēts lietojumprogrammu priekšgala (angl. *Front-end*) satura, lietotāja mijiedarbības un vadības uzvedības modelēšanai (Brambilla, Umuhoza, et al., 2017). IFML ir OMG atbalstīts projekts ar domu uzlabot lietotāja saskarnes modelēšanu (http://ifml.org). IFML ir pieejams arī redaktors – https://www.ifmledit.org/

Tās metamodelis izmanto UML metamodeļa pamatdatus, specializējas vairākās UML metaklasēs kā pamats IFML metaklasēm un pieņem, ka IFML problēmvides modelis ir attēlots UML (Brambilla, Umuhoza, et al., 2017). IFML modelis atbalsta šādas izstrādes perspektīvas (Brambilla, Umuhoza, et al., 2017):

1) Skata struktūras specifikāciju, kas sastāv no skata konteineru definīcijas, to hierarhiskajām attiecībām, redzamības un sasniedzamības.
2) Skata satura specifikācijas, kas sastāv no skata komponenta – satura un datu elementi, kuri ir ietverti skata konteinerā.
3) Notikumu specifikācijas, kas satur notikumu definīciju – notikums var ietekmēt stāvokli. Notikumus var ģenerēt gan lietotājs ar mijiedarbību, lietojumprogrammatūra vai ārēja sistēma.

4) Notikuma pārejas specifikācija satur definīciju, kura apraksta notikuma ietekmi uz lietotāja saskarni. Parametru saistīšanas specifikācija (angl. *Parameter Binding Specification,*) satur ievades un izvades atkarības starp komponentiem un konteineriem un darbībām (angl. *Actions*)
5) Atsauci uz darbībām, kuras ir izsauktas no lietotāja mijiedarbības notikuma.

1.12. attēlā ir redzams vienkārša IFML modeļa piemērs, aprakstot lietotāja saskarni, kurā lietotājs var meklēt produktu, produktu meklēšanas formā ievadot dažus meklēšanas kritērijus.




**1.12. att. Vienkārša IFML piemēra shēma (aizgūts no (Brambilla, Umuhoza, et al., 2017))**

IFML – Mijiedarbības plūsmas modelēšanas valoda (IFML) ir standartizēta modelēšanas valoda programmatūras inženierijas jomā.

IFML automātiskai lietotāju saskarnes ģenerēšanai ir vairāki trūkumi, piemēram, ar pārmērīgu paļaušanos uz IFML automatizētiem rīkiem un sagatavēm (Sajji, Rhazali et al., 2022). Neskatoties uz to, ka IFML nodrošina strukturētu lietotāja saskarnes modelēšanas sistēmu, atspoguļot visas lietotāja mijiedarbības nianses sagādā lielās pūles un ir laikietilpīgi (Hamdani, Haider et al., 2018). Ja tiek veikta manuāla projektēšanas iejaukšanās šajā procesā, tad izveidotajai lietotāja saskarnei var trūkt radošuma, inovāciju un rezultātā var veidoties vispārīga, neefektīva vai neiedvesmojoša lietotāja saskarne (Queiroz, Marques et al., 2018).

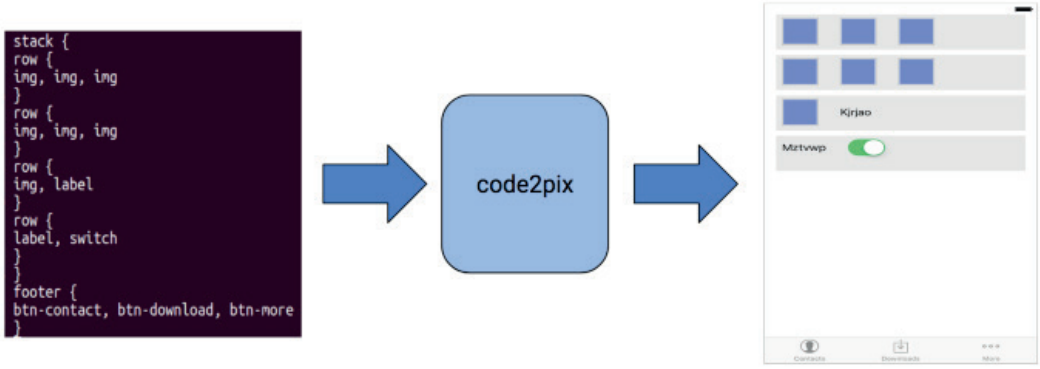
Kā citu trūkumu var minēt abstrakto modeļu pārveidošana pilnīgi funkcionālās gala lietotāja saskarnēs. IFML nodrošina augsta līmeņa lietotāja saskarnes komponentu un mijiedarbības attēlojumu, taču automātisks koda ģenerēšanas process no šiem modeļiem var radīt kļūdas un prasa ievērojamu manuālu iejaukšanos, lai nodrošinātu rezultāta precizitāti un

lietojamību (Cao & Liu, 2020). Sarežģītu lietotāju saskarņu izveide (ar dinamiskiem vai interaktīviem elementiem) var radīt problēmas, kuras nav viegli atrisināt izmantojot tikai ar automatizācijas rīkus.

Modeļvadāmās lietotāja saskarnes kvalitāte ir ļoti atkarīga no avota (ievades) modeļu precizitātes un pilnīguma (Marín, Giachetti et al., 2010). IFML gadījumā, ja modeļi precīzi neatspoguļo lietotāju prasības vai neaptver visus atbilstošos lietotāja saskarnes dizaina aspektus, tad izveidotā saskarne var neatbilst lietotāja vajadzībām vai neatbalstīt sistēmas funkcionalitāti (Queiroz, Marques et al., 2018). No tā izriet, ka ir svarīgi ieguldīt laiku un pūles, lai izveidotu precīzus un visaptverošus IFML modeļus – tas kalpo par pamatu kvalitatīvas lietotāja saskarnes izveidei.

#### *1.3.5.4. No koda ģenerētas struktūrskices*

*Pix2Code* Alternatīvs projekts, saukts par *Code2Pix*, darbojas pretēji *Pix2Code*. *Pix2Code* izmanto esošu lietotāja saskarnes attēlu un pārvērš to par tekstuāliem aprakstiem problēmvides specifiskajā valodā (angl. *Domain-Specific Language, DSL*). *Pix2Code* izmanto dziļās mācīšanās metodes, lai pārveidotu lietotāja saskarnes dizainus, kas attēloti kā attēli, funkcionālā pirmkodā. *Code2Pix* izmanto tekstuālos aprakstus (1.13. att.) veidotus problēmvides specifiskajā valodā un ģenerē lietotāja saskarni.




**1.13. att. Code2pix kopējā darbības shēma (aizgūts no (Gundrota, 2018))**

Tādējādi Code2Pix faktiski ir dziļās mācīšanās (angl. *Deep Learning*) "kompilators", kas spēj ģenerēt saskarnes (Gundrota, 2018). Kopumā Pix2Code izmanto dziļās mācīšanās iespējas, lai automatizētu lietotāja saskarnes izstrādes procesu, piedāvājot daudzsološu pieeju programmatūras izstrādes paātrināšanai un izstrādātāju produktivitātes uzlabošanai.

#### *1.3.5.5. No prasību specifikācijas ģenerēta lietotāju saskarne*

Lietotāja saskarnes ģenerēšana no prasībām ir metodika, kuras mērķis ir izveidot saskarnes tieši no norādītajām sistēmas funkcionālajām un nefunkcionālajām prasībām. Lai gan šī pieeja piedāvā vairākas priekšrocības, tai ir arī savs izaicinājumu kopums.

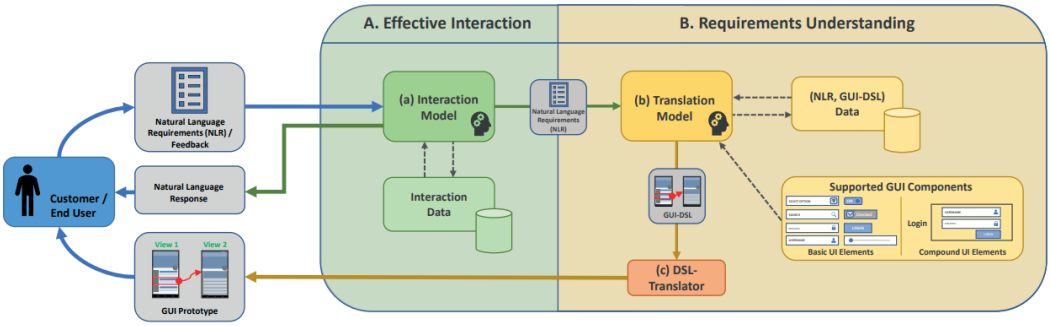
Saskarņu ģenerēšana tieši no prasībām nodrošina atbilstību vispārējiem biznesa mērķiem un lietotāju vajadzībām. Pārvēršot prasības lietotāja saskarnes elementos, dizaineri var noteikt prioritātes funkcijām un funkcijām, kas ir būtiskas galalietotājam, tādējādi uzlabojot vispārējo lietotāju apmierinātību.

Automātiskā lietotāja saskarnes ģenerēšana lielā mērā ir atkarīga no ievades prasību precizitātes un specifikas. Sarežģītas vai neskaidras prasības var radīt problēmas ģenerēšanas algoritmiem, izraisot neprecizitātes vai neatbilstības ģenerētajos lietotāja saskarnes komponentos.

Daži rīki, piemēram, *Axure RP* un *Justinmind*, ļauj dizaineriem importēt prasību dokumentus vai lietotāju stāstus un ģenerēt lietotāja saskarnes tieši no šīm specifikācijām. Šis process parasti ietver saskarnes komponentu un mijiedarbību kartēšanas prasības.

Viens no interesantiem projektiem, kas atrodas izstrādē, ir mēģinājums pielietot dabīgās valodas apstrādi (angl. *Natural Language Processing, NLP*), lai automātiski transformēt prasību specifikāciju uz problēmvides specifisko valodu, kurā būtu aprakstīta lietotāju saskarne un tā navigācijas shēma (Kolthoff, 2019). Konceptuālo metodes shēmu var redzēt 1.14. attēlā.

Izveidoto DSL var tālāk pārveidot atbilstošajos mērķa platformas prototipos. Lielākā daļa saistīto sistēmu apstājas pēc artefaktu ģenerēšanas, kamēr šī metode piedāvātu inteliģentu un automātisku mijiedarbības mehānismu, kas lietotājiem ļauj iteratīvā veidā sniegt dabiskās valodas atsauksmes par ģenerētajiem prototipiem, kas attiecīgi tiks pārveidotas prototipa izmaiņās (Kolthoff, 2019).




**1.14. att. GUI-DSL kopējā darbības shēma (aizgūts no (Kolthoff, 2019))**

### *1.3.5.6. No lietošanas gadījumiem ģenerēta lietotāju saskarne*

Automātiskā lietotāja saskarnes ģenerēšana no lietošanas gadījumiem piedāvā gan priekšrocības, gan trūkumus – racionalizē izstrādes procesu, tieši pārvēršot funkcionālās prasības taustāmos lietotāja saskarnes elementos. Tas samazina manuālai projektēšanai un ieviešanai nepieciešamo laiku un pūles. Atvasinot lietotāja saskarnes komponentus tieši no lietošanas gadījumiem, izstrādātāji nodrošina konsekvenci starp paredzēto funkcionalitāti un lietotāja saskarni. Tas samazina neatbilstību vai pārpratumu iespējamību starp projektēšanas un ieviešanas posmiem.

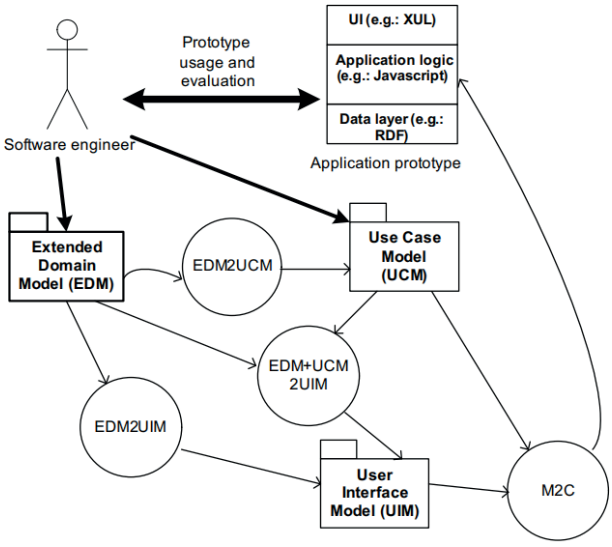
Tomēr, paļaujoties tikai uz lietošanas gadījumiem lietotāja saskarnes ģenerēšanai, var tikt ignorēti noteikti lietotāju mijiedarbības scenāriji, kas nav skaidri ietverti prasībās. Tas var radīt lietotāja saskarnes dizainu, kas neatbilst visām lietotāju vajadzībām.

Lietošanas gadījumi ne vienmēr var pilnībā aptvert noteiktas lietotāja saskarnes mijiedarbības sarežģītību, piemēram, daudzpakāpju darbplūsmas vai dinamiskas lietotāja ievades. Automātiskajai ģenerēšanai var rasties grūtības, lai šīs sarežģītības pārvērstu intuitīvās un lietotājam draudzīgās saskarnēs.

Programmatūras inženierijas kopienā izplatīta prakse ietver vienotas modelēšanas valodas (angl. *Unified Modelling Language, UML*) sistēmas modeļa izveidi, kas ietver problēmvides un lietošanas gadījumu modeļus, kas papildināti ar nefunkcionāliem lietotāja saskarnes prototipiem. Tomēr šiem modeļiem bieži trūkst integrācijas, un tie ir neskaidri un nepilnīgi, kas kavē automātisko validāciju (Cruz (da), Miguel et al., 2010a).

UML modelēšanas valoda pati par sevi nav risinājums programmatūras izstrādes problēmām – tas ir rīks kas nodrošina vietu vajadzīgās informācijas atspoguļošanai. Bet, kas ir jāatspoguļo, no kāda perspektīvas un kādā secībā jau ir konkrēta risinājuma vai metodikas rezultāti. Tādā veidā UML darbojas kā izteiksmes metode, kas līdzinās vārdiem latviešu valodā, bet nozīmes piešķiršanai nepieciešama grupēšana tāpat kā vārdi veido teikumus. Tātad UML ir efektīvs modelēšanas rīks, bet, lai izveidotu patiesi nozīmīgus un noderīgus modeļus, tas ir jāizmanto kopā ar pareizu pieeju (Kozačenko, 2014).

Modeļvadītas izstrādes pieejas, piemēram, problēmvides specifiskā modelēšana (angl. *Domain-Specific Modelling, DSM*) un modeļu vadītā arhitektūra (angl. *Model Driven Architecture, MDA*), balstās uz modeļu pilnveidošanu un automātisku koda un apakšmodeļu ģenerēšanu, tādēļ ir nepieciešamas nepārprotamas modeļu definīcijas (Cruz (da), Miguel et al., 2010a). Lai risinātu lietotāja saskarnes modeļu un prototipu ģenerēšanas problēmas, autori (Cruz (da), Miguel et al., 2010a) piedāvā pieeju formu lietojumprogrammu automātiskai ģenerēšanai modeļa vadītas programmatūras izstrādes veidā. Tas ietver iteratīvu problēmvides modeļa izstrādi, pēc izvēles lietošanas gadījuma modeļa, un automātiski ģenerētu izpildāmo prototipu testēšanu (Cruz (da), Miguel et al., 2010a).




**1.15. att. Vispārēja pieeja lietotāja saskarnes ģenerēšanai (aizgūts no (Cruz (da), Miguel et al., 2010a))**

Viena no efektīvākajām pieejām ir automātiska lietotāja saskarnes modeļu un prototipu ģenerēšana no problēmvides un lietošanas gadījumu modeļiem (Cruz (da), Miguel et al., 2010a) – ļauj automātiski ģenerēt lietotāja saskarnes modeļus (angl. *User Interface Model, UIM*) un izpildāmos lietotāja saskarnes prototipos (angl. *User Inteface Prototype, UIP*) no agrīniem, pakāpeniski bagātinātiem sistēmas modeļiem, kas vēl nav lietotāja saskarne (1.15. att.) (Cruz (da), Miguel et al., 2010a). Šajā pieejā tāpat kā daudzās citās, darbības tiek kartētas ar CRUD (angl. *Create, Read, Update, Delete*) aktivitātēm.

Zemāk tiek apskatītas dažādas pašreizējās modeļvadāmas pieejas lietotāja saskarnes prototipu vai lietotāja saskarnes modeļu automātiskai ģenerēšanai no sistēmas modeļiem, kas nav lietotāja saskarne.

1. XIS pieeja (da Silva, 2003): šī pieeja sadala sistēmas problēmas dažādos apakšmodeļos un ļauj ģenerēt UIM/UIP no modeļa uz modeli modeļvadītas izstrādes iestatījumā. Tas ietver fiktīvu un gudru pieeju, kas ļauj ģenerēt lietotāja saskarnes modeļus no citiem modeļiem.
2. OO-metodes pieeja (Molina, 2004; Molina, Meliá et al., 2002): šīs pieejas mērķis ir izveidot formālu programmatūras sistēmas specifikāciju izpildāmā formālā objektorientētā valodā. Tas sākas ar konceptuālu modeli, kas sastāv no objektu, dinamiskiem, funkcionāliem un prezentācijas modeļiem.
3. ZOOM pieeja (Jia, Steele et al., 2004): ZOOM nodrošina procesu, apzīmējumu un rīku kopu modeļa virzītai izstrādei ar atsevišķiem apzīmējumiem struktūrai, darbībai un lietotāja saskarnei. Tas ļauj grafiski formāli modelēt programmatūras sistēmu un integrēt dažādas daļas, izmantojot uz notikumiem balstītu sistēmu.
4. Citas pieejas (Cruz (da), Miguel et al., 2010a): Martinesa (Martinez, Estrada et al., 2002) pieejā lietotāja saskarnes atvasina no biznesa procesu modeļiem, Elkoutbi (Elkoutbi, Khriss et al., 2006) pieeja ir balstīta uz lietošanas scenārijiem, un Forbriga (Forbrig, Dittmar et al., 2004) pieeja modeļu pārveidošanai izmanto lietotāja saskarnes modeļus. Katrai pieejai ir savas stiprās puses un ierobežojumi.

### *1.3.5.7. Modeļvadamās izstrādes principu lietošanas secinājumi*

Var novērot esošo pieeju trūkumus, tostarp piepūle, kas nepieciešama modeļa izveidei, lietotāja saskarnes modeļu manuāla izveide, izpildāmu prototipu trūkums un ierobežojumi attiecībā uz klases stāvokļa ierobežojumu un trigeru apstrādi. Kopumā, lai gan šīs pieejas piedāvā dažādus risinājumus lietotāja saskarnes ģenerēšanai, tām ir trūkumi, kas kavē to efektivitāti programmatūras izstrādes komandās.

Automātiskā lietotāja saskarnes ģenerēšana no lietošanas gadījumiem ietver lietotāja mijiedarbības kartēšanu lietotāja saskarnes elementiem un darbplūsmām. Priekšrocības ietver lietotāja saskarnes dizaina pielāgošanu lietotāju vajadzībām, bet mīnusi var ietvert lietošanas gadījumu precīzas kartēšanas sarežģītību ar lietotāja saskarnes elementiem. Automātiska lietotāja saskarnes ģenerēšana no prasību specifikācijām nodrošina, ka lietotāja saskarne atbilst norādītajām funkcionālajām un nefunkcionālajām prasībām. Priekšrocības ietver prasību un lietotāja saskarnes dizaina konsekvences saglabāšanu, bet mīnusi var ietvert izaicinājumu precīzi interpretēt un pārvērst prasības lietotāja saskarnes elementos. Automātiskā lietotāja saskarnes ģenerēšana no maketiem ietver lietotāja saskarnes vizuālo attēlojumu pārveidošanu funkcionālos lietotāja saskarnes elementos. Priekšrocības ietver ātru prototipu izstrādi un vizuālo validāciju, bet mīnusi var ietvert ierobežojumus interaktīvu uzvedību un dinamisku elementu uztveršanā. Struktūrskices nodrošina lietotāja saskarnes izkārtojumu skeleta kontūras, atvieglojot automātisku lietotāja saskarnes ģenerēšanu, definējot strukturālos komponentus un navigācijas plūsmas. Priekšrocības ietver ātru lietotāja saskarnes struktūras vizualizāciju, bet mīnusi var ietvert detaļu trūkumu lietotāja saskarnes mijiedarbības un estētikas attēlošanā. Šabloni piedāvā iepriekš izstrādātus lietotāja saskarnes izkārtojumus un komponentus automātiskai lietotāja saskarnes ģenerēšanai. Priekšrocības ietver strauju attīstību un konsekvenci, bet mīnusi var ietvert ierobežotu pielāgošanu un vispārīgu vai atkārtotu dizainu potenciālu. Kopumā, automātiskā lietotāja saskarnes ģenerēšana no dažādiem avotiem piedāvā tādas priekšrocības kā efektivitāte, konsekvence un saskaņošana ar lietotāju vajadzībām, taču rada arī problēmas, kas saistītas ar ievades avotu interpretācijas un tulkošanas lietotāja saskarnes elementos sarežģītību, elastību un precizitāti.

Modeļvadāmā izstrāde ir piemērota automātiskai lietotāja saskarnes ģenerēšanai tās abstrakcijas dēļ, kas ļauj izstrādātājiem strādāt konceptuālā līmenī, nevis koncentrēties uz

ieviešanas detaļām (Nikiforova, Babris et al., 2020). Šī pieeja nodrošina konsekvenci, veicina automatizāciju, veicina komponentu un modeļu atkārtotu izmantošanu, atvieglo problēmu nošķiršanu un ļauj automātiski atjaunināt lietotāja saskarni, pamatojoties uz izmaiņām pamatā esošajos modeļos. Kopumā modeļvadāma izstrāde racionalizē lietotāja saskarnes izstrādi, uzlabo uzturēšanu un uzlabo lietotāja pieredzi, nodrošinot vienotu un efektīvu izstrādes procesu.

## 1.4. Nodaļas secinājumi un rezultāti

Esošo risinājumu analīze lietotāja saskarnes projektēšanā ļauj secināt, ka eksistējošie pētījumi skar tikai atsevišķus lietotāja saskarnes aspektus un dod iespēju daļēji ģenerēt lietotāja saskarnes prototipus. Šajā promocijas darbā ir izvirzīta hipotēze, ka lietotāja saskarnes automātiskajai prototipu ģenerācijai, pielietojot modeļvadāmas izstrādes principus, var pietikt ar divpusložu modeli kā avota modeli, kas saturēs nepieciešamo informāciju, lai veidotu lietotāja saskarnes prototipus. Lai pārbaudītu šo hipotēzi, pirmkārt ir jānoskaidro, kāds elementu saturs ir pieprasīts kā transformācijas rezultāts, ko lietotāja saskarnes projektētājs vēlas redzēt savā pusē kā rezultātu un pārliecināties vai divpusložu modelis satur visu informāciju, kas ir nepieciešama transformācijas likumu definēšanai.

Lai gan lietotāja saskarnes ģenerēšana ir ievērojami attīstījusies, pašreizējā stāvoklī tai joprojām pastāv dažas grūtības un ierobežojumi. Lietotāja saskarnes ģenerēšanas rīkiem var būt grūti tikt galā ar sarežģītām jaunāko saskarņu vajadzībām, piemēram, adaptīvu dizainu, vairāku platformu piemērotību un mainīgu saturu. Bieži dizaineriem ir jāiejaucas manuāli, lai risinātu ārkārtas situācijas un apstiprinātu, ka izveidotās saskarnes atbilst noteiktām lietotāja prasībām un lietojumprogrammu scenārijiem. Joprojām ir sarežģīti pārliecināties, ka automātiski izveidotās lietotāja saskarnes ir labas kvalitātes un tās var pareizi izmantot. Lai gan daudziem rīkiem ir iebūvētas validācijas pārbaudes un lietojamības noteikumi, joprojām ir vajadzīgas spēcīgas testēšanas un novērtēšanas metodes, lai uzzinātu par problēmām, kas saistītas ar lietošanas vienkāršību, pieejamības šķēršļiem, kā arī dizaina atšķirībām. Lietotāja saskarnes ģenerēšanas metodes, kurās tiek izmantots mākslīgais intelekts, bieži vien nepiedāvā caurspīdīgumu vai interpretējamību (Alfaridzi & Yulianti, 2020; Riccio, Jahangirova et al., 2020).

Uzticēšanās pieejai un spēja izprast automātiskās ģenerēšanas algoritmu rezultātus var būt šķērslis dizaineriem. Tā kā mākslīgā intelekta vadīta lietotāja saskarnes ģenerēšana kļūst arvien izplatītāka (Stige, Zamani et al., 2023), pieaug arī bažas par ētiku, piemēram, objektivitātes trūkuma algoritmos, privātuma riskiem un iespējamo cilvēku dizaineru nomaiņu. Dizaineriem un izstrādātājiem ir rūpīgi jādomā par automatizētas lietotāja saskarnes izveides morālajām un sociālajām sekām, kā arī jāatrod veidi, kā novērst jebkādu no tās radīto kaitējumu. Kad šie šķēršļi būs pārvarēti, automātiskā lietotāja saskarnes ģenerēšana var būtiski ietekmēt to, kā tiek izstrādāta programmatūra, jo dizaineriem būs vieglāk izveidot lietotāja saskarnes, kas ir efektīvākas un uz cilvēku centrētas.

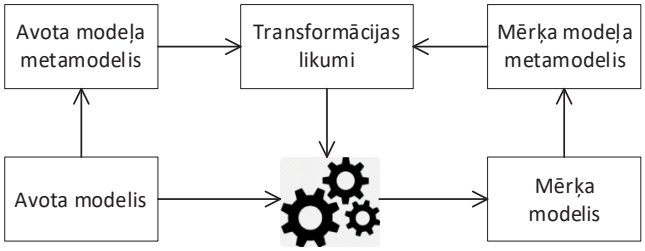
Kā arī apskatot zema koda platformas, kuru mērķis ir vienkāršot izveides procesu neprofesionāliem programmētājiem, izmantojot vizuālās saskarnes un gatavus elementus, modeļvadāmā inženierija piedāvā metodisku un formālu paņēmienu programmatūras projektēšanai. Šis veids ir labs, lai sistēmas sarežģījumus pārvērstu vispārīgākā formā un atvieglotu darbu, kas palīdz izstrādātājiem tikt galā ar programmatūras izmaiņām un tās attīstības raksturu

Automātiski ģenerētas lietotāja saskarnes kontekstā modeļvadāmai izstrādei ir vairākas priekšrocības salīdzinājumā ar zema koda izstrādes platformām. Izmantojot modeļvadāmo pieeju, ir iespējams izveidot precīzus un formālus modeļus gan lietotāja saskarnei, gan biznesa loģikai – tas nodrošina vienmērīgu kombināciju, kas atbilst iepriekš noteiktiem modeļiem un arhitektūras normām.

Nākamajā sadaļā ir aprakstīti lietotāja saskarnes elementi, lai būtu iespējams veidot mērķa modeli un mērķa metamodeli transformācijas likumu definēšanai, kā arī tiek aprakstīts divpusložu modelis un tā adaptācija transformācijas likumu definēšanai, lai rezultātā var iegūt lietotāja saskarnes prototipus.

# 2. MODEĻVADĀMAS IZSTRĀDES ELEMENTI PIEDĀVĀTAJĀ RISINĀJUMĀ

Analizējot saistītus pētījumus, kur ir piedāvāti dažādi risinājumi, kā var automatizēt lietotāja saskarnes projektēšanu, ir secināts, ka modeļi un metamodeļi ir pietiekams formālisms lietotāja saskarnes projektēšanā, jo lietotāja saskarnes elementu kopu var aprakstīt kā modeli (Escalona, Garcia-Borgonon et al., 2021), un līdz ar to pielietot modeļvadāmas izstrādes principus, kur pamata koncepti ir modeļi, metamodeļi un transformācijas likumi (skat. 2.1. attēlu)




**2.1. att. Modeļvadamās izstrādes konceptuālā shēma (adaptēts no (Kleppe, Warmer et al., 2003))**

Modeļvadāmas izstrādes pamatā ir programmatūras koda automātiskā ģenerēšana no problēmvides modeļiem, sākumā veidojot neatkarīgu no realizācijas platformas modeli, un pēc tam to transformējot platformas specifiskajā modelī un tālāk kodā. Šīs pieejas pamatā ir problēmvides analīze un modelēšana, meta modelēšana, modeļvadāmā koda ģenerēšana, šablona valodas, un problēmvidē balstītu ietvaru izstrāde (Völter & Bettin, 2004). Lietotāja saskarnes projektēšanas kontekstā ir izmantojami šādi modeļvadāmās izstrādes principi:

1. Tiek veidots modelis vai modeļi, kas apraksta lietotāja saskarnes scenārijus un satur pilnīgu un nepretrunīgu informāciju par problēmvides zināšanām un kontekstu, tā saucamo avota modeli.
2. Tiek savākta visa nepieciešamā informācija par lietotāja saskarnes formām, to saturu un pārejām starp tām, kas ir attēlota mērķa modelī.
3. Avota un mērķa modeļi dod iespēju definēt avota un mērķa metamodeļus, kas tiek izteikti, pielietojot UML vai kādu problēmvides specifisko valodu.

4. Tiek definēti transformācijas likumi, kas dod iespēju iegūt mērķa modeli, pielietojot avota modelim iepriekš definētus transformācijas likumus.

Transformācijas rezultāts var tikt izmantots tādā veidā kā tas ir iegūts pēc transformācijas likumu pielietošanas, vai manuāli papildināts. Lai transformācijas rezultāts būtu lietojams, tam ir jābūt palaižamam kodam vai importējamam kādā no vidēm, kurā ir iespējams transformācijas rezultātu apstrādāt. Lietotāja saskarnes kontekstā tas varētu būt kāds no lietotāja saskarnes projektēšanas rīkiem, piemēram, *T:XML* (Lopez-Jaquero, Montero, 2011), *Sparx Enterprise Architect* vai kādā citā.

## 2.1. Mērķa modeļa un metamodeļa izstrāde

Lai būtu iespējams pielietot modeļvadāmas inženierijas principus, automātiski ģenerētai lietotāja saskarnes izveidei, ir nepieciešama mērķa modeļa un metamodeļa veidošana, lai noteiktu, kāds būs lietotāja saskarnes saturs, struktūra, darbības un vizuālie elementi. Parasti process ir šāds – metamodelis darbojas kā formāls elementu, savienojumu un funkciju attēlojums, kas veido lietotāja saskarnes izstrādes procesu (Cruz (da), Miguel et al., 2010b). Tajā tiek iestatīta ievades modeļu struktūra un parādīts, kā tos pārveidot. Tas ietver paskaidrojumu sniegšanu par lietotāja saskarnes komponentiem, izkārtojumiem, navigācijas ceļiem, datu savienojumiem un stila izvēli.

Mērķa modelis ir tas, ko vēlamies iegūt lietotāja saskarnes izveides procesa rezultātā. Tajā ir paskaidrots, kā metamodelī norādītie elementi un līdzekļi tiks pārveidoti par īstiem lietotāja saskarnes artefaktiem. Mērķa modelī var būt informācija, piemēram, ekrāna izkārtojumi, logrīku izvietošanas vieta, to darbība, mijiedarbojoties ar tiem, un izvēlētie vizuālie stili.

Ir ļoti svarīgi izveidot savienojumu starp ievades modeļiem (piemēram, problēmvides modeļiem vai lietošanas gadījumu modeļiem) un metamodeļos definētajiem elementiem (Cruz (da), Miguel et al., 2010a). Savukārt, transformācijas process nosaka, kā informācija no ievades modeļiem tiek pārvērsta mērķa modelī, pārliecinoties, ka izveidotās lietotāja saskarnes precīzi attēlo plānotās funkcijas un lietotāju mijiedarbības.

Ar lietotājas saskarnes elementiem apzīmē dažādus komponentus, kas palīdz lietotājam mijiedarboties ar lietojumprogrammu. Pogas, teksta lauki, izvēlnes saraksti un citas vadīklas – visi šie ir lietotāja saskarnes komponenti, ko lietotāji izmanto, lai veiktu lietojumprogrammas datu

ievadi un izvadi kā arī mijiedarboties ar saturu. Lai precīzi attēlotu šos komponentus un saglabātu vēlamo lietotāja pieredzi (angl. *User Experience, UX*), mērķa modelim ir jāsatur precīza lietotāja saskarnes definīcija.

Galvenā loma lietotāja saskarnes izkārtojumam un struktūras organizēšanā ir tieši šiem komponentiem. Dažādi izkārtojuma konteineri, piemēram, režģi (angl. *Grids*), rāmji (angl. *Frames*) un paneļi – tas viss nosaka lietotāja saskarnes komponentu telpisko izvietojumu ekrānā. Metamodelī ir definēti likumi, kas regulē šo elementu izkārtojumu un izlīdzināšanu, lai nodrošinātu konsekvenci un saskaņotību lietotāja saskarnē (Liu & Ma, 2010).

Eksistē lietotāja saskarnes elementi, kuri tiek izmantoti lietotāja multivides, datu un vizualizāciju prezentēšanai – tabulas, diagrammas, grafiki, utt. Lai nodrošinātu datu korekti un efektīvu attēlošanu, šiem elementiem nepieciešams būt precīzi attēlotiem gan mērķa modelī, gan metamodelī (Lumertz, Ribeiro et al., 2016).

Lietotāja saskarnes elementus cenšas veidot, ņemot vērā lietotāja pieredzes principus, tas nepieciešams, lai nodrošinātu lietojumprogrammas intuitīvo un vieglo lietošanu kā arī lai lietojumprogramma būtu vizuāli pievilcīga (Kiruthika, Khaddaj et al., 2016). Mērķa modelī lietotāja saskarnes elementu atlase un izkārtojums nosaka kopējo lietotāja pieredzi, savukārt, metamodelis aptver šo dizainu un nodrošina, ka tas tiek saglabāts transformācijas procesā.

Lai izstrādātu lietotāja pieredzei atbilstošu saskarni, lietotāja saskarnes elementiem ir jāatbilst dizaina standartiem, definētajām stila vadlīnijām un pieejamības prasībām (Aizpurua, Harper et al., 2016). Likumus un ierobežojumus nepieciešams definēt, lai nodrošinātu konsekvenci lietotāja saskarnes dizainā un atbalstītu attiecīgos standartus – krāsu shēmas, tipogrāfiju u.c.

Nākamajās sadaļā ir apskatītas lietotāja saskarnes bibliotēkas un ir veikta saskarnes elementu kartēšana. Lietotajā saskarnē sagaidāmu elementu kopa ir modeļu transformācijas rezultātā sagaidāma koda satura kopu, kas pēc modeļvadāmas inženierijas principiem tiek definēta kā mērķa metamodelis un tiek izmantota transformācijas likumu definēšanā.

### 2.1.1. Vizuālo komponentu bibliotēkas un ietvari

Izmantojot konsekventu vizuālo vadlīniju kopumu, dizaineri var koncentrēties uz lietotāju pieredzes uzlabošanu un problēmu risināšana. Tas ļauj nodrošināt vienmērīgāku darbplūsmu un

ātrāka rezultātu. Šādam produktam ir daudz priekšrocību (Soares, Falcao et al., 2023) – izstrāde, kas iegūst atkārtoti lietojamu kodu, mērogojamu procesu, un pat samazina daudzas tehniskas problēmas. Standartizēti elementi tiek izmantoti saskaņoti, harmoniski un dod iespēju atkārtoti izveidot paredzamāku un vieglāk apgūstamu lietojumprogrammu (Hanelt, Bohnsack et al., 2021).

#### *2.1.1.1. Bibliotēka React.js*

*React.js*, ko bieži sauc tikai par *React*, ir ļoti spēcīga JavaScript bibliotēka, ko *Facebook* izveidoja lietotāja saskarņu izveidei. *React.js* ir visizplatītākais (Soares, Falcao et al., 2023) vienas lapas lietojumprogrammu (angl. *Single Page Application, SPA*) rīks. *React.js* ir kļuvusi nozīmīgāka salīdzinājumā ar citām sistēmām, piemēram, *Angular* un *Vue.js*, jo tā ir salīdzinoši maza apjoma un izmēra, salīdzinot ar citām konkurējošām sistēmām, kuru mērķis ir piedāvāt daudz vairāk (Singh, Srivastava et al., 2023). Tomēr *React.js* nodrošina vairākas izplatītas koda ērtības izstrādātājiem un piedāvā iespēju sadalīt lietojumprogrammu vairākos mazākos komponentos, kas apstrādā gan to stāvokli, gan stilu, tādējādi veicinot efektīvu koda atkārtotu izmantošanu (Javeed, 2019). Tādejādi, lietotāja saskarnes tiek veidotas no komponentiem, kurus var izmantot atkārtoti un turēt atsevišķi. *React.js* sintakse ir deklaratīva.

*React.js* unikāla iezīme ir tā virtuālajā DOM (angl. *Document Object Model*, *DOM*), kas nodrošina efektīvus lietotāja saskarnes atjauninājumus. Tas atkārtoti atveido tikai tos komponentus, kuros ir notikušas izmaiņas, tādējādi nodrošinot labāku veiktspēju un lietotāja pieredzi, jo īpaši apjomīgām lietojumprogrammām

*React.js* ir plaša ekosistēma un tajā ir pieejami tādi rīki kā *React Router*, kas apstrādā klienta puses maršrutēšanu, *Redux.js* lietojumprogrammu stāvokļa pārvaldībai, utt. React veids, kā strādāt ar komponentiem, veicina koda atkārtotu izmantošanu un modulāro dizainu, ļaujot izveidot lietojumprogrammas, kuras ir labi mērogotas un kuras ir viegli uzturēt. Turklāt *React.js* spēja veikt servera puses renderēšanu, kā arī tā pielāgošanās spēja nevainojami sadarboties ar dažādām bibliotēkām un ietvariem padara to daudzpusīgu modernu tīmekļa lietojumprogrammu veidošanā.

#### *2.1.1.2. Satvars Vue.js*

*React.js* ir rakstīts tīrā JavaScript valodā, kamēr *Vue.js* ir rakstīts ar JavaScript un HTML balstītas veidnes sintaksi. *React.js* ir bibliotēka, kas koncentrējas uz lietotāja saskarņu izveidi,

savukārt, *Vue.js* ir visaptverošs ietvars, kas nodrošina pilnīgu lietojumprogrammu izstrādes sistēmu.

Galvenā Vue.js priekšrocība ir tā vienkāršība, kas ļauj to bez grūtībām integrēt esošajos projektos. *Vue.js* struktūra, kuras pamatā ir komponenti, veicina koda atkārtotu izmantošanu un pārvaldību, ļaujot programmētājiem izveidot lietojumprogrammas modulārā veidā, kuras var efektīvi mērogot (Novac, Madar et al., 2021). *Vue.js* vietnei ir arī ļoti laba dokumentācija, kā arī aktīva sociālā kopiena, kas sniedz daudz resursu un palīdz izstrādātājiem (https://vuejs.org/). Viens no iespējamiem trūkumiem ir tas, ka *Vue.js* ekosistēma ir šaurāka nekā lielākiem ietvariem, piemēram, *React.js* vai *Angular*. Tas var samazināt trešo pušu spraudņu un rīku klāstu, kas ir pieejami *Vue.js*. Turklāt, lai gan *Vue.js* ir atzīts par savu lielo ātrumu, optimizācijas faktoru ziņā tas var nebūt tik piemērots ļoti liela mēroga lietojumprogrammām, salīdzinot ar citām sistēmām. Kopumā *Vue.js* kalpo kā pielāgojams un efektīvs satvars tīmekļa lapu izveidei, dēļ savas vienkāršības un jaudas, kas nodrošina līdzsvaru starp vieglumu, pielāgojamību un veiktspēju.

#### *2.1.1.3. Bibliotēka Angular*

Angular (https://angular.io/) ir strukturāls satvars dinamiskām tīmekļa lietotnēm, faktiski tā ir platforma un satvars vienas lapas klientu lietojumprogrammu veidošanai, izmantojot HTML un *TypeScript*.

*AngularJS*, ko bieži sauc vienkārši par *Angular*, ir visaptveroša JavaScript sistēma, ko atbalsta *Google*. *Angular* ir paredzēts, lai izveidotu tīmekļa lietotnes, kas ir aktīvas un var mainīties bez nepieciešamības atkārtoti ielādēt visu lapu katru reizi, kad tiek veiktas izmaiņas. *Angular* galvenā stiprā puse ir tā spēcīgā struktūra, kas palīdz izveidot vienas lapas lietojumprogrammas. *Angular* paplašina HTML ar vairāk atribūtiem un norādījumiem, tā automātiski sasaista modeli un skatu, izmantojot divvirzienu datu saistīšanu, samazinot vajadzību pēc tiešas DOM izmaiņas. Angular satvarā ir iekļauti daudzi iebūvēti moduļi, piemēram, maršrutēšana, formu apstrāde un HTTP pieprasījumi, kas palīdz vienkāršot izstrādes procesu un samazina paļaušanos uz ārējām bibliotēkām. Angular nodrošina arī labu atbalstu *TypeScript* — uzlabotai JavaScript versijai, kas palīdz uzlabot koda kvalitāti un padarīt izstrādātājus produktīvākus. Taču *Angular* apguves slieksnis var būt augsts, īpaši tiem, kas sāk no nulles. Tam ir sarežģītas idejas un metodes, kas sākotnēji varētu apgrūtināt to uztveršanu. Kā arī katra jauna *Angular* versija rada būtiskas izmaiņas,

kas var radīt zināmas grūtības, migrējot esošos projektus uz jaunākajiem Angular laidieniem. Kā arī *Angular* divvirzienu datu sasaistīšanas mehānisms var radīt papildu slodzi noteiktiem scenārijiem (piemēram, daudzu elementu renderēšanai). Neskatoties uz visām grūtībām, *Angular* joprojām ir iecienīts risinājums lieliem uzņēmumiem.

#### *2.1.1.4. Satvars Material-UI*

*Material-UI* ir *React.js* balstīta *Material Design* sistēma, kas palīdz tīmekļa lietotnes ieviešanai. *Material-UI* (https://mui.com/system/getting-started/) piedāvā lielu skaitu iebūvētu komponentu, kas ir uzreiz pieejamai izmantošanai.

*Material-UI* ir viens no populārākajiem (Soares, Falcao et al., 2023; Mukthapuram, 2021) *React.js* lietotāja saskarnes satvariem, ko izmanto daudzos atvērtā pirmkoda un komerciālos projektos. *Material-UI* tik populāru padara tas, ka tajā ir iekļautas divas atbilstošas tehnoloģijas – *Google Material Design* komponenti un *Facebook React.js*. Turklāt daudzi dizaineri pietiekami labi pārzin Material Design, lai izstrādātu konsekventu lietotāja saskarni, savukārt, daudzi izstrādātāji zina pietiekami daudz *React.js*, lai izveidotu lietotni, kas darbojas un apmierina gala lietotāja prasības. Kopumā *Material-UI* ir tilts starp *React.js* un *Material Design* (Boduch, 2019).

Līdz šim labākais veids, kā izveidot "atomu" kopu, bija rakstīt pielāgotu CSS. Skatoties no "atomu" projektēšanas perspektīvas, *Material-UI* ir koncentrējies uz atomu un molekulāro slāņu problēmu risināšanu. *Materiāl-UI* lietotāja saskarnes satvars nenodrošina augstu komponentu līmeņu izvēli (Elrom, 2021), tādēļ lietotāja saskarnes dizainerim ir jāizvēlas un jāsaliek patstāvīgi visas nepieciešamās daļas. Izvēloties komponentus un tos patstāvīgi pielietojot var rasties jautājumi par šo komponentu pielāgošanu.

Ja ir nepieciešams pielāgot *Material-UI* komponentus, piemēram, ar atsevišķu stila likumu, kuru nav iespējams izlaist, jo tas ir strukturāls, tad, iespējams, labākais veids ir izveidot jaunu komponentu, kuru var turpmāk lietot vairākkārtīgi. Kā cits risinājums, ko var apskatīt, ir statiskā CSS pieeja. Šajā risinājumā tiek izveidots vienu kopējs CSS fails, kurā ir visi klašu nosaukumi, kurus, iespējams, izmantot. Šādu pieeju izmanto tādi satvari kā *Tailwind CSS* un *Bootstrap CSS*. Kā otrs risinājums ir dinamiskā CSS pieeja, kur CSS likumus izveido nevis koda būvējuma (angl. *Build Time*), bet lietojumprogrammas izpildlaikā (angl. *Runtime*). Šādu pieeju izmanto arī *Material-UI*, kas tiek izvēlēts par demonstrēšanas satvaru. *Material-UI* dinamiskā CSS pieeja

izmanto metodes, kuras saņem *React.js* atribūtus un kā atbildi sniedz ar CSS bagātinātu komponentu. Šāda pieeja veicina lietotāja saskarnes konsekvenci, pat pielāgojot to un strādā ar faktiski visiem komponentiem.

#### *2.1.1.5. Satvars React Bootsrap*

*React Bootstrap* (https://react-bootstrap.netlify.app/) ir iepriekš izstrādātu lietotāja saskarnes komponentu kolekcija, kas paredzēta *React.js* lietotāja saskarņu izveidošanai. Lietošanai gatavie elementi, piemēram, pogas, formas, navigācijas joslas, utt., ir veidoti, izmantojot *Bootstrap* stilu un izkārtojuma sistēmu, taču tie darbojas kā *React.js* komponenti. Tas nozīmē, ka izstrādātāji savās *React.js* lietojumprogrammās var efektīvi izveidot vizuāli pievilcīgas un vienotas lietotāja saskarnes, neveidojot pielāgotus CSS vai HTML marķējumus. Tomēr, *React Bootstrap* izmantošanai ir arī daži trūkumi. Ne katru *React Bootstrap* komponentu elementu var pielāgot tik daudz, cik tā tīro *React.js* alternatīvu, kas var ierobežot elastību konkrētām dizaina vajadzībām. Kā *React Bootstrap* komponenti ir cieši saistīti ar *Bootstrap* CSS klasēm un JavaScript spraudņiem. Kā arī šie satvari var nesniegt pietiekamu elastību, un lietojumprogramma izskatīsies tāpat kā daudzas citas (Subramanian, 2017).

#### *2.1.1.6. "Atomu" projektēšanas ideja*

Lietotāja saskarnes izstrādē viena no lietojamākam saskarnes satura plānošanas metodēm ir "atomu" projektēšana (angl. *Atomic Design*), kas sastāv no pieciem posmiem, kuri darbojas kopā, lai izveidotu saskarnes projektēšanas sistēmu hierarhiskā veidā, sākot no atomiem, molekulām, organismiem un šabloniem (Matra, Kusumasari et al., 2023). Atomi ir visvienkāršākie nedalāmie komponenti, piemēram, pogas, formu lauki, virsraksti un ikonas, kas apvienojas veidojot molekulas, piemēram, formas ievade ar etiķeti, un pogu vai navigācijas joslu ar saitēm un logotipu. Molekulas tālāk apvienojas, veidojot organismus, kas veido šablonus un galu galā – lietotājam paredzētās lapas.

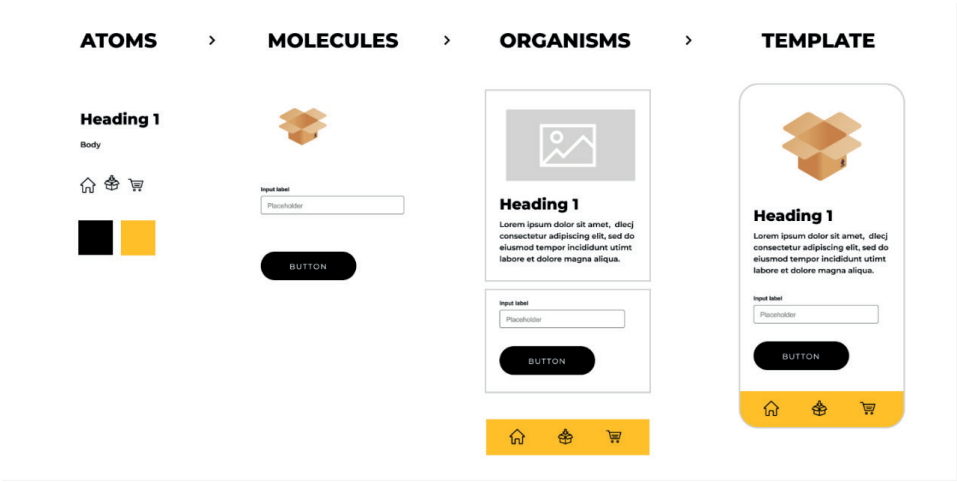
"Atomu" projektēšanu 2013. gadā Breds Frosts (angl. *Brad Frost*) prezentēja ar mērķi izstrādāt tīmekļa lietotnes izstrādes metodisku pieeju. Kombinējot lietotāja saskarnes šablonus ar "atomu" projektēšanas principiem, var veidot mērogojamus, modulārus un konsekventus lietotāja saskarnes sistēmu risinājumus – efektīvākai un vienmērīgākai lietotāja pieredzei visā

lietojumprogrammatūrā (Frost, 2016). Izmantojot lietotāja saskarnes kopā ar "atomu" projektēšanas principiem ietver sevī atkārtoti lietojamus lietotāja saskarnes komponentus.

Sākumā nepieciešams izprast "atomu" projektēšanas principus un tālāk – identificēt un definēt lietotāja saskarnes šablonus. Tie var būt gan pogas, formas, navigācijas elementi vai arī veseli procesi, kā piemēram, interneta veikala grozs, lietotāja reģistrēšanās, pieslēgšanās un atslēgšanās šabloni, kā arī citi.

Lai pielietotu "atomu" projektēšanu lietotāja saskarnes šablonos, nepieciešams tos savā starpā un attiecīgajā hierarhijā kartēt (2.2. att.):

1. Atomi – lietotāja saskarnes pamatelementi, tādi kā pogas, ievad lauki, ikonas, utt.
2. Molekulas – Atomu (skatīt augstāk) kombinācija, kas formē vienotu funkcionālu vienumu, kā piemēram, meklēšanas josla (kombinācija starp ievadlauku un pogu).
3. Organismi – Lielāki komponenti, kas satur sevī molekulu vai atomu grupu, kā piemēram, navigācijas izvēlni (kombinācija starp logotipu, navigācijas saitēm, utt.)
4. Veidnes – Augsta līmeņa struktūras, kas definē lapas izklājumu, iekļaujot sevī vairākus organismus vai molekulas.
5. Lapas – Faktiskās veidnes, aizpildīti ar reālu saturu.

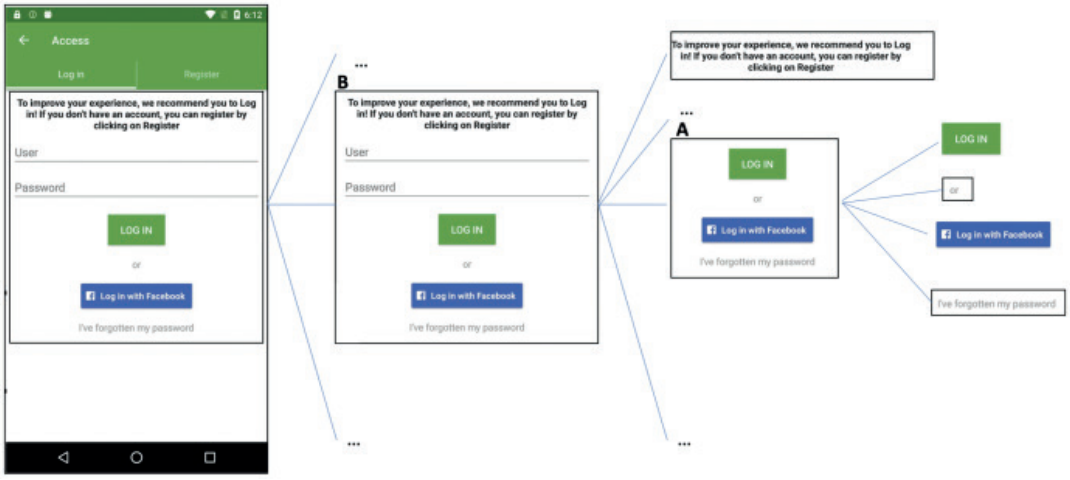



**2.2. att. "Atomu" projektējuma soļu piemērs (aizgūts no (McGinnity, 2021))**

Šī hierarhijas sistēma palīdz nodrošināt, ka dizains un izstrāde ir konsekventi, mērogojami un viegli uzturami. Izstrādātāji vai dizaineri var izveidot katru galveno komponentu ar dabisku pieejamību. Pēc to izveides šos komponentus var atkal izmantot arī pielāgošanas nolūkos – tas ļauj efektīvāk pārvaldīt pieejamību sarežģītās tīmekļa saskarnēs vai vietnēs.

Komponenti ar "atomu" projektēšanas un standartizētām pieejamības funkcijām palīdz izveidot vietnes saskarni, kurā cilvēki var ērti pārvietoties, jo izmanto ierastus elementus un maršrutus. Izmantojot šāda veida lietotāj saskarni, darbība vietnē kļūst pazīstama un paredzama pat tiem lietotājiem, kuri nav tipiski (piemēram, cilvēki ar invaliditāti). Tā kā pieejamības prasības laika gaitā mainās, tās var apstrādāt komponentu līmenī, pilnībā nepārveidojot vietni.

2.3. attēlā ir redzama mobilās lietotnes lietotāja saskarne un tai attiecīgais izkārtojuma grupēšanas koks. Sekcijā A ir zema līmeņa grupa, kurā atrodas tikai lietotāja saskarnes elementi, sekcijā B ir augstāka līmeņa grupa, kura iekļauj sekciju A. Kopējais lietotāja saskarnes skatījums ir galvenais mezgls.




**2.3. att. Semantiskas grupēšanas definīcija (aizgūts no (Duan, Hartmann et al., 2023))**

Lietotāja saskarnes šablonu pielietošanai izmantojot "atomu" projektēšanas pieeju sniedz dažādas priekšrocības, bet rada arī dažādus izaicinājumus. Galvenās priekšrocības ir modularitāte un atkārtotas izmantošanas iespējas ne tikai konkrēta projekta ietvaros. Kombinējot lietotāja saskarnes šablonus ar "atomu" projektēšanu – šāda pieeja atļauj dizaineriem un izstrādātājiem

viegli izmantot elementus dažādos projekta segmentos, kā arī starp projektiem. Kā nākamie ieguvumi ir konsekvence – tā uzlabo lietotāja pieredzi un padara lietotnes navigāciju intuitīvu. Elastība ļauj izstrādātājiem izmantot dažādus elementus dažādās projekta vietās veidojot lietotāja prasībām piemērotus izkārtojumus. Izstrādātāji var pievienot jaunas funkcijas vai modificēt esošās, vienkārši apvienojot un atkārtoti izmantojot esošos lietotāja saskarnes komponentus, kas padara šo pieeju par viegli mērogojamu. Šāda strukturēta pieeja padara risinājumu gana efektīvu.

Atsevišķi (atomiskie) elementi un/vai veidojošie bloki balstās viens uz otru, un tos var elastīgi salikt, veidojot tipu (konkrēts tipa gadījums ir objekts). Katru tipu vai objektu var izstrādāt piecos posmos – no mazākā iespējamā elementa līdz vispārējai veidnei un reālam, unikālam objektam (individuālam eksemplāram) (Nagel, 2015). Skatīt 2.1 tabulā.

**2.1. tabula**

**Piecu pakāpju bloku veidošanas princips (adaptēts no (Nagel, 2015))**

| | **Elements** | **Komponents** | **Segments** | **Tips** | **Objekts** |
|---|---|---|---|---|---|
| | Mazākais vienums | Mazāko vienumu kombinācija | | Struktūra vispārēja | Struktūra konkrēta |
| **Saturs** | Nosaukums, apakšnosaukums, apraksts, norāde, datums, attēls, meta dati, utt. | Satura modulis<br><br>Attēls un nosaukums, Cena un poga, | Moduļu grupa<br><br>Teksta sadaļa, rindkopa | Satura struktūrskice<br><br>Raksts, produkta specifikācija | Reāls saturs<br><br>Informācijas objekts |
| **Lietotāja saskarne** | Etiķete, ievades lauks, poga | Meklēšanas forma (satur etiķeti, ievades lauku un pogu) | Izkārtojuma laukums (galvene ar meklēšanas joslu, logo, navigāciju) | Lietotāja saskarnes šablons | Reāla lapa<br><br>Šablona eksemplārs |
| **Atomu projektējums** | Atoms | Molekula | Organisms | Šablons | Lapa |

Tomēr, šādai pieejai ir arī savi trūkumi un izaicinājumi. Pirmkārt, ir nepieciešamas zināšanas un izpratne par "atomu" projektējumu un tā pielietošanu lietotāja saskarnes šablonos. Pārvaldīt liela apjomu, ar "atomu" projektējumu elementu un lietotāja saskarnes šablonu, kartējumiem var būt sarežģīti, īpaši, lielāka mēroga lietojumprogrammās. Šādai pieejai ir jābūt efektīvi organizētai un dokumentētai, kas var prasīt papildus piepūli un pastāvīgu uzturēšanu.

### 2.1.2. Lietotāja saskarnes elementu kartēšana

Viena no darbā izstrādātās pieejas svarīgākajām daļām ir identificējamo lietotāja saskarņu elementu kataloga definēšana. Tā var tikt definēta vienu reizi un atkārtoti izmantojama turpmākajā izstrādē (Morgado & Paiva, 2015). Lietotāja saskarnes elementu kartēšana ir veidota apkopojot *React Bootstrap*, *Material-UI* un *AntDesign* elementos ietvertos līdzvērtīgus komponentus, kas ir identificējami šajās lietotāja saskarnes bibliotēkās un to attiecīgu aizstāšanu (skatīt 2.2. tabulu). *React Bootstrap*, *Material-UI* un *AntDesign* ir trīs populārākās lietotāju saskarnes bibliotēkas, kas vairākos pētījumos (Soares, Falcao et al., 2023; Javeed, 2019; Elrom, 2021; Subramanian, 2017), un, kas šajā kontekstā ir svarīgi, arī izstrādātāju blogos ir atzītas par lietotākām un pilnākām bibliotēkām lietotāja saskarnes projektēšanā. Šādi var uzticēties tam, ka šajās bibliotēkās ir pārstāvēts maksimālais lietotāja saskarnes elementu klāsts, kas var kalpot par pamatu lietotāja saskarnes metamodeļa veidošanā.

**2.2. tabula**

**Lietotāja saskarnes elementu kartēšana**

| **Lietotāja saskarnes elementa tips** | **React Bootstrap** | **Material-UI** | **AntDesign** |
|---|---|---|---|
| **Ievade** | | | |
| Checkbox | Form.Check (checkbox) | Checkbox | Checkbox |
| Radio button | Form.Check (radio) | Switch, Radio group | Switch, Radio |
| Dropdown list | Form.Select, Dropdown button | Autocomplete, Select | Autocomplete, Select, Dropdown |
| List boxes | - | Transfer list | Transfer |
| Buttons | Button, Button Group | Button, Button Group | Button |
| Toggles | Form.Check (switch) | Toggle button | Switch |
| Text field | Form.text, Form.control (Textarea) | Text field, Textarea Autosize | Input |

| **Lietotāja saskarnes elementa tips** | **React Bootstrap** | **Material-UI** | **AntDesign** |
|---|---|---|---|
| Numberic field | Form.Control (Number) | NumberInput | InputNumber |
| Date picker | - | Date picker, Date range picker | DatePicker, RangePicker |
| Time picker | - | Time picker | DatePicker, RangePicker |
| Slider | Form.Range | Slider, Rating | Slider, Rate |
| **Navigācija** | | | |
| Search field | - | Autocomplete | Autocomplete |
| Breadcrumb | Breadcrumb | Breadcrumb, Stepper, | Breadcrumb, Steps |
| Pagination | Pagination | Pagination, Table Pagination | Pagination |
| Icons | - | Material Icons | AntDesign Icons |
| Menus | Nav, Nav Bar, Offcanvas, | App Bar, Menu, Speed Dial, Drawer, Bottom Navigation | Menu |
| Tabs | Tabs | Tabs | Tabs |
| **Atgriezeniskā saite** | | | |
| Notifications | Alert, | Alert, Snackbar | Notification, Popconfirm |
| Progress bar | Progress, Spinner | Backdrop, Progress, Skeleton | Progress, Skeleton |
| Tooltip | Popover, Overlay, Tooltip | Popover, Popper | Popover, Tooltip |
| Message box | Toast | Dialog | Message |
| Modal window (popup) | Modal | Dialog, Modal | Modal, Popconfirm |

| Lietotāja saskarnes elementa tips | React Bootstrap | Material-UI | AntDesign |
|---|---|---|---|
| **Satura konteineri** | | | |
| Accordion | Accordion, List Group, Stack | Accordion, Stack | |
| Container | Container, Row, Col | Paper, Box, Container, Grid | Grid |
| **Datu attēlojums** | | | |
| Data table | Table | Table, Data Grid (MUI X) | Table |
| Cards | Card, Placeholder | Card | Card |

Visām problēmām, kas saistītas ar dizainu, *Material-UI* ir noteiktas vadlīnijas un noteikumi, kas ir piemērojami. No otras puses, *Material-UI* visaptverošais raksturs var ietvert dažas sarežģītas specifikācijas, kuras iesācējiem var būt grūti saprast. *Material-UI* lietotāja saskarnes satvars var nebūt intuitīvs – izstrādātājiem ļoti svarīga ir intuitīva projektēšana. Tomēr, ja *Material-UI* tiek izmantots automātiskā lietotāja saskarnes ģenerēšanā, izmantojot lietotāja saskarnes šablonus, šie trūkumi var tikt mazināti.

### 2.1.3. Lietotāja saskarnes elementu metamodelis

Lai izveidotu lietotāja saskarnes elementu metamodeli, nepieciešams definēt dažādus lietotāja saskarnē izmantoto elementu struktūras – atribūti, savstarpējās elementu attiecības un ierobežojumus. Jāsāk ar lietojumprogrammās kopējo izmantoto lietotāja saskarnes elementu identificēšanu. Piemēram, pogas, teksta laukus, izvēles rūtiņas, radio pogas, izkrītošās izvēlnes, utt.

Kā nākamais solis ir atribūtu noteikšana, kas katru lietotāja saskarnes elementu raksturo. Piemēram, tādas īpašības kā nosaukums, etiķete, veids, izmērs, pozīcija, redzamība, utt. Neskaitot atribūtu definēšanu, jānosaka arī attiecības starp dažādiem lietotāja saskarnes elementiem. Piemēram, formā var būt vairāki ievades lauki, pogas un starp tiem izveidojas vecākelementu un bērnelementu attiecības, kas arī ir jāiekļauj metamodeļa definīcijā.

Atkarībā no metamodeļa precizitātes, nepieciešams definēt ierobežojumus un likumus, kas nosaka kā lietotājs redz un mijiedarbojas ar lietotnes saskarnes elementiem. Piemēram, izvēles rūtiņai var būt ierobežojumi, kas saistīti ar noklusējuma stāvokli vai validācijas likumiem.

Metamodeļa izveidei nepieciešams izmantot kādu no modelēšanas valodām, piemēram, UML (*Unified Modelling Language*) vai problēmvidei specifisku valodu (angl. *Domain Specific Language, DSL*). Tādā veidā iespējams izmantot klases, lai attēlotu lietotāju saskarnes elementus un to atribūtus (piemēram, ievades laukus) un to atribūtus (piemēram, ievades lauka tips), asociācijas, lai apzīmētu attiecības un vajadzības gadījumā likumus un ierobežojumus.

Lietotāja saskarnes metamodeļa saturs lietotāja saskarnes prototipu ģenerēšanai no informācijas par IT risinājumu problēmvidei var atšķirties atkarībā no sistēmas īpašajām prasībām un īpašībām. Tomēr šeit ir daži izplatīti elementi, kas var tikt iekļauti lietotāja saskarnes metamodelī:

1. Problēmvides entītiju un atribūtu attēlojumi, kas jāparāda vai jāmaina lietotāja saskarnē. Tas var ietvert datu tipus, validācijas noteikumus un attiecības starp entītijām.
2. Dažādu ekrānu vai lapu apraksti, kas veido lietotāja saskarni. Tas var ietvert informāciju par izkārtojumu, navigācijas plūsmu un vispārējo lietotāja saskarnes struktūru.
3. Dažādu lietotāja saskarnes komponentu, piemēram, pogu, ievades lauku, izvēles rūtiņu, radio pogu, nolaižamo sarakstu, tabulu un konteineru definīcijas. Katram komponentam var būt tādas īpašības kā izmērs, pozīcija, stils un uzvedība.
4. Lietotāju darbību (piemēram, klikšķu, ievades) un atbilstošo notikumu (piemēram, formu iesniegšanas, datu atjaunināJumu) specifikācijas, kas aktivizē lietotāja saskarnes izmaiņas. Tas var ietvert notikumu apdarinātājus (angl. *Event Handlers*), pārejas un animācijas.
5. Lietotāja saskarnes komponentu kārtošanai šablonos un kārtošanai ekrānos vai lapās. Tas var ietvert uz režģi balstītus (angl. *Grid-based*) izkārtojumus, adaptīvus dizainus un atkārtoti lietojamus šablonus parastajiem lietotāja saskarnes modeļiem.
6. Lietotāja saskarne izmantoto vizuālo stilu, motīvu un zīmola elementu (piemēram, krāsu, fontu, ikonu) definīcijas. Tas nodrošina konsekventu un vienotu prezentāciju visā lietojumprogrammā.

7. Apsvērumi, lai padarītu lietotāja saskarni pieejamu lietotājiem ar invaliditāti un pielāgojamu dažādām valodām un kultūras vēlmēm. Tas var ietvert atbalstu ekrāna lasītājiem, tastatūras navigācijai un lokalizācijai.
8. Lietotāju lomu un atļauju specifikācijas, kas nosaka piekļuves tiesības un privilēģijas lietotāja saskarnē. Tas var ietvert uz lomām balstītas drošības politikas, autorizācijas noteikumus un lietotāja saskarnes pielāgošanas opcijas.
9. Lietotāju mijiedarbības un darbplūsmu apraksti kā arī secību diagrammas, plūsmu diagrammas un stāvokļa diagrammas, palīdz lietotājiem veikt uzdevumus un procesus lietojumprogrammā, kā arī attēlo dažādus lietošanas scenārijus.
10. Integrācijas punktu specifikācijas ar citām sistēmām vai pakalpojumiem, piemēram, API, tīmekļa pakalpojumiem vai ārējiem datu avotiem. Tas nodrošina netraucētu mijiedarbību un datu apmaiņu starp lietotāja saskarni un aizmugursistēmām.

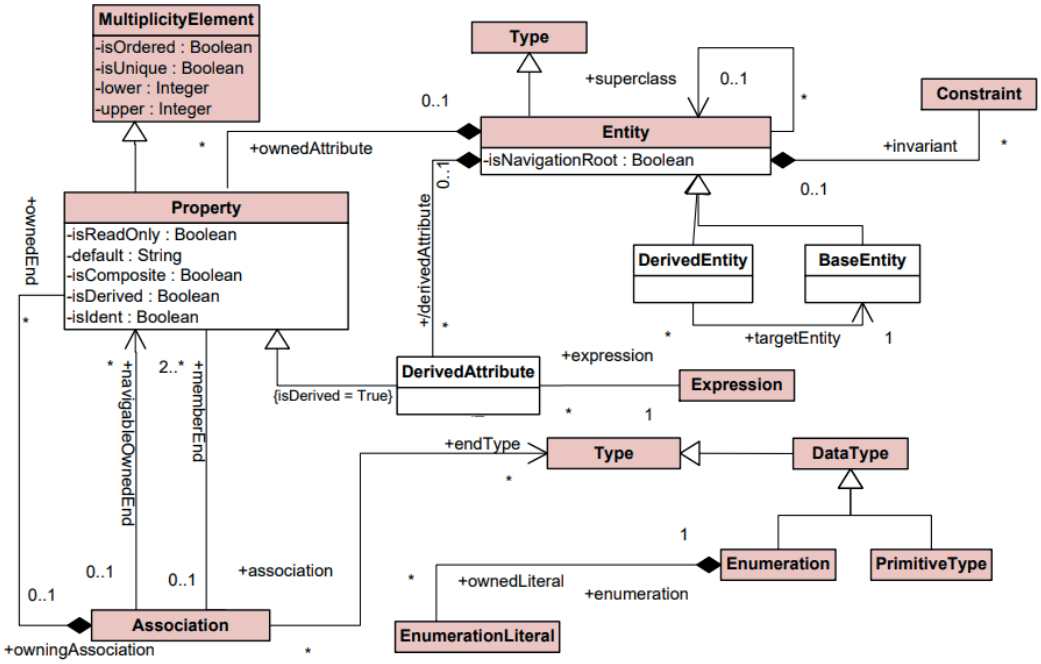
Lietotāja saskarnes metamodelis kalpo kā formāls lietotāja saskarnes struktūras, uzvedības un izskata attēlojums, nodrošinot pamatu automatizētai lietotāja saskarnes prototipu ģenerēšanai un nodrošinot iegūto dizainu konsekvenci un saskaņotību. Esošā darba gadījumā ir procesa modelis ar darbībām (angl. *Actions*) un notikumiem (angl. *Events*) un konceptu modelis ar entītijām un atribūtiem – viss, ko var iegūt no problēmvides

Lietotāja saskarnes metamodeļus var uzskatīt par lietotāja saskarņu melnrakstiem. Tie nosaka pamata veidošanas blokus, kas ir lietotāja saskarnes komponenti. Piemēram (līdzīgi kā 2.4. attēlā), šie komponenti var būt pogas, teksta lauki, izvēlnes vai jebkurš cits elements, ko var redzēt ekrānā.

Īpašības (angl. *Properties*) ir katra komponenta iezīmes. Piemēram, uz pogas parādītais teksts vai nolaižamajā izvēlnē uzskaitītie vienumi var būt īpašības. Metamodelis norāda, kādas īpašības var būt katram komponentam. Metamodeļi nosaka lietotāja saskarnes komponentu īpašības un darbību, nodrošinot konsekventu izskatu un darbību visā lietojumprogrammā. Tomēr, metamodeļi var kļūt pārāk specifiski, ierobežojot izstrādātāju iespējas ieviest unikālus lietotāja saskarnes elementus vai mijiedarbības, kas neietilpst iepriekš noteiktā tvērumā. Tas varētu ierobežot radošumu un kavēt novatoriskas lietotāja saskarnes dizaina pieejas. Kā arī visaptveroša metamodeļa definēšana sarežģītai lietotāja saskarnei (kurā ir daudz komponentu un tai ir sarežģīta mijiedarbība) var būt sarežģīts uzdevums. Tādā gadījumā paša metamodeļa sarežģītība var kļūt par

apgrūtinājumu tā pārvaldīšanai un uzturēšanai. Bieži vien lietotāja saskarnes metamodeļu priekšrocības ir atkarīgas no specializētiem rīkiem, kas izprot konkrēto izmantoto metamodeli. Ja šie rīki ir ierobežoti vai nav pieejami, metamodeļa izmantošana kļūst mazāk efektīva.

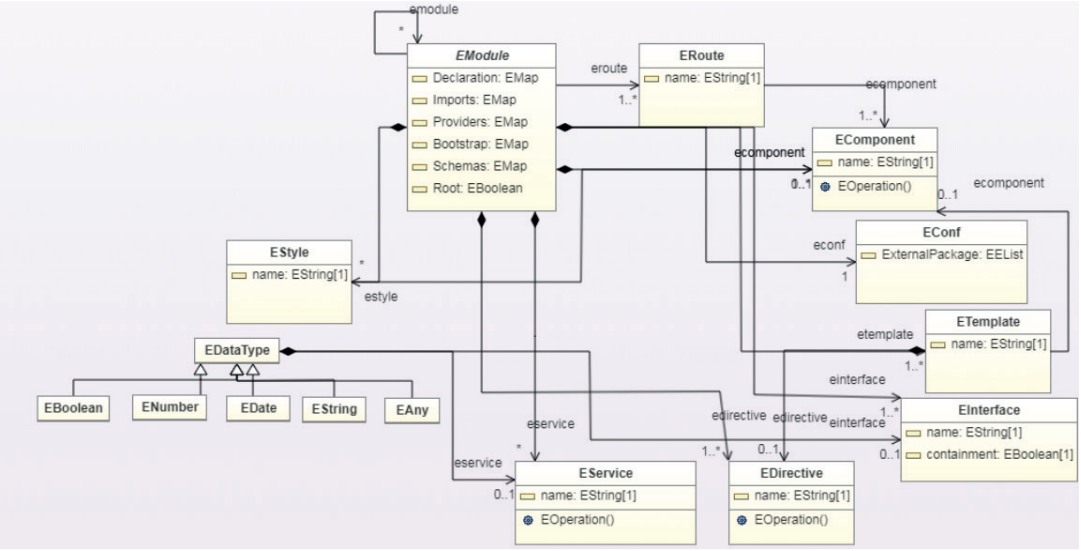
Vienā no pētījumiem (Cruz (da), Miguel et al., 2010b) tiek secināts, ka metamodeļa efektivitāte ir atkarīga no skaidriem procesu nosaukumiem, labi definētas lietotāja saskarnes šablonu bibliotēkas un, iespējams, papildu konteksta informācijas (2.4. attēlā ir redzams piemērs problēmvides modeļa metamodelim).




**2.4. Att. Problēmvides modeļa metamodelis (strukturālās iezīmes) (aizgūts no (Cruz (da), Miguel et al., 2010b))**

Citā pētījumā (Omari, Erramdani et al., 2020), kur tiek piedāvāta metodi Angular 7 lietojumprogrammu ģenerēšanai, izmantojot modeļvadāmu pieeju ar UML, tiek secināts, ka palielinās sākotnējās pūles UML modeļu izveidē un uzturēšanā, īpaši sarežģītām lietojumprogrammām. Kā arī metode ir atkarīga no koda ģenerēšanas rīka brieduma un

efektivitātes (metamodeli skatīt 2.5. attēlā). Lietotāja saskarnes metamodeļi piedāvā pieeju, lai nodrošinātu konsekvenci, samazinātu kļūdas un, iespējams, automatizētu lietotāja saskarnes izstrādi. Tomēr ir ļoti svarīgi atrast līdzsvaru starp specifiku un elastību. Pārāk specifiski metamodeļi var kavēt radošumu un radīt sarežģītību. Metamodeļa izmantošanas izvēlei un tā detalizācijas pakāpei jābūt balstītai uz lietotāja saskarnes dizaina projekta īpašajām vajadzībām.

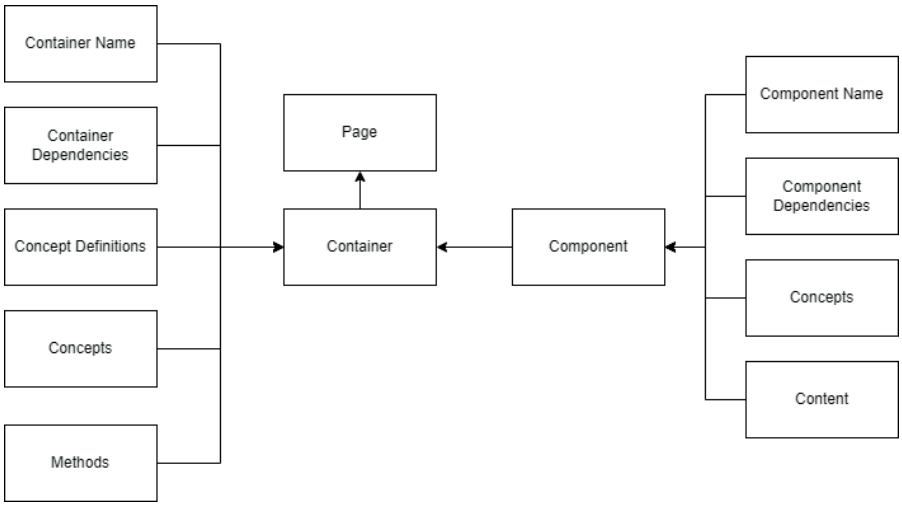



**2.5. att. Angular 7 metamodelis (aizgūts no (Omari, Erramdani et al., 2020))**

Ir gadījumi, kad lietotāja saskarnes metamodeļa izmantošana var nebūt labākā pieeja – piemēram, gadījumos, ja tiek veidota lietotāja saskarne ar unikālu mijiedarbību vai vizuāliem elementiem, kas vēl nav standartizēti. Parasti metamodeļi tiek veidoti, pamatojoties uz esošajiem lietotāja saskarnes modeļiem un komponentiem. Jauns dizains var slikti iekļauties šajās iepriekš definētajās struktūrās, iespējams, bloķējot ceļu radošumam un novitātei. Gadījumos, kad projekts atrodas lietotāja saskarnes izstrādes sākumā, ātri izmēģinot idejas un aplūkojot dažādas iespējas.

2.6. att. ir parādīts lietotāja saskarnes metamodelis *React.js* / *Redux.js* tīmekļa lietotnes organizēšanai automātiskas lietotāja saskarnes izveidei. Metamodelis koncentrējas uz to, kā lietotāja saskarnes konteineri un komponenti ir saistīti un atkarīgi viens no otra.

Šis metamodelis parāda, kādi elementi ir sagaidāmi risinājuma rezultātā, lai automātiski izveidotu *React.js* / *Redux.js* tīmekļa lietotni, nodalot funkcionalitāti konteineros un komponentos, kā arī *Redux.js* iekļaušana datu pārvaldīšanai un *XState* – procesu organizēšanai. Šāda pieeja piedāvā elastīgu un mērogojamu struktūru lietotāja saskarnes automātiskai ģenerēšanai.




**2.6. att. Lietotāja saskarnes metamodelis izstrādājamā risinājuma ietvaros**

Metamodelī tīmekļa lietotne tiek sadalīta lapās (angl. *Pages*), konteineros un komponentos. Lapas sastāv no konteineriem, kas apstrādā esošo procesa stāvokli, definē konceptus un metodes. Komponenti apzīmē atsevišķus lietotāja saskarnes elementus, kas saņem datus un metodes no saviem konteineriem, izmantojot īpašības (angl. *Properties*). Konteiners ir definēts ar nosaukumu, atkarību sarakstu, koncepta definīcijas sadaļu kurā ir arī mainīgie un metodes. Tādā pašā veidā tiek definēti komponenti, kuriem ir nosaukumi, atkarības, koncepti un pats galvenais – saturs, kas satur sevī izskatu un darbību. Savienojumi starp lapām, konteineriem un komponentiem nodrošina, ka viss ir iestatīts modulāri, lai atvieglotu sistēmas uzturēšanu. Šis metamodelis palīdz pārveidot biznesa procesu modeļus funkcionālās lietotāja saskarnēs, kartējot entītijas ar *React.js* komponentiem. Šī strukturētā sistēma palīdz viegli izveidot dinamiskas lietotāja saskarnes ar stāvokļiem, saglabājot skaidru pienākumu sadali un palielinot elastību.

Lietotāja saskarnes metamodeļi ir ļoti noderīgi projektiem, kuriem nepieciešama konsekvence, mazāk kļūdu un, iespējams, automatizēta lietotāja saskarnes izveide. Taču ļoti novatoriskas konstrukcijas vai situācijās, kad svarīga ir ātra prototipu izveide, nav pieejami daudz resursu vai ir nepieciešamas vienkāršas lietotāja saskarnes – citas metodes varētu būt efektīvākas un elastīgākas. Lietotāja saskarnes metamodeļa izmantošana ir izvēle, kas balstās uz konkrētajām projekta vajadzībām un tā izstrādes iestatījumiem

## 2.2. Avota modeļa izvēles pamatojums

Divpusložu modeļvadāma pieeja (Nikiforova, 2009) ir viena no modeļvadāmas programmatūras izstrādes pieejām, kas tiek izstrādāta un attīstīta Rīgas Tehniskajā Universitātē profesores Oksanas Ņikiforovas vadībā un pirmo reizi bija publicēta 2004. gadā (Nikiforova & Kirikova, 2004). Atšķirībā no citām modeļvadāmām pieejām, kas bija attīstītas tajā laikā (Nikiforova, Kozacenko et al., 2015), divpusložu modeļvadāmā pieeja piedāvāja balstīt programmatūras izstrādi uz diviem saistītiem savā starpā modeļiem, proti, procesu modeli, kas aprakstīja problēmvides funkcionēšanu, un konceptu modeli, kas attēloja problēmvidē esošā datu struktūras. Un tieši sasaiste starp šiem diviem modeļiem nodrošināja automātisku problēmu analīzes informāciju transformēt programmatūras sistēmas projektēšanas modeļos, kas bija veidoti dažādu UML diagrammu veidā. Šī transformācija, tādējādi piedāvāja plašāku projektēšanas artefaktu kopumu turpmākai koda ģenerēšanai uzreiz no problēmvides modeļiem. Šajā gadījumā, pirmkārt, tiek samazināts laika un cilvēkresursu patēriņš. Divpusložu modeļvadāma pieeja bija aprobēta vairākās problēmvidēs (Nikiforova, Sukovskis et al., 2015; Nikiforova, el Marzouki et al., 2017; Nikiforova & Gusarovs, 2020; Nikiforova, Iacono et al., 2020) un pielietota vairākos promocijas darbos kā pamatojuma formālisms (Pavlova, 2008; Gusarovs, 2020; Marzuoki (el), 2021).

Iepriekšējos pētījumos ir pierādīts, ka programmatūras sistēmas izstrādei nepieciešamo elementu kopas ģenerēšana no problēmvides modeļa, pamatojoties uz formāli definētām transformācijām, ļauj samazināt "manuālo" programmatūras klašu projektēšanas riskus (Grundspenkis, 1997). Šie riski ir saistīti ar informācijas zudumu, nekonsekvenci un dublēšanos. Kā arī divpusložu modeļvadāma pieeja ir atbalstīta ar *BrainTool* rīku, kurš atbalsta divpusložu

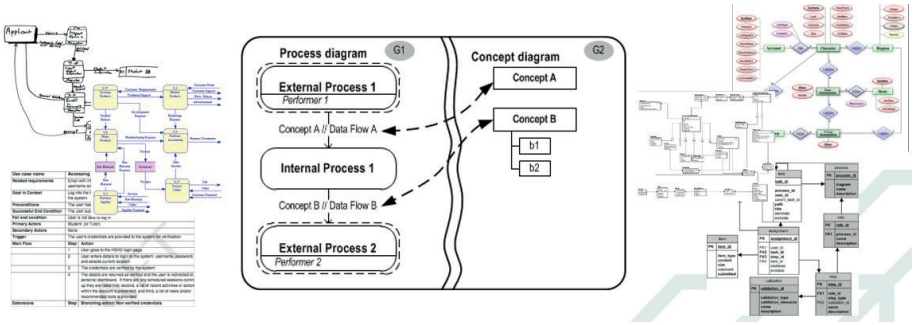
modeļa veidošanu, tā validāciju un eksportu XML formātā turpmākai lietošanai transformāciju likumos. Kā arī *BrainTool* rīkā ir definēti transformācijas likumi un to redaktors, kas dod iespēju pārveidot divpusložu modeli UML lietošanas gadījumu, klašu, secību un stāvokļu diagrammās un attiecīgajos eksportējamos XML failos importam UML modelēšanas rīkos.

Gusarova promocijas darbā (Gusarovs, 2020), kurā bija piedāvāta aizstāvēšanai un pieradīta tēze, ka, ja no divpusložu modeļa var ģenerēt UML diagrammas, no kurām tālāk var ģenerēt programmatūras pirmkodu, tad no divpusložu modeļa var uzreiz ģenerēt attiecīgo pirmkodu (ko arī var apskatīt kā modeli) ar nosacījumu, ka ir korekti uzdoti modeļu transformācijas likumi (Gusarovs, 2020), dod pamatojumu arī lietotāja saskarnes ģenerēšanai izvēlēties divpusložu modeļa formālismu kā avota modeli un izmantot tā metamodeli transformācijas likumu definēšanā.

Nākamajās sadaļās ir aprakstīta divpusložu modeļa būtība, ir pamatota iespēja papildināt divpusložu modeļa notāciju ar saskarnes elementu ģenerēšanai nepieciešamiem elementiem un ir izveidots divpusložu modeļa bagātināts metamodelis, kas turpmāk var kalpot transformācijas likumu definēšanai.

### 2.2.1. Divpusložu modeļa apraksts

Divpusložu modeļvadāmā pieeja ir programmatūras izstrādes metodoloģija, kas apvieno divus modelēšanas aspektus: procesa modelis attēlo problēmvides dinamiskos aspektus, izmantojot datu plūsmas diagrammai (angl. *Data Flow Diagram, DFD*) līdzīgu apzīmējumu, un konceptuālais modelis parāda sistēmas strukturālos aspektus, izmantojot apzīmējumus, kas ir līdzīgi entītijas-attiecību (ER) diagrammai (angl. *ER diagram*), bet bez attiecībām (Nikiforova, 2009).




**2.7. att. Divpusložu modeļa notācija (aizgūts no (Nikiforova, 2009))**

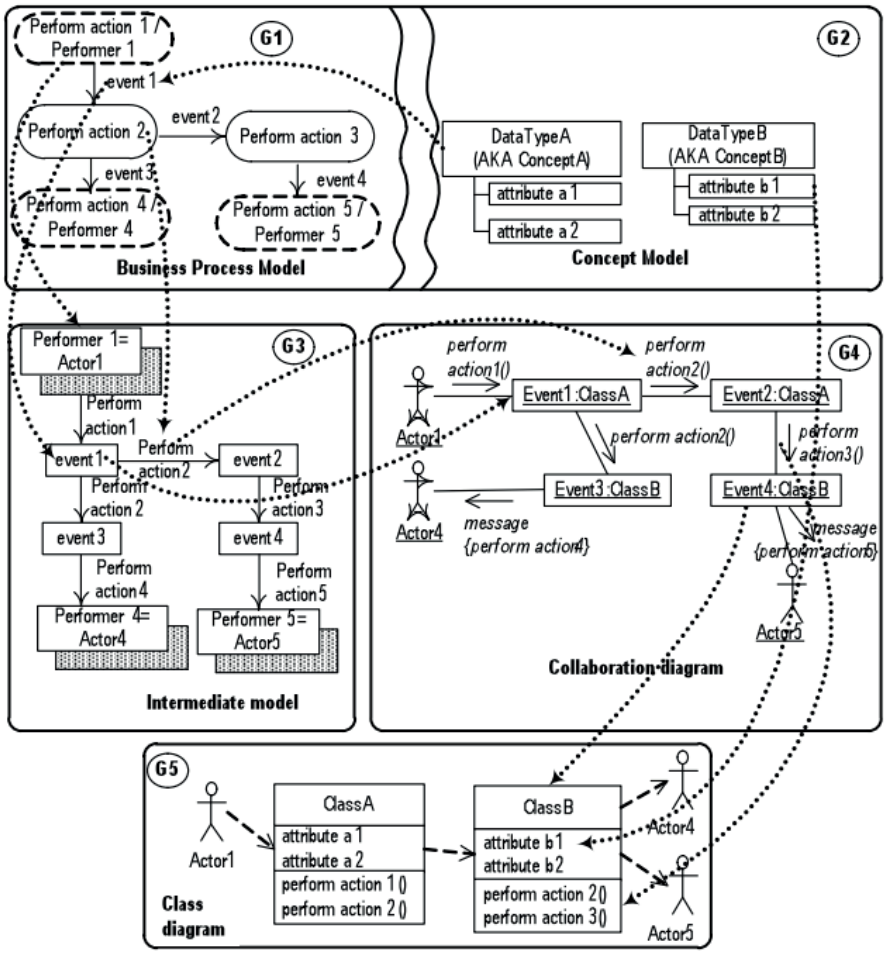
Divpusložu modeļa notācijas tapšana ir parādīta 2.7. attēlā, kur kreisajā pusē uz attēlota procesu modeļa notācijas aizgūšana no datu plūsmas diagrammas elementiem, un labajā pusē – konceptu modeļa notācijas aizgūšana no entītiju saišu diagrammas elementiem.

Divpusložu modelis paredz, ka visi objekti (dati), kuriem jābūt problēmvides atbalsta sistēmā, tiek attēloti ar konceptu diagrammas palīdzību, kas satur informāciju par datu tipiem un to atribūtiem. Un katrs topošās problēmvides atbalsta sistēmas lietošanas gadījuma scenārijs, savukārt, tiek attēlots procesu diagrammas veidā. No matemātiskā viedokļa procesu diagramma ir orientētais grafs, kur virsotnes ir procesi, bet loki – datu plūsmas. Divpusložu modeļa jauninājums pēc būtības ir divu minēto diagrammas notāciju savienošana vienā abstrakcijas līmenī, kas nebija paredzēts citās notācijās vai modelēšanas rīkos – tas ir saistīt šo divu diagrammu notācijas datu plūsmas elementā, kas procesu modelī nāk no viena procesa uz otru un konceptuālajā modelī strukturāli definēts koncepta elementā.

Divpusložu modelī izmantotam procesu modeļa grafam ir definētas vairākas prasības (Gusarovs, 2020; Kozačenko, 2014):

1. Apkārtne – grafu var uzskatīt par divpusložu procesu diagrammu tikai, ja tām ir definētas ieejas un izejas. No matemātiskā viedokļa tas ir līdzīgs tīklam bez loku svariem (Grundspenkis & Jekabsons, 2001). Par ieejām un izejām kalpo speciāls procesu paveids – ārējie procesi. Pārējie procesi tiek saukti par iekšējiem.
2. Jebkurām grafa lokam ir definēti divi atribūti – datu plūsmas nosaukums un attiecīgais koncepts no konceptu diagrammas. Rezultātā divpusložu procesu grafs tiek savienots ar konceptuālo diagrammu. Grafu matemātiskais modelis neļauj izveidot loku, kas iznāk no cita loka un ienāk virsotnē, nav iespējams definēt divpusložu modeli vienā grafa formā.
3. Saistība – divpusložu procesu modelis ir vāji saistīts grafs, kurā, aizvietojot visus orientētus lokus ar neorientētiem, ir iespējams definēt ceļu starp jebkurām divām virsotnēm.
4. Izsekojamība – no jebkuras ieejas procesu diagrammā ir jāeksistē vismaz vienām ceļam līdz vienai no izejām.

2.8. attēlā ir parādīta divpusložu modeļa transformācija uz UML klašu diagrammu caur starpmodeli, kas ir tuvu UML komunikāciju diagrammas notācijai, uz īsto UML komunikāciju diagrammu, kas satur visus klašu diagrammai nepieciešamus elementus.

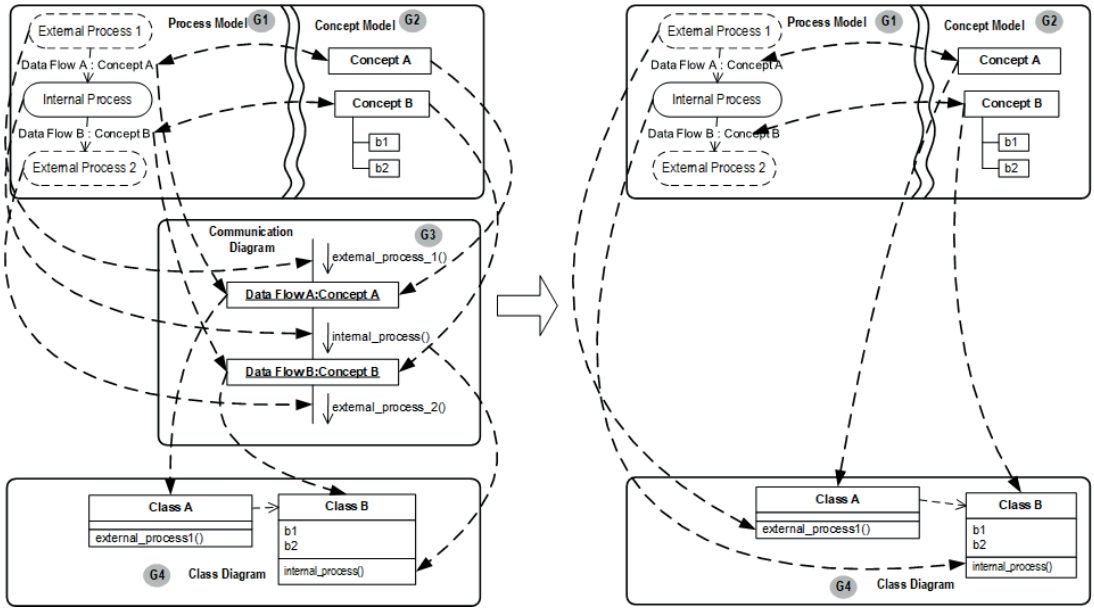



**2.8. att. Sākotnēja pieeja divpusložu modeļa transformācijai UML klašu modelī (aizgūts no (Nikiforova & Pavlova, 2011))**

Divpusložu procesu grafa virsotnes var saturēt informāciju par vienu vai vairākiem procesa izpildītājiem, kas var būt sistēmas sastāvdaļas, vai arī sistēmas apkārtnes objekti, piemēram tās lietotāji. Divpusložu modeļa transformācija balstās grafu inversijas (Grundspenkis & Zeltmate,

2008) principu pielietošanu, kas dod iespēju pārveidot sasaistītus procesu un konceptu modeļus UML secības un klašu diagrammā, kur šīs divas diagrammas tālāk kalpo par pamatu visām pārējām un pirmkoda ģenerēšanai.

Vēlāk uzlabotais algoritms (Nikiforova, Gusarovs et al., 2012) piedāvāja UML klašu diagrammas ģenerēšanu no divpusložu modeļa bez komunikāciju diagrammas izmantošanas. Metodes uzlabošanas rezultātā rezultējoša klašu diagramma tika bagātinātā ar papildus starpklašu relāciju – implementāciju un papildus elementu – interfeisu. Uzlabotas (2.9. att.) transformācijas metodes izpildes rezultāts ir UML klašu diagramma, kas satur klases, interfeisus un 5 veidu attieksmes starp tiem – agregāciju (angl. *Aggregation*), atkarību (angl. *Dependency*), ģeneralizāciju (angl. *Generalisation*), implementāciju (angl. *Implementation*) un asociāciju (angl. *Association*). Uzlabota metode ir shematiski paradīta 2.9. att.




**2.9. att. Uzlabota pieeja divpusložu modeļa transformācijai UML klašu modelī (aizgūts no (Nikiforova, Gusarovs et al., 2012))**

*BrainTool* rīka izstrādes laikā (Nikiforova, Sukovskis et al., 2015; Nikiforova & Gusarovs, 2016) tika definēti vienkārši līkumi attieksmju starp klasēm veidu noteikšanai, šādā veidā

izvairoties no iespējamo transformācijas gadījumu tabulas. 2013. gadā ir izstrādātas transformāciju likumu kopa UML secību diagrammas iegūšanai no divpusložu modeļa. Metodes pamatā ir eksistējošo elementu pārveidošana par secību diagrammas elementiem: koncepti tiek pārveidoti par objektiem, procesu izpildītāji par aktieriem, bet paši procesi – par ziņojumiem. Datu plūsmas tiek izmantotas ziņojumu avotu/saņēmēju noteikšanai. Kopumā tiek identificēti 9 dažādi transformāciju gadījumi, kas pēc tam tiek izmantoti secību diagrammas konstruēšanas laikā (Nikiforova, Kozacenko et al., 2013). Līdzīgi kā uzlabotajā transformācijā uz UML klašu diagrammu, netiek izmantoti papildus modeļi – rezultējošais modelis tiek iegūts no divpusložu modeļa ar formalizēta algoritma palīdzību. Vēlāk UML lietošanas gadījumu un stāvokļu diagrammu ģenerēšanai arī bija definētas atsevišķās transformāciju likumu kopas bez starpniek modeļi lietošanas (Nikiforova, Gusarovs et al., 2013; Kozačenko, 2014)

Divpusložu modeļa lietošanā koda ģenerēšanai procesu modelis arī ir analizēts kā grafs ar iepriekšdefinētu fragmentu meklēšanu, kas ir atbilstoši sagaidāmā koda metamodeļa elementos, un procesu modeļa fragmentiem ir pielietoti dažādi grafu pārveidošanas algoritmi (Gusarovs, 2020; Nikiforova & Gusarovs, 2020).

### 2.2.2. Divpusložu modeļa formālismi

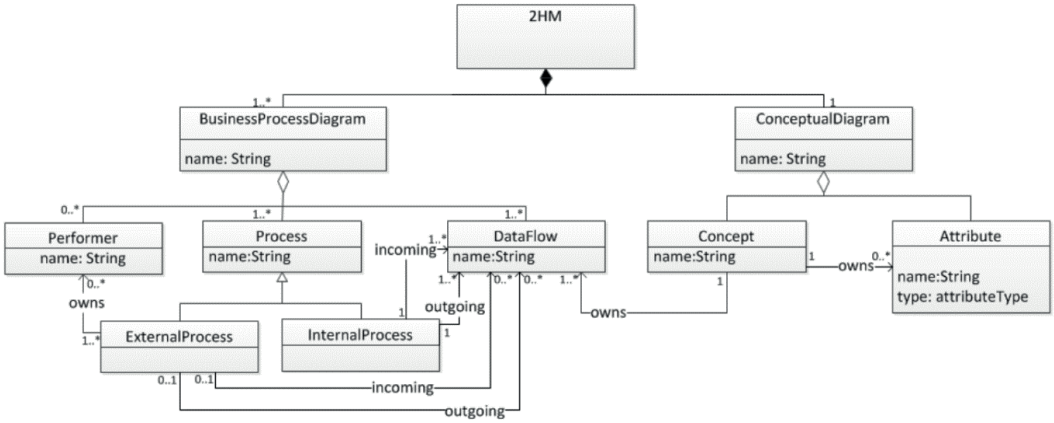
Divpusložu modeļvadāmā pieeja programmatūras komponentu iegūšanai no divpusložu modeļa izmanto trīs formālismus:

1. Konceptuālā/procesa modelēšana ietver problēmvides modeļa izveidi, kas ir neatkarīgs no jebkuras konkrētas ieviešanas valodas vai platformas un satur programmatūras projektēšanas komponentiem nepieciešamo un noteiktajā abstrakcijas līmenī pietiekamo informāciju.
2. Modeļa transformācijas likumi ietver modeļu pārveidošanu programmatūras komponentu komplektā noteiktā sistēmas aspektā (sistēmas funkcionālās prasības, lietošanas scenāriji, objektu mijiedarbība, datu struktūras, sistēmas dinamika, komponentu izvietošana, utt.) atkarībā no izvēlētas projektēšanas vai realizācijas platformas specifikas. Šo transformāciju var veikt manuāli vai automātiski.

3. Automātiskai transformāciju veikšanai ir nepieciešams modeļu transformācijai rīks, kuriem kā ieejas dati tiek padots divpusložu modeļa XML fails, kas ir veidots saskaņā ar divpusložu modeļa metamodeli, sagaidāma rezultējoša faila (failu) struktūra saskaņā ar mērķa metamodeli un tajā ir jābūt realizētai transformācijas likumu palaišanai, kas dos iespēju izveidot rezultējošo failu (failus) noteiktā formātā.

Tātad, lai būtu iespējams realizēt lietotāja saskarnes prototipa automātisku ģenerēšanu saskaņā ar tā metamodelis (skat. attēlu 2.10.) no divpusložu modeļa ir nepieciešams pārliecināties, ka divpusložu modeļa esošais metamodelis saturēs visus nepieciešamos elementus, ko turpmāk lietot transformācijas likumos.

Divpusložu modeļa metamodelis ir parādīts 2.10. attēlā. Ir redzams, ka divpusložu modelis pēc būtības ietver konceptuālu diagrammu un vienu vai vairākas biznesa procesu diagrammas. Šo divu modeļu sasaiste ir attēlota ar saistot datu struktūru no konceptuālās diagrammas attiecību ar datu plūsmu elementu procesa diagrammā (Nikiforova, 2002). Katra datu plūsma ir precīzi saistīta ar vienu konceptu, lai gan katrs koncepts var būt saistīts ar vairākām procesu plūsmām. Šī saikne veido pamatu atbildības noteikšanai objektu klasēm turpmāko transformāciju laikā.




**2.10. att. Divpusložu modeļa metamodelis (aizgūts no (Kozačenko, 2014))**

## 2.3. Nodaļas secinājumi un rezultāti

Lietotāja saskarnes elementu apkopojums ļauj identificēt saskarnes prototipam nepieciešamu elementu kopu. Turklāt iepriekšējo pētījumu esamība par lietotāja saskarnes metamodeļa izstrādi un pielietojumu transformāciju definēšanai dažādos abstrakcijas līmeņos dod papildus pārliecību par šajā promocijas darbā izvēlēta problēmas risināšanas ceļa korektumu. Esošie metamodeļi kalpoja par pamatu izveidot vienotu lietotāja saskarnes metamodeli, kas var tikt izmantots ne tikai šī promocijas darba ietvaros izstrādātajā risinājumā, bet to var turpmāk izmantot citi pētnieki (Babris & Nikiforova, 2024a).

Kā arī divpusložu modeļa praktiskās pielietošanas ietvaros iepriekšējos pētījumos izstrādāts metamodelis jau satur pilnīgu elementu kopu, kas potenciāli ir izmantojama transformāciju definēšanā lietotāja saskarnes elementu ģenerēšanai. Lai būtu iespējams pielietot divpusložu modeļa lietošanas formālismus saskarnes prototipa automātiskajā ģenerēšanā, nākamajā sadaļā ir aprakstīta transformācijas likumu veidošana sagaidāmajos mērķa modeļa elementos saskaņā ar tā metamodeli.

# 3. TRANSFORMĀCIJAS LIKUMU DEFINĒŠANA

Modeļvadāmas izstrādes lietojumprogrammu programmatūras inženierijas izaicinājumiem aizsākās 2000. gados ar ļoti pievilcīgu ideju izveidot no platformas neatkarīgu modeli programmatūras izstrādei, tā no neatkarīgas platformas transformēšana par platformas specifisku modeli, vēl vairāk veicinātu izmantot automātisku koda ģenerēšanu (Suleri, Pandian et al., 2019; Stahl, Völter et al., 2006). Ideja bija daudzsološa, taču dažādu iemeslu dēļ utopiska (Hailpern & Tarr, 2006; Thomas, 2004). Tomēr tajā piedāvātais formālisms (Trehan, Chapman et al., 2015) ir visai noderīgs uzdevumos, kad pilnīgu un konsekventu modeli, pielietojot metamodelēšanas principus, var pārveidot par tā zināšanām pasniegtajām citā formātā (Nikiforova & Gusarovs, 2020; Domingo, Echeverría et al., 2020).

Transformācijas likumi bieži nozīmē sarežģītu ideju pārņemšanu no problēmvides un pārveidošanu vienkāršos lietotāja saskarnes elementos, kurus lietotājiem ir viegli saprast. Tas var ietvert informācijas precizēšanu vai sadalīšanu, nevajadzīgu detaļu slēpšanu un datu rādīšanu sakārtotā, viegli saprotamā formātā. Transformācijas likumi nosaka, kā informācija parādās vizuāli un mijiedarbojas ar lietotāja saskarni. Tas ietver piemērotu lietotāja saskarnes elementu (piemēram, formu, tabulu vai diagrammu) izvēli, aprakstu, kā tie parādīsies un izkārtosies ekrānā, kā arī detalizētu informāciju par to, kā šiem elementiem vajadzētu reaģēt, kad lietotāji ar tiem mijiedarbojas. Lai panāktu lietotāja saskarnes prototipu vienveidību un lietošanas vienkāršību, transformācijas likumos ir jāizmanto lietotāja saskarnes šabloni, vadlīnijas un labākās metodes. Tas nozīmē, ka ir jāsaglabā vienmērīgs vizuālās valodas standarts, jāievēro pieejamības principi un jāveido vienkārša izpēte un mijiedarbība. Lietotāja saskarnes šablonu ģenerēšanas kontekstā, lietotāju saskarnes, automātiski noteikto rezultātu sistemātiska pielāgošana, ir izaicinājums. Lai gan tas var šķist pašsaprotami, ka noteiktus pielāgojumus var izteikt kā īpašus transformācijas likumus, lai pievienotu šādus likumus, parasti ir jāpielāgo esošie likumi un tas var radīt nepārvaldāmu transformācijas likumu kopu (Raneburger, Kaindl et al., 2015).

Transformācijas likumiem var būt jāņem vērā problēmvides dinamiskie aspekti, piemēram, datu stāvokļa maiņa, lietotāja uzstādījumi vai vides apstākļi. Transformācijas likumiem jāietver nosacījumu pārveidošana, uz notikumiem balstītu uzvedību vai adaptīvu izkārtojumu definēšanu – visam tam, kas reaģē uz izpildlaika (angl. *Runtime*) izmaiņām. Tiem jābūt pielāgojamiem un

paplašināmiem – lietotāja saskarnes prasības un uzstādījumi var variēties. Šādā kontekstā, transformācijas likumiem, jāietver konfigurējamu parametru nodrošināšanu, atļauju manuāli ignorēt nosacījumus vai problēmvidei specifiskas pielāgošanas loģikas atbalstīšana. Šie likumi ir jāvalidē un jāpārbauda, lai nodrošinātu, ka tie precīzi iekļauj paredzēto kartēšanu starp problēmvidi un lietotāja saskarnes prototipiem (vai citiem rezultējošiem artefaktiem).

Tā kā pilnībā automātiski ģenerētu lietotāja saskarnes lietojamība parasti ir diezgan zema (Meixner, Paterno et al., 2011), tiek veikta pielāgošana. Modeļvadāmas ģenerēšanas kontekstā to var izdarīt, izmantojot pielāgošanas pārveidošanas likumus (īsumā – pielāgotus likumus). Piemēram, viens no pētījumiem (Raneburger, Kaindl et al., 2015) ir motivēts izstrādāt īpašus likumus, cenšoties uzlabot ģenerēto lietotāja saskarnes lietojamību. Tomēr šajā rakstā pētījuma autori neparādīja, kā to var izdarīt, un neziņoja par īstenošanu (Raneburger, Kaindl et al., 2015).

Transformācijas likumi var ietvert uzģenerētās lietotāja saskarnes testēšanu, atbilstoši iepriekš noteiktiem kritērijiem (Eriksson, Lindström et al., 2013), lietojamības novērtēšanu utt. Var rezumēt, ka transformācijas likumu būtība ir pārvarēt plaisu starp problēmvidi, kura ir formāli aprakstīta un lietotāja saskarnes prototipiem, kas ir ērti gala lietotājiem. Transformācijas likumi, ļauj automātiski ģenerēt lietotāja saskarnes prototipus (vai citus artefaktus), lai precīzi atspoguļotu projektējamo sistēmu, vienlaikus apmierinot gala lietotāju prasības.

No modeļa uz modeli vai no modeļa uz tekstu (kodu) var veikt izmantojot transformācijas likumus (Mens, 2013), tas veido modeļvadāmu procesu un ļauj precizēt avota un mērķa modeļus vai metamodeļus. Šādā procesā tiek izveidota kartēšanas definīcija, kas nosaka attiecības starp elementiem katrā modelī vai metamodelī, kas ietver sevī kartēšanas likumus – tie nosaka savienojumus starp elementiem un var saturēt sarežģītu ieviešanas informāciju.

Modeļa pārveidošana kodā parasti rezultējas ar kodu vai citiem artefaktiem. Šāds process palīdz izstrādātājiem pārvērst augsta līmeņa lietotāja saskarnes modeļus, kas parasti tiek veidoti standarta modelēšanas valodās, piemēram, XML vai UML, ieviešamos komponentos. Šajā gadījumā, transformācijas likumi, kas norāda, kā avota modeļa elementi un to attiecības jāsaista ar atbilstošiem koda attēlojumiem mērķa kodā, tiek papildināti ar lietotāja saskarnes šabloniem un ģenerētiem demonstrācijas datiem. Mērķa teksts var būt pirmkods iezīmēšanas vai programmēšanas valodās, piemēram, HTML, CSS vai JavaScript kā arī specifikācijās, piemēram, JSON vai XML formātā (Ammar, 2021).

Pārveidošana no modeļa uz kodu palīdz automatizēt un padarīt lietotāja saskarnes izstrādi konsekventu. Šī metode palīdz arī apvienot lietotāja saskarnes izstrādes darbības ar citām programmatūras izstrādes darbībām, piemēram, testēšanu vai projekta lietotāja saskarnes prasību dokumentēšanu. Modeļa uz kodu automatizācija uzlabo produktivitāti, jo tā automātiski pārvērš augsta līmeņa lietotāja saskarnes modeļus kodā vai specifikācijās.

Modeļvadāmas izstrādes kontekstā transformācijas likumi definē elementu pārveidošanu no problēmvides modeļa elementiem uz elementiem definētiem lietotāja saskarnes modelī formātā, kas pēc būtības ir tīmekļa lietojumsistēmas priekšgala komponentu pirmkods. Tie ir likumi, kas izskaidro, kā informācija, darbība un struktūra no problēmvides tiek kartēta uz lietotāja saskarnes attēlojumu (Bajovs, Nikiforova et al., 2013). Šī promocijas darba izstrādāta risinājuma kontekstā definētie transformācijas likumiem ir jāpārklāj problēmvides būtības pārņemšana un pārveidošana par lietojumsistēmas satura vizuālo daļu, kā arī interaktīvās lietotāja saskarnes daļās, kas ir sistēmas lietošanas scenārija soļi. Ir ļoti svarīgi, lai transformācijas likumi izveidotu skaidru saikni starp idejām problēmvides modelī un saistītajiem elementiem lietotāja saskarnē (Arrhioui, Mbarki et al., 2017). Tas nozīmē, ka ir jāpaskaidro lietotāja saskarnes formu, tās satura elementu, darbību un ierobežojumu jēga un attiecības ar problēmvidē entītijām (divpusložu modeļa konceptiem) un scenārija soļiem (divpusložu modeļa procesiem), kā arī tas, kā tie ir jāattēlo lietotāja saskarnē.

Lietotāja saskarni pēc būtības arī var apskatīt kā noteikta aspekta programmatūras komponentu attēlošanu modelī noteiktajā abstrakcijas līmenī, kas atbilst iepriekš noteiktam formātam, t.i. metamodelī aprakstītam sagaidāmā rezultāta saturam un struktūrai.

Iepriekšējā nodaļā ir izveidots mērķa modeļa metamodelis, saskaņā ar kuru lietotāja saskarnes prototipā sagaidāmie komponenti ir sadalīti trīs grupās (Leuthold, 2010; Koch & Mandel, 1999):

1) Koncepta elementi
2) Navigācijas elementi
3) Prezentācijas elementi

Par avota modeli ir izvēlēts divpusložu modelis, kas iepriekš jau bija pielietots projektēšanas un programmatūras komponentu automātiskajai ģenerēšanai.

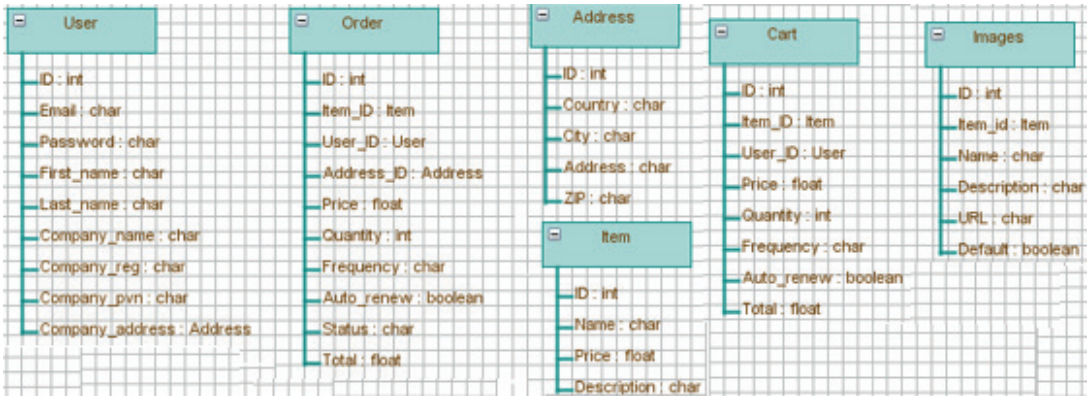
Šajā nodaļā ir definēta transformācijas likumu kopa avota modeļa pārveidošanai par mērķa modeli saskaņā ar mērķa metamodelī definētām elementu grupām. Turklāt, attiecībā uz avota

modelī trūkstošiem elementiem ir pētītas citu formālismu lietošanas iespējas, lai noklātu pilnīgu mērķa modeļa elementu iegūšanu.

## 3.1. Koncepts

Konceptuālās projektēšanas soļa aktivitātes ir klašu noteikšana, atribūtu un operāciju precizēšana, hierarhisko struktūru definēšana un apakšsistēmu noteikšana. Lai sasniegtu šo mērķi, tiek izmantotas labi zināmas objektorientētas modelēšanas metodes.

Šajā darbībā definētās klases un asociācijas tiek izmantotas arī navigācijas projektēšanas laikā, lai iegūtu tīmekļa lietotnes struktūras saites. Attiecības tiks izmantotas, lai iegūtu saites. Klases un asociācijas var tikt organizētas grupās. Klases ir aprakstītas, izmantojot atribūtus un darbības, un attēlotas grafiski (Koch & Mandel, 1999). Konceptuālās izstrādes soļa rezultāts ir apkopots konceptuālā modelī, kas sastāv no klasēm un asociācijām starp klasēm, kas modelē problēmas jomu. Šajā darbā koncepts kā pamats tiek izmantots no divpusložu modeļa ģenerētais koncepta modelis, kas ļauj atrast sakarību starp dažādiem datu modeļiem un aprakstīt tos. 3.1. attēlā var redzēt no divpusložu modeļa izgūto koncepta modeli.




**3.1. att. No divpusložu modeļa izgūtais koncepta modelis**

Pētījumā *Screen2Vec* (Li, Popowski et al., 2021), kas tika pielāgots no cita pētījuma (Liu, Craft et al., 2018), tiek definētas 26 kategorijas ar izmantojamiem lietotāja saskarnes elementiem, kuri ir kartēti ar klašu tipiem (3.2. att.). Dažiem tipiem ir papildus heiristika. Faktiski, ir

nepieciešams izmantot plaši pielietoto lietotāja saskarnes elementu kartēšanu ar izvēlēto ietvaru elementiem, definēt to grupēšanu un hierarhiju, lai ar maksimāli īsu aprakstu varētu izgūt nepieciešamos lietotāja saskarnes elementus tādā formā, ka tie iekļautos transformējamā kodā loģiskā secībā un varētu tikt demonstrēti lietotājam apvienojot ar funkcionālu navigācijas modeli – padarot redzamo lietotāja saskarni dinamisku un lietojamu.

| GUI Component | Associated Class Type | GUI Component | Associated Class Type |
|---|---|---|---|
| Advertisement | `AdView, HtmlBannerWebView, AdContainer` | Layouts | `LinearLayout, AppBarLayout, FrameLayout, RelativeLayout, TableLayout` |
| Bottom Navigation | `BottomTabGroupView, BottomBar` | Button Bar | `ButtonBar` |
| Card | `CardView` | CheckBox | `CheckBox, CheckedTextView` |
| Drawer (Parent) | `DrawyerLayout` | Date Picker | `DatePicker` |
| Image | `ImageView` | Image Button | `ImageButton, GlyphView, AppCompatButton, AppCompatImageButton, ActionMenuItemView, ActionMenuItemPresenter` |
| Input | `EditText, SearchBoxView, AppCompatAutoCompleteTextView, TextView`[a] | List Item (Parent) | `ListView, RecyclerView, ListPopupWindow, TabItem, GridView` |
| Map View | `MapView` | Multi-Tab | `SlidingTab` |
| Number Stepper | `NumberPicker` | On/Off Switch | `Switch` |
| Pager Indicator | `ViewPagerIndicatorDots, PageIndicator, CircileIndicator, PagerIndicator` | RadioButton | `RadioButton, CheckedTextView` |
| Slider | `SeekBar` | TextButton | `Button`[b], `TextView`[c] |
| Tool Bar | `ToolBar, TitleBar, ActionBar` | Video | `VideoView` |
| Web View | `WebView` | Drawer Item | `Others category and ancestor is Drawer(Parent)` |
| List Item | `Others category and ancestor is List(Parent)` | Others | `...` |

**3.2. att. Lietotāja saskarnes elementu un klašu nosaukumi (aizgūts no (Li, Popowski et al., 2021))**

### 3.1.1. Eksperimenti ar lietotāja saskarnes konceptu

Lai izmantotu divpusložu modeļa konceptu kā datu modeli, ir nepieciešams izvērtēt šobrīd esošā rīka *BrainTool* eksportējamo XML datni. Šajā datnē glabājas informācija par to kādi koncepti ir definēti un kādas ir to savstarpējās attiecības. 3.1. tabulā ir redzams *BrainTool* XML datnes informācija par konceptu.

Papildus koncepta modelim, no divpusložu modeļa ir iespējams izgūt arī secību diagrammu, no kuras, savukārt, var izgūt koncepta modelim pieejamās metodes. No šīm metodēm var izgūt arī izpratni par to vai koncepta lauki atrodas lasīšanas vai rediģēšanas režīmā.

Katram koncepta atribūtam ir vairāki tā aprakstošie elementi. Viens no galvenajiem elementiem ir atribūta nosaukums un atribūtu datu tips. Tieši atribūta datu tips var tikt izmantots

kā norāde uz to, kādu lietotāja saskarnes elementu nepieciešams demonstrēt (izejas dati) un kādus datus koncepta atribūt var pieņemt (ieejas dati). Ņemot vērā šo informāciju jau var demonstrēt vienkāršas formas, kuras jau tiek veidotas dažādos ietvaros izmantojot balstīšanas sistēmu (angl. *Scaffolding*), kuras pamatā, parasti ir CRUD pieeja, kas ļauj izveidot kartēšanu starp modeļiem (konceptiem), to atribūtiem, maršrutiem un formām (Pantelelis & Kalloniatis, 2023).

**3.1. tabula**

**Divpusložu modeļa koncepta saturs**

| **Divpusložu modeļa koncepta elements** | **Piemērs** | **Apraksts** |
|---|---|---|
| <id> | 1 | Koncepta identifikators |
| <name> | Product | Koncepta nosaukums |
| <description> | Product for internet shop | Koncepta apraksts |
| <comment> | | Koncepta komentārs |
| <type> | CLASS_CONCEPT | Koncepta tips |
| <classAttributes> | <attribute> | Koncepta atribūtu saraksts |

Divpusložu modeļa koncepta *<classAttributes>* satur koncepta atribūtus un faktiski norāda gan to, kāds būs koncepta atribūtu datu tips, gan arī attiecības starp konceptiem, jo *<attributeType>* var atzīmēt arī norādi (atkarību) uz citu konceptu (*<Concept>*). 3.2. tabulā ir redzama pieejamā informācija no *BrainTool* XML datnes elementu *<attribute>*, kas ir bērnelements *<classAttributes>* vecākelementam.

Faktiski šī informācija ļauj izveidot kaut ko līdzīgu ER diagrammai, no kuras var ģenerēt lietotāja saskarnes elementus. Koncepta atribūtu kartēšana uz lietotāja saskarnes elementiem nodrošina, ka saistītā informācija tiek parādīta un ar to var pareizi mijiedarboties lietotāja saskarnē. Ne visai informācijai no koncepta modeļa ir jābūt redzamai lietotājam, faktiski ir nepieciešams norādīt kādi lauki ir pieejami lietotājam un kādiem ir jābūt paslēptiem. Turklāt, šai lauku atribūtu pievienošanai ir jābūt pārvaldāmai, jo, piemēram, rediģēšanas vai formas ievades laikā lauks var tikt slēpts, bet lasīšanas režīmā, šim laukam un tā saturam var jābūt redzamam.

**3.2. tabula**

**Divpusložu modeļa koncepta atribūtu saturs**

| **Divpusložu modeļa koncepta elements** | **Piemērs** | **Apraksts** |
|---|---|---|
| <id> | 12 | Atribūta identifikators |
| <name> | Price | Atribūta nosaukums |
| <description> | Product price | Atribūta apraksts |
| <comment> | | Atribūta komentārs |
| <type> | CLASS_ATTRIBUTE | Atribūta tips |
| <attributeType> | Int, char, float, <Concept>, utt. | Atribūta datu tips |

Piemēram, konceptam ar nosaukumu "Produkts", kurā atrodas tādas kolonnas kā "ID", "Nosaukums", "Cena", un "Apraksts". Lai lietotāja saskarnē rādītu šīs kolonnas, piemēram, kā HTML elementus, viens no veidiem varētu būt formas izveide produktu pievienošanai vai rediģēšanai (izmantojot zināšanas par CRUD). Šādā formā koncepta atribūti tiktu kartēti ar kartējamiem lietotāja saskarnes elementiem (ievades datiem).

Koncepta atribūtu "ID" nepieciešams slēpt, jo šis lauks nav rediģējams un darbojas kā unikāls identifikators. Koncepta atribūtu "Nosaukums" un "Apraksts", kuri satur teksta veida datus, var piesaistīt teksta ievades laukiem, kur lietotāji var ievadīt produktu nosaukumus un aprakstus. Koncepta atribūta "Cena", kas paredzēta skaitliskiem datiem, var saistīta ar ciparu ievades lauku. Tas nodrošinātu, ka var ievadīt tikai derīgas cenas. Turklāt, ja pastāv sasaistes attiecības ar citiem kontekstiem, piemēram, "CategoryID", kas sniedz informāciju par to, kurā kategorijā pieder produkts, var atlasīt izmantojot izkrītošo sarakstu vai automātiskās pabeigšanas ievadi (angl. *Autocomplete*).

### 3.1.2. Transformācijas likumi saskarnes koncepta elementu ģenerēšanai

Konceptuālais plānojums veido konceptuālo problēmvides modeli, kuru izgūst no klasēm un asociācijām starp tām (Koch & Mandel, 1999). Lai saprastu, kādā stāvoklī datiem ir jāatrodas –

ievades vai izvades veidā, ir nepieciešams piesaistīt divpusložu modeļa procesu modeli, kurā glabājas zināšanas par to, kādā veidā (kādā skatījumā) ir jāparāda dati.

Vienā no pētījumiem (Cruz (da), Miguel et al., 2010a) sniedz pieeju lietotāja saskarnes modeļu un prototipu automātiskai izveidei no problēmvides modeļiem un lietošanas gadījumu modeļiem. Tas uzsver nepieciešamību izveidot lietotāja saskarnes, kas precīzi parāda sistēmas pamatprasības un mijiedarbību ar lietotājiem. Izmantojot šo metodi, vispirms tiek iegūti problēmvides modeļi kā arī tiek iegūti lietošanas gadījumu modeļi, lai izprastu priekšstatu par sistēmas apkārtni un tās mijiedarbību ar lietotājiem. Šie modeļi kalpo par pamatu lietotāja saskarnes ģenerēšanai.

Pēc tam semantiskās kartēšanas process veido savienojumus starp elementiem problēmvidē un lietošanas gadījumu modeļus ar tiem atbilstošajiem lietotāja saskarnes elementiem. Tas nodrošina, ka lietotāja saskarne precīzi attēlo to, kam ir jānotiek un kā lietotāji mijiedarbojas (3.3. tabulā ir redzams procesa modeļa elementa un darbības ar konceptu kartēšana). Šajā tabulā var redzēt, ka lietošanas gadījumi ir kartēti ar datu modeli, tāpēc, saņemot CRUD operāciju (Roubi, Erramdani et al., 2016) no iepriekš definētiem šabloniem, palīdz izlemt, kuriem lietotāja saskarnes komponentiem jābūt redzamiem. (Arrhioui, Mbarki et al., 2017).

No semantiskās kartēšanas procesa tiek izgatavoti lietotāja saskarnes prototipi. Šie prototipi nāk no problemvides un lietošanas gadījumu modeļiem. Tie sniedz vizuālu ieskatu par to, kāda varētu būt sistēmas lietotāja saskarne. Iesaistītās personas var to pārbaudīt, lai noskaidrotu, vai viņi piekrīt projektam, kā rezultātā var tikt veiktas izmaiņas pirms galīgā lietošanas apstiprinājuma piešķiršanas.

Rakstā (Tena, Diez et al., 2013) tiek piedāvāta metode lietošanas gadījumu standartizēšanai, izstrādājot kontrolētu vārdu krājumu, kas paredzēts tieši tīmekļa lietotāju uzdevumu aprakstīšanai. Šīs kontrolētās vārdnīcas galvenais mērķis ir nodrošināt konsekvenci un skaidrību, izskaidrojot uzdevumus, ko tīmekļa lietotāji veic dažādos projektos un komandās. Sākotnējais posms ietver biežu tīmekļa lietotāju uzdevumu atpazīšanu, kas notiek dažādās tīmekļa lietotnēs. Šis solis palīdz izveidot pamatu kontrolētam vārdu krājumam. Pēc tam tiek izveidots šo kontrolēto vārdu krājums, kas satur vienotus terminus un to nozīmes, lai attēlotu tīmekļa lietotāju pienākumus. Šīs vārdu izvēles mērķis ir panākt, lai daži apraksti būtu skaidri, precīzi un vienmērīgi. Izmantojot šo veidu,

lai aprakstītu procesa diagrammu, tas palīdz cilvēkiem vieglāk saprast, kā sistēma darbojas (3.3. tabula).

Faktiski, izmantojot divpusložu modeļa koncepta modeli, ir iespējams kartēt iespējamos izgūstamos elementus no divpusložu modeļa un sadalīt par ievades un izvades elementiem. Ievades elementi ir tādi elementi, kuri pieņem lietotāja ievadītos datus, bet izvades elementi ir informācija, kura tiek parādīta lietotājam. 3.3. tabulā ir redzama elementu un darbību sadale.

Tādas operācijas kā "Create", "Update" un "Create related" ir nepieciešams attēlot rediģējamā formā, kamēr "Only value update" var tikt attēlots arī skatīšanas režīmā. "Retrieve" gadījumā ir jāparāda lauki skatīšanās režīmā. Izvēloties "List" vai "List related", tiek parādīts saraksts ar ierakstiem.

**3.3. tabula**

**Procesa modeļa elementa un darbības ar konceptu kartēšana (adaptēts no (Elec, Uyanik et al., 2020))**

| **Procesa modeļa elements** | **Koncepts** | **Operācija** | **Tips** |
|---|---|---|---|
| List Orders | Order | List | Izvades dati |
| Add new order | Order | Create | Ievades dati |
| Edit order | Order | Update | Ievades dati |
| Delete order | Order | Delete | Ievades dati |
| Update order status | Order | Only value update | Ievades dati |
| List order items | Item | List related | Izvades dati |
| Add order item | Item | Create related | Ievades dati |
| Edit order item | Item | Update | Ievades dati |
| Select Item | Item | Select related | Izvades dati |
| View Item | Item | Retrieve | Izvades dati |

Turpinot koncepta elementu kartēšanu var detalizēt kādus lietotāja saskarnes elementus attēlot lietotājam, balstoties uz divpusložu modeļa koncepta atribūtu tipiem (3.4. tabula). Kā redzams tabulā, tad divpusložu modeļa kartēšana ar lietotāja saskarnes elementiem ir redzama

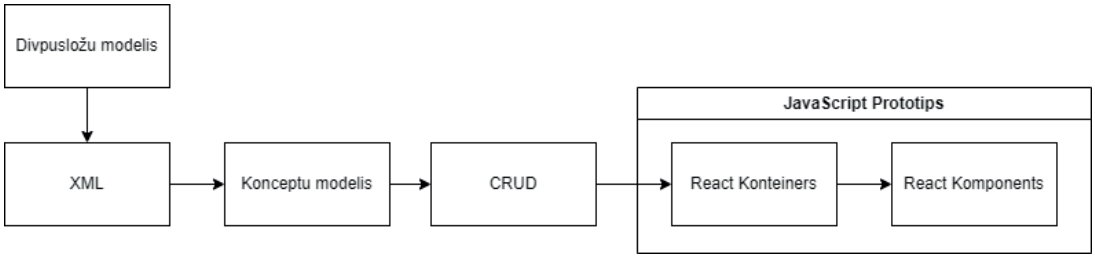
ierobežotā apjomā un, balstoties tikai uz datu tipu, nav iespējams pilnīgi noteikt tā attēlošanas formu. Pirmkārt, ir nepieciešams zināt vai tie ir ievades vai izvades dati un tālāk ir nepieciešams saprast kādā kontekstā tie ir jālieto, piemēram, ja lauka vērtība ir Int (angl. *Integer*), tad var secināt, ka tas būs ievades vai izvades lauks ar iespēju ievadīt tikai skaitliskas vērtības, bet kā tas būtu jāattēlo var tikai pievienojot nozīmi šim laukam.

**3.4. tabula**

**Divpusložu modeļa datu tipu kartēšana ar lietotāja saskarnes elementiem**

| **Tips** | **Parametru tips** | **Grafiskais attēlojums** |
|---|---|---|
| **Ievades dati** | Boolean | Checkbox |
| | Char | Text field |
| | Double | Numberic field |
| | Float | |
| | Byte | Numberic field |
| | Int | |
| | Short | |
| | Long | |
| | Saistītais koncepts | Dropdown list |
| **Izvades dati** | Boolean | Disabled Checkbox |
| | Char | Disabled Text field |
| | Double | Disabled Numberic field |
| | Float | |
| | Byte | Disabled Numberic field |
| | Int | |
| | Short | |
| | Long | |
| | Saistītais koncepts | Related record disabled data |

Datu ievades / izvades identificēšanai var palīdzēt lietotāja saskarnes šabloni, kuros jau būtu zināšanas par to kā ir jāattēlo. Neizmantojot lietotāja saskarnes šablonus, var izgūt lietotāja saskarnes prototipu tādā veidā, kā tiek attēlots 3.3. attēlā.




**3.3. att. Koncepta skatījuma izgūšana no divpusložu modeļa**

Otrkārt, ir novērots, ka vienkāršām lietotāja saskarnēm ir pietiekoši kartēt funkcionalitāti, izmantojot, piemēram, balstīšanas sistēmu, tomēr sistēmām ar sarežģītu loģiku var rasties problēmas (piemēram, lieka koda duplikācija, u.c.) (Andrade, Vilar et al., 2017). Lai risinātu sarežģītu sistēmu lietotāja saskarnes izveidošanas problēmas, var tikt izmantoti lietotāja saskarnes šabloni.

Lai no divpusložu modeļa izgūtu lietotāja saskarnes datu modeļa elementus, ir nepieciešams izvedot kartēšanas shēmu, kurā kreisajā pusē būtu divpusložu modeļa koncepta modelis, bet labajā pusē rezultējošais JavaScript kods. Šādas kartēšanas rezultāts ir redzams 3.5. tabulā. Tajā brīdī, kad no divpusložu modeļa koncepta atribūts tiek transformēts par JavaScript objekta atribūtu, tiek zaudēta informācija par datu tipiem, jo JavaScript datu tipu klāsts ir šaurāks nekā divpusložu modelim, t.i., "string", "number", "boolean", "object", "function", "null", "undefined" un divi objektu tipi "Object" un "Array".

**3.5. tabula**

**Divpusložu modeļa *Order* koncepta kartēšana ar JavaScript kodu**

| Divpusložu koncepta atribūts | JavaScript objekta atribūts | JavaScript datu tips |
|---|---|---|
| ID: int | Order.id | Number |
| Item_ID: Item | Order.item_id | Number, String, Array, Object |

| Divpusložu koncepta atribūts | JavaScript objekta atribūts | JavaScript datu tips |
|---|---|---|
| Address_ID: Address | Order.address_id | Number, String, Array, Object |
| Price: float | Order.price | Number |
| Quantity: int | Order.quantity | Number |
| Frequency: char | Order.frequency | String |
| Auto_renew: boolean | Order.auto_renew | Boolean |
| Status: char | Order.status | String |
| Total: float | Order.total | Number |

### 3.1.3. Divpusložu modeļa papildināšana ar koncepta atribūtu tipiem

Kā transformēt saistītos konceptus, piemēram, divpusložu modeļa atribūtu "Item_ID" uz JavaScript "Object" paliek atvērts jautājums, jo var izmantot gan tekstuālu vai ciparu norādi uz saistītā koncepta identifikatoru vai arī iekļaut šo saistīto konceptu masīvā vai objektā, ja tas ir viens ieraksts. Ja divpusložu modeļa koncepta notāciju papildinātu ar specifiskākiem datu tipiem, piemēram, "Date", tad varētu transformācijas procesā, JavaScript objekta datu tipu norādīt kā "Date" objekta instanci un Lai samazinātu neskaidrības faktoru, šajā darbā tiek pieņemts, ka saistītais koncepta elements tiek norādīts ar ciparu vērtību, kas pēc semantikas atbilst ID koncepta atribūtam saistītajā ierakstā.

Pēc eksperimentiem ar konceptu, tika noskaidrots, ka divpusložu modelī (*BrainTool* rīka ietvaros ir tikai 8 tipi) ir ierobežots koncepta lauku tipu skaits, kas nozīmē, ka, šobrīd, nav iespējams izmantot pilnīgu lietotāja saskarnes elementu kartēšanu. Var secināt, ka divpusložu modelis ir jāpapildina ar citiem lauku tipiem. Kā pozitīva iezīme ir tas, ka konceptā var diezgan viegli norādīt uz saistītajiem ierakstiem. Šādā gadījumā, var noderēt lietotāja saskarnes šabloni, kuros būtu jau norādīti izkārtojuma un formu elementi un tie tikai būtu jāsavieno ar koncepta lauku tipiem un nosaukumiem. Faktiski, līdzīgi kā citā pētījumā (Mahatody, Ilie et al., 2021), būtu nepieciešams papildināt koncepta lauku veidus ar, piemēram:

1. *Text*, lai noteiktu gara teksta ievades opciju.
2. *Enum*, kas palīdzētu definēt īsus izkrītošos sarakstus.

3. *Date* un *Datetime*, kas palīdzētu izveidot datumu un laika laukus (piemēram, no datubāzes).

Iespējams, izmantojot lietotāja saskarnes šablonus ar jau definētu koncepta sasaisti ar lietotāja saskarnes elementiem, šī problēma būtu atrisināta. Piemēram, lauku slēpšanu un parādīšanu lasīšanas un rakstīšanas režīmā, varētu atrisināt pielietojot noteiktu šablonu katrā gadījumā. Darba turpinājumā tiek aprakstīta ar navigāciju saistītā informācija.

## 3.2. Navigācija

Navigācijas izstrādei tiek pielietota stāvokļa diagramma (Sunitha & Samuel, 2019), kura tiek transformācijas ceļā izgūta no divpusložu modeļa. Līdzīgi kā citos pētījumos, piemēram, izgūstot stāvokļa diagrammas no prasību specifikācijas (Pimentel, Castro et al., 2014; Briand, Labiche et al., 2005). Lietotāja saskarnes navigāciju var attēlot kā stāvokļa diagrammu, pa kuru pārvietojas lietotājs izmantojot navigāciju.

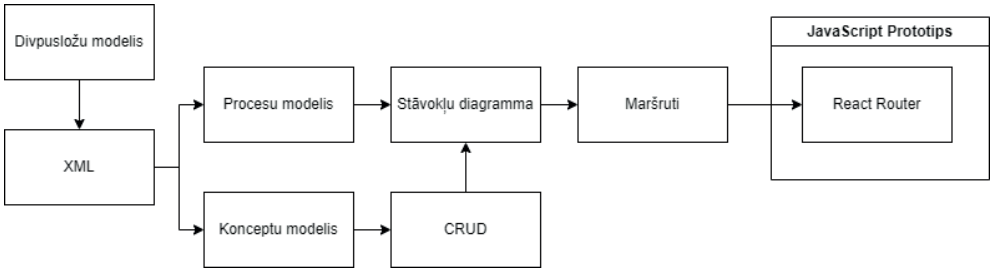
Stāvokļa diagrammas, arī pazīstamas kā UML stāvokļa mašīnas ir galīgais automāts (angl. *Finite State Machines, FSM*) papildināts ar hierarhiju un laiksakritību (angl. *Concurrency*) (Marinescu, Seceleanu et al., 2015). Izmantojot šo modeli, var izmantot stāvokļa mašīnas, lai modelētu sarežģītas reāllaika sistēmas. Stāvokļa diagrammas modelis parāda iespējamos stāvokļus un pārejas, kas izraisa stāvokļa izmaiņas.

Viens no iemesliem, kāpēc stāvokļa diagrammas (angl. *Statechart diagram*) nav ieguvušas plašu popularitāti priekšgala lietotāju saskarnēs ir labo praksu un vadlīniju trūkums. Šobrīd nav pietiekami standartizētas pieejas, kā izmantot stāvokļa diagrammas tādos uz komponentiem balstītos ietvaros kā *React.js*, *Vue.js* vai *Angular*. Neskatoties uz to, ka stāvokļu grafi tiek plaši un ilgstoši izmantoti citās jomās, tiek pieņemts, ka lietotāju saskarnes izveide ir vienkārša, jo ir pieejami dažādi izstrādes rīki. Fakts ir tāds, ka lietotāja saskarnes rīki neatrisina visas problēmas, un lietotāja saskarnes koda uzturēšanas izmaksas var būt milzīgas (Horrocks, 1999).

Galīgie automāti, augšupējs dizains (angl. *Bottom-up*) un stāvokļa diagrammas ir trīs dažādas metodes. Tos izmanto programmatūras inženierijā, jo īpaši tādu sistēmu konstruēšanā, kurām ir sarežģīta uzvedība. FSM ir teorētisks modelis, kas apraksta sistēmu kā stāvokļu kopu un pārejas starp šiem stāvokļiem. Šajā modelī sistēma var pastāvēt tikai vienu reizi jebkurā stāvoklī

noteiktā brīdī – to sauc par "pašreizējo stāvokli". FSM bieži izmanto, lai attēlotu sistēmas ar diskrētu un deterministisku uzvedību, tādējādi padarot tos vispiemērotākos tieši definētām sistēmām. Projektēšana izmantojot *Bottom-up* ir metode, kas sākas ar zema līmeņa projektēšanas komponentiem vai detaļām un virzās uz augstāka līmeņa struktūrām vai sistēmām. Tas ietver problēmas sadalīšanu mazākās, vieglāk apstrādājamās daļās un pakāpenisku šo daļu pievienošanu, lai iegūtu galīgo risinājumu. Funkcionalitātes izveide no nulles ir galvenais uzsvars uz augšupēju dizainu, bieži uzsverot, kā moduļus var izmantot atkārtoti, un to modularitāti. No otras puses, stāvokļa diagrammas ir FSM paplašinājums. Tās var tikt galā ar sarežģītāku uzvedības modelēšanu, ieviešot hierarhiskus stāvokļus, paralēlus stāvokļus un pārejas, kas reaģē uz notikumiem. Veids, kā stāvokļa diagrammas vizuāli parāda sistēmas uzvedību, ir izteiksmīgāks un vieglāk mērogojams nekā tradicionālajos FSM. Tas padara tos piemērotus sistēmu modelēšanai ar sarežģītu uzvedību un mijiedarbību. Lai gan FSM ir vienkāršāki un tiešāki, stāvokļa diagrammas var būt elastīgas un abstraktas. Tas ļauj dizaineriem uztvert smalkākas sistēmas darbības.

Stāvokļa diagrammas ir iespējams ģenerēt arī no UML secību diagrammām (angl. *Sequence Diagram*), izmantojot, piemēram, grafu transformācijas (Grønmo & Møller-Pedersen, 2011). No stāvokļa diagrammām ir piedāvājuši ģenerēt lietotāja saskarni arī citi autori (Horrocks, 1999; Wellner, 1989; Harel, 1987), pielietojot stāvokļus dažādās lietotāja saskarnes izšķirtspējās – sākot no lietotāja saskarnes pamatelementu, piemēram, pogu darbības, beidzot līdz kopējās lietotāja saskarnes navigācijas shēmām. Šī darba ietvaros stāvokļa diagrammas tiek izmantotas kā pamats kopējam navigācijas modelim, par kuru var pārvietoties gala lietotājs, neiedziļinoties katra lietotāja saskarnes elementā (3.4. att.).




**3.4. att. No divpusložu modeļa izgūstamo stāvokļu diagrammu kopējā shēma**

Ņemot par pamatu konceptuālo modeli, balstoties uz navigācijas plānojumu, tiek veidots navigācijas modelis, kas ir skatījums uz konceptuālo modeli. Šo navigācijas modeli izgūst divos etapos (Koch & Mandel, 1999):

1) Kādi objekti ir potenciāli sasniedzami izmantojot navigāciju.
2) Kā šie objekti tiek sasniegti.

Navigācijas modeļa izveidošanai ir nepieciešams izmantot zināšanas no citiem objektiem, lai veiktu objektu grupēšanu (Koch & Mandel, 1999).

Modeļvadāmo pieeju mērķis ir automātiski ģenerēt kodu no modeļiem. Tādejādi, katrā modeļvadāmā metodoloģijā darbs notiek ar modeļiem un transformācijām. Šiem modeļiem ir jāatbilst attiecīgajiem meta-modeļiem. Savukārt transformācijas tiek aprakstītas izmantojot transformācijas valodas (Gharaat, Sharbaf et al., 2021).

### 3.2.1. Eksperimenti ar navigācijas elementu izgūšanu no divpusložu modeļa

Lai iegūtu stāvokļa diagrammas no divpusložu modeļa, jāsāk ar procesu analīzi, jānosaka stāvokļi, notikumi un pārejas. Piemēram, interneta veikalam, stāvokļi varētu būt "Skatīt preci", "Pievienot preci grozam", "Veikt norēķinu" utt. Notikumi, kas ierosina pāreju no viena stāvokļa uz citu, var ietvert lietotāja darbības vai sistēmas atbildi (piemēram, kļūdas).

Stāvokļa diagrammu automātiska izgūšana no divpusložu modeļa, attiecas uz programmatūras rīku vai algoritmu izmantošanu, kas var interpretēt divpusložu modeļa procesus un interpretēt nepieciešamo informāciju, lai izveidotu stāvokļa diagrammas. Tieši divpusložu modeļa procesi glabā sevī informāciju par sistēmas navigācijas struktūru, kā arī tās uzvedību un mijiedarbību.

No divpusložu modeļa ir nepieciešams izgūt stāvokļus, notikumus un pārejas. Faktiski, divpusložu modelī, process ir stāvoklis – ņemot vērā šo informāciju, var divpusložu modeļa procesus transformēt uz stāvokļiem. Savukārt kā pārejas starp stāvoļiem iespējams izgūt no nākošajiem procesiem jeb stāvokļiem, kuri ir blakus. Tie stāvokļi, kuri ir nākošajā posmā tiek interpretēti kā pārejas – veids kā nokļūt no esošā stāvokļa nākamajā.

Lai veiktu eksperimentus ar lietotāja saskarnes navigācijas ģenerēšanu no divpusložu modeļa, ir nepieciešama kāda JavaScript stāvokļu diagrammu bibliotēka. Autora piedāvātajā

risinājumā tiek izmantota *XState*, bet tā vietā var izmantot jebkuru citu bibliotēku, kura atbalsta stāvokļu diagrammas.

*XState* (https://xstate.js.org/) ir viena no JavaScript bibliotēkām, kurā ir pietiekami plaši realizētas iespējas veidot stāvokļa diagrammas, kā arī var tikt izmantota ar tādiem ietvariem kā *Reac.js*, *Vue.js*, *Angular*, utt. *XState* bibliotēkā ir arī vizualizācijas rīks, kas palīdz gan atkļūdošanai gan arī kopējās shēmas novērošanai. Tiek uzskatīts, ka stāvokļu diagrammas ir efektīvāks veids lietotāja saskarnes ģenerēšanas aspektā.

Divpusložu modeļa procesi dalās divās grupās – iekšējie procesi un ārējie procesi, kuriem var norādīt procesa veicēju "<*performer*>", kas faktiski sastāv no nosaukuma. Tabulā 3.6. ir redzams iekšējā procesa XML definīcija, savukārt, 3.7. tabulā tiek demonstrēts ārējais process. Tehniski vienīgā atšķirība ir tā, ka ārējam procesam ir cits tips un tam var norādīt procesa veicēju sarakstu. Turklāt, neskaitot iekšējā un ārējā procesa definīciju, divpusložu modelis glabā arī savienojumus jeb plūsmas starp procesiem, kas tiek glabātas mezglā "<*dataFlows*>" un katra plūsmas definīcija ir iekļauta iekš "<*connection*>". 3.8. tabulā ir redzama "<*connection*>" definīcija.

**3.6. tabula**

**Divpusložu modeļa iekšējā procesa saturs**

| **Divpusložu modeļa procesa elements** | **Piemērs** | **Apraksts** |
|---|---|---|
| <id> | 1 | Procesa identifikators |
| <name> | List Items | Procesa nosaukums |
| <description> | Product for internet shop | Procesa apraksts |
| <comment> | - | Procesa komentārs |
| <type> | PROCESS_INTERNAL | Procesa tips |

**3.7. tabula**

**Divpusložu modeļa ārējā procesa saturs**

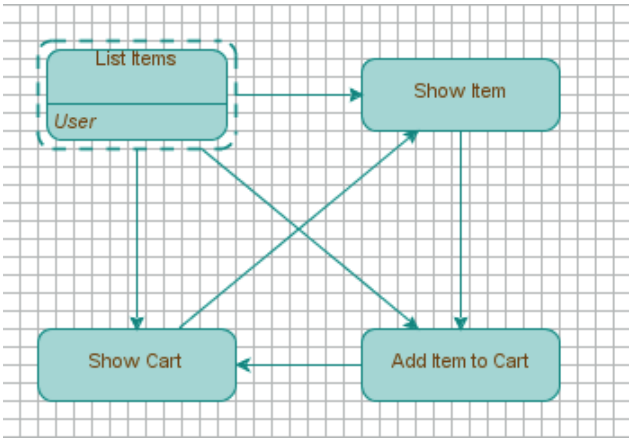
| **Divpusložu modeļa procesa elements** | **Piemērs** | **Apraksts** |
|---|---|---|
| <id> | 2 | Procesa identifikators |
| <name> | Show Cart | Procesa nosaukums |
| <description> | Simple external process | Procesa apraksts |
| <comment> | - | Procesa komentārs |
| <type> | PROCESS_EXTERNAL | Procesa tips |
| <performers> | <performer> | Procesu veicēju kopa |

**3.8. tabula**

**Divpusložu modeļa datu plūsmas definīcija**

| **Divpusložu modeļa procesa elements** | **Piemērs** | **Apraksts** |
|---|---|---|
| <elementId> | 3 | Plūsmas identifikators |
| <sourceId> | 1 | Avota procesa identifikators |
| <targetId> | 2 | Mērķa procesa identifikators |

3.5. attēlā ir redzams divpusložu modeļa procesu fragments, kurš sastāv no tādām darbībām kā apskatīt preču sarakstu (angl. *List items*), skatīt preci (angl. *Show item*), pievienot preci grozam (angl. *Add item to cart*) un rādīt preču grozu (angl. *Show cart*). Turklāt gan no saraksta skatījuma, gan no preces detalizētā skatījuma var preci pievienot grozam. Tāpat preci var pievienot grozam gan no preču saraksta skatījuma, gan no preces detalizētā skatījuma.



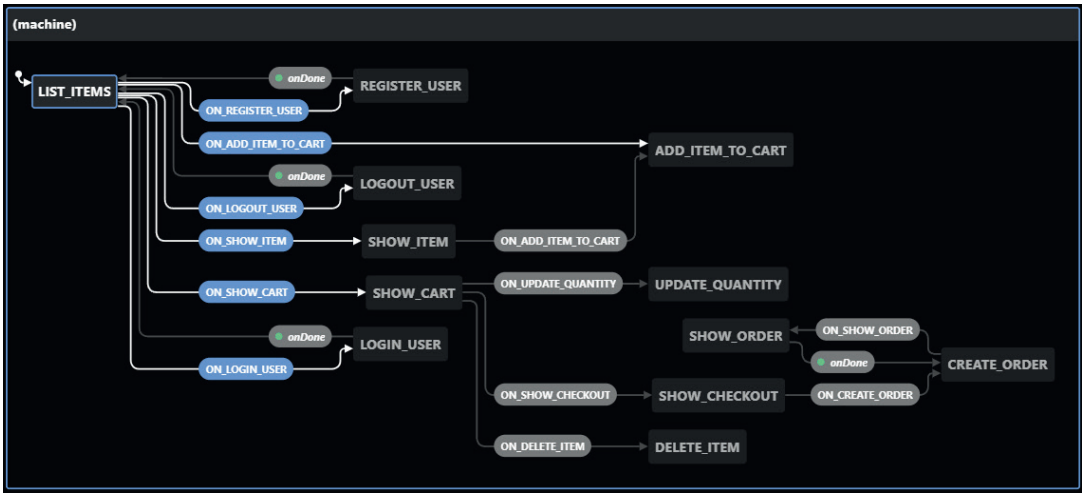

**3.5. att. Divpusložu modeļa procesu fragments**

Lai veiktu transformāciju uz stāvokļa diagrammu, ir nepieciešams noteikt kādos stāvokļos sistēma var atrasties. Pēc dotā divpusložu modeļa procesu piemēra var secināt, ka sistēma var atrasties 4 stāvokļos. Katrā stāvoklī (izņemot stāvokli "*Add Item to Cart*") ir sava vizuālā reprezentācija, tas nozīmē, ka ir nepieciešams papildināt divpusložu modeļa notāciju ar to kā process ir jāinterpretē – vai nu tas ir stāvoklis ar savu skatījumu vai tas ir starpstāvoklis, faktiski jāinterpretē kā notikums. Ņemot vērā šīs zināšanas var izveidot stāvokļu kartēšanu no zināmā divpusložu modeļa fragmenta (3.9. tabula) un attiecīgā stāvokļu diagramma ir parādīta 3.6. attēlā.

**3.9. tabula**

**Divpusložu modeļa procesa kartēšana uz stāvokļa diagrammas elementiem**

| **Stāvoklis** | **Notikums** | **Nākošais stāvoklis** |
|---|---|---|
| List Items | ON_SHOW_ITEM | Show Item |
| | ON_ADD_TO_CART | Show Cart |
| Show Item | ON_ADD_TO_CART | Show Cart |
| Show Cart | ON_SHOW_ITEM | Show Item |




**3.6. att. No divpusložu modeļa piemēra izgūtā stāvokļa diagramma**

Turklāt, "Add Item to Cart" no lietotāja saskarnes pēc būtības ir process jeb notikums, tad nepieciešams to izlaist un analizēt nākošo procesu, šajā gadījumā "Show Cart". Ja starpstāvoklis var tālāk virzīties uz vairāk nekā vienu procesu, tad tas ir jāņem vērā projektējot divpusložu modeļa procesu modeli, lai nebūtu lieki jāpapildina notācija. Darbā piedāvātajā piemērā izgūtā stāvokļa diagramma no divpusložu modeļa ir redzama 3.6. attēlā. Diagrammā var redzēt divpusložu modeļa procesa ceļu izmantojot piedāvāto interneta veikala piemēru. Šī stāvokļa diagramma ir izgūta izmantojot BrainTool kā divpusložu modeļa veidošanas rīku un XState bibliotēku, kartējot divpusložu modeļa procesus izmantojot augstāk minēto pieeju.

### 3.2.2. Transformācijas likumi navigācijas elementu ģenerēšanai

Pamatojoties uz stāvokļa diagrammas izveidi katram stāvoklim var veidoties atsevišķi ekrāni kā karkasi un tie tiek savienoti, izmantojot lietotāja plūsmu (angl. *User Flow*). Darbā piedāvātajā gadījumā lietotāja plūsma sastāvēs no stāvokļa diagrammas.

Katrs stāvoklis tiek attēlots kā ekrāns vai lapa. Katra pāreja no viena stāvokļa citā tiek transformēta par maršruta vienumu. Navigācijas maršruti tiek aktivizēti vai nu uzklikšķinot uz saites vai iesniedzot formu (*Create* vai *Update* darbības). Katram ģenerētam maršrutam tiek piesaistīts URI atslēgvārds (/login, /show_cart, utt.). Tomēr ir nepieciešams pseidostāvoklis

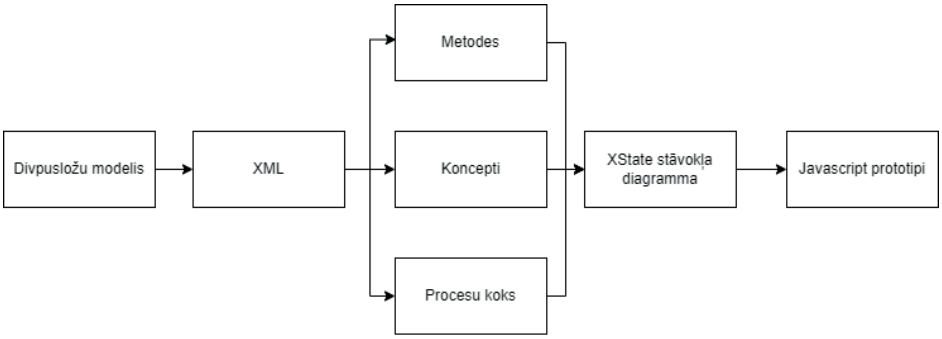
"*Initial*", lai noteiktu, kura lapa tiek rādīta pēc noklusējuma – navigācijas maršruts vai sākumlapa. Katrai stāvokļu pārejai tiek piesaistīts kāds aktivizēšanas notikums – nosaukumam ir arī prefikss ar notikuma veidu, piemēram, *onClick* vai *onCreate*. Stāvokļa pārejas notikuma izsaucējs pēc noklusējuma ir *onClick*. Ja, piemēram, pogas nosaukums ir tāds pats kā mērķa lapai, tad tas automātiski tiek sasaistīts. Visi notikumu tiek izsaukti avota lapā. *Create* un *Update* notikumi tiek aktivizēti, kad tiek iesniegta forma avota lapā. Var izdalīt katru elementu kā CRUD kopu un attiecīgi ar to mijiedarboties, automātiski veidojot skatījumus, kas būs attiecināmi uz kād no funkcijām. 3.10. tabulā ir redzams divpusložu modeļa procesu elementu kartēšana ar stāvokļu diagrammas elementiem.

**3.10. tabula**

**Divpusložu modeļa procesu elementu kartēšana ar stāvokļa diagrammas elementiem**

| **Procesa elements** | **Stāvokļa diagrammas elements** |
|---|---|
| Nosaukums | Stāvokļa nosaukums |
| Nākošo procesu nosaukums | Stāvoklim piešķir pāreju ar prefiksu "ON_" |
| Nākošo procesu nosaukums | Izveido nākošo stāvokli |
| Ja nav nākošā procesa | Izveido pāreju onDone |

3.7. attēlā ir redzama kopējā shēma stāvokļa diagrammas izgūšanai no divpusložu modeļa. Pirmajā solī tiek izgūta XML datne no *BrainTool* rīka, kurā ir izstrādāts divpusložu modelis. Tālāk, tiek izgūti dati no XML datnes un normalizēti tā, lai varētu izgūt procesu koku (kādi stāvokļi ir pieejami un kur var nokļūt tālāk), koncepti, kuri tālāk tiek nodoti stāvokļa diagrammai kā konteksts un metodes, ja tādas ir izstrādātajam divpusložu modelim. Metodes tiek attiecīgi pievienotas stāvokļa diagrammai pie konteksta. Stāvokļa diagrammas kontekstam tiek izveidoti katram konceptam izstrādātie atribūti un pievienotas vērtības pēc atribūtu datu tipiem.




**3.7. att. Stāvokļa diagrammas izgūšana no divpusložu modeļa**

Divpusložu modeļa procesa modelis pēc būtības ir ļoti tuvs stāvokļu diagrammai, jo tas sevī satur stāvokļus (procesus), pārejas starp tiem (savienojumi), bet nesatur informāciju, kura norādītu vai esošais stāvoklis ir lapa, kuru nepieciešams demonstrēt lietotājam vai arī, no lietotāja saskarnes skatupunkta, ignorējams stāvoklis.

Autors darbā apvieno divpusložu modeli ar *xState* stāvokļa diagrammām un *React Router* deklaratīvai maršrutēšanai *React.js* lietojumprogrammās. Būtībā var izmantot arī citas bibliotēkas un satvarus – tas nemaina principa būtību. 3.11. tabulā ir redzama darbā apskatāmo komponentu kartēšana savā starpā, lai vispārīgi izprastu būtību.

**3.11. tabula**

**Darbā apskatāmo komponentu kartēšana**

| **Divpusložu modeļa elementi** | **XState elementi** | **React Router elementi** |
|---|---|---|
| Procesa nosaukums | Stāvokļi (angl. *States, S*) | Maršruts (angl. *Route, R*) |
| Datu plūsma | Pārejas (angl. *Transitions, T*) | Pārejas |
| - | Sargi (angl. *Guards, G*) | Sargi |
| Secību diagrammas metodes | Darbības (angl. *Actions, A*) | Sargi |
| Koncepta atribūti | - | Maršruta parametri |

Lai veiktu kartēšanu komponentu starpā, sākumā ir nepieciešams identificēt funkcionālos stāvokļus un maršrutus. To var aprakstīt šādos soļos:

1. Ar katru divpusložu modeļa procesa elementu:

a. Izveidot stāvokli (S)
b. Izveidot maršrutu (R) ar URL ceļu uz stāvokli

2. Ar katru divpusložu modeļa procesa datu plūsmu:
   a. izveidot pāreju (T) ar lietotāja darbību, kā izraisošo notikumu (angl. *Trigger*)
   b. noteikt datu plūsmas konceptu un definēt maršruta parametru, ja nepieciešams
3. Ar katru funkcionālo stāvokli, definēt "*onEnter*" darbību (A), kura atjaunināsURL izmantojot *React Router* "*history.push*" metodi lai parādītu esošo stāvokli (maršrutu)
4. Pārvaldīt maršruta pieejas
   a. Izmantot *React Router* komponentu atribūtus, tādus kā "path" un "exact" priekš precīziem maršrutiem
   b. Nepieciešamības gadījumā izmantot *React Router* "*Loader*" komponentu vai sadalīt maršrutus uz konteksta pamata.

Kā piemēru dotajai metodei tiek piedāvāts interneta veikals ar preču katalogu un izvēlētā produkta detalizēto skatījumu. Pēc noklusējuma tiek izsaukts pirmais stāvoklis, kurš saucas "*List Items*", aktivizējot pārejas saiti, lietotājs tiek novirzīts uz "*Show Item*" stāvokli. 3.12. tabulā ir redzams elementu kartējums.

**3.12. tabula**

**Interneta veikala fragmenta kartēšanas piemērs**

| Divpusložu modeļa process | XState stāvoklis | React Router maršruts |
|---|---|---|
| List Items | ListItemsState | /list_items |
| Show Items | ShowItemState | /show_item |

Šāda pieeja strukturē lietotāja mijiedarbību, jo *XState* definē lietotāja mijiedarbību ar skaidriem stāvokļiem un pārejām. Kā arī deklaratīva maršrutēšana ar *React Router* vienkāršo lietotāja saskarnes navigāciju, pamatojoties uz URL izmaiņām un sniedz turpmākas deklaratīvas iespējas izmantošanu izstrādē. Šāda metode var apstrādāt gan funkcionālās (lietotājam neredzamas), gan prezentācijas daļas (lietotājam redzamie stāvokļi), tomēr šajā gadījumā ir nepieciešams papildināt divpusložu modeli ar iespēju noteikt vai esošais process ir jāparāda

lietotājam vai nē. Lielām lietojumprogrammām stāvokļu pārvalde var kļūt sarežģīta un iespējams, nepieciešams padomāt, kā šos stāvokļus, maršrutus un pārejas veidot modulārākas.

### 3.2.3. Divpusložu modeļa papildināšana

Sakarā ar to, ka divpusložu modeļa procesiem nevar norādīt vai esošais process ir stāvoklis ar savu prezentācijas veidni vai tas ir starpstāvoklis (pāreja), tad nepieciešams divpusložu modeļa procesu papildināt ar tipu, kas no lietotāja saskarnes puses, noteiktu vai šim procesam ir sava veidne, ko nepieciešams attēlot vai arī tas ir process, kas lietotājam nav jāredz. Par šādas iespējas piedāvāšanu tiek diskutēts nākošajā sadaļā, kur tiek apskatīta prezentācijas izgūšana.

Divpusložu modeli var nepapildināt ar šādu tipa atribūtu, ja nākošajā sadaļā – prezentācijas sadaļā – var noteikt vai esošais process ir demonstrējams lietotājam vai arī tam nav jābūt redzamam lietotājam. Šāda pieeja varētu atvieglot divpusložu modeļa izstrādi un uzturēšanu.

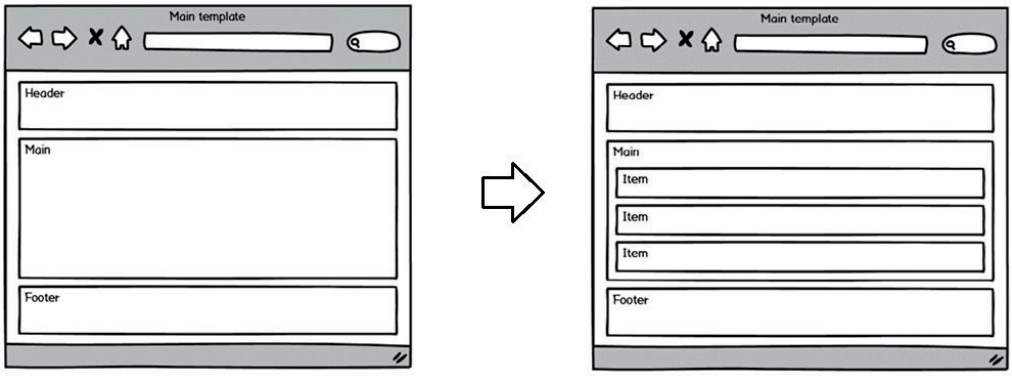
## 3.3. Prezentācija

Prezentācijas projektējums iekļauj abstrakta lietotāja saskarnes modelēšanu, parādot, kā lietotājam tiek demonstrēta navigācijas struktūra (Koch & Mandel, 1999). Prezentācijas izstrāde nosaka veidu, kā parādīsies navigācijas mezgli, atlasīt lietotāja saskarnes objektus, lai aktivizētu navigāciju, un noteikt, kuras saskarnes transformācijas notiks. Viena un tā pati navigācijas struktūra var nodrošināt dažādas prezentācijas atkarībā no mērķa platformas ierobežojumiem un izmantotās tehnoloģijas (Koch & Mandel, 1999).

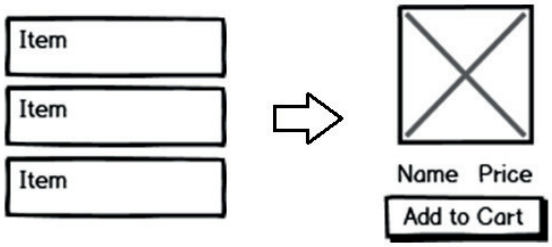
Līdzīgi kā (Wolff, Forbrig et al., 2005), kur automātiski ģenerētās XUL (angl. *XML User Interface Language, XUL*) specifikācijas tiek pilnveidotas, izmantojot lietotāja saskarnes redaktoru, kas ļauj aizstāt lietotāja saskarnes elementus ar citiem elementiem vai komponentiem. Komponenti ir iepriekš izstrādātas lietotāja saskarnes daļas, un tām var pēc izvēles noteikt papildus parametrus (Li, Popowski et al., 2021). Tas ļauj atbalstīt lietotāja saskarnes izstrādi, izmantojot modeļus. Citā pētījumā (Koch, Baumeister et al., 2000) tiek definēti stereotipi, kuri tiek izmantoti atbilstošos modeļos, kas atbalsta tīmekļa lietojumprogrammu analīzi un izstrādi. Tie ir daļa no UML balstītas metodoloģijas (Hennicker & Koch, 2001), kas ļauj sistemātiski izveidot tīmekļa

lietojumprogrammas, t.i., konceptuālo modeli, navigācijas modeli, kas balstīts uz konceptuālo modeli, un prezentācijas modeli, kas izmanto navigācijas modeli par pamatu.

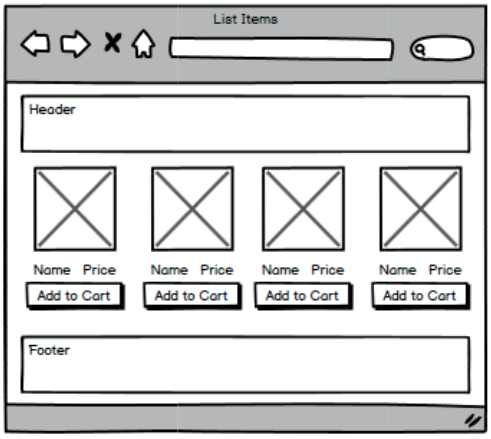
Lietotāja saskarnes prezentācijai tiek izmantots "atomu" projektēšanas modelis (skatīt "Atomu" projektēšanas ideja), ar ko tiek izvietoti pamat elementi hierarhiskā secībā veidojot augstāka līmeņa elementus, kurus izstrādātājs var norādīt modelī. Zem katra augstākā līmeņa elementa tiek saprasts noteikta satvara (*Bootstrap*, *Material Design*, utt.) elementu kopums. Darba piedāvātajā risinājumā izdala galveno veidni, kurā tiek izvietoti citi elementi (Elec, Uyanik et al., 2020). Veidnes shematisks attēlojums ir redzams 3.8. attēlā. Skatoties attēlu no kreisās puses uz labo, var ievērot, ka sākumā ir veidne ar 3 elementiem "Header", "Main" un "Footer". Mainoties navigācijai tiek izmainīts "Main" lauka saturs, šajā gadījumā, procesā "List Items", tiek ievietots preču saraksta skatījuma šablons. Pielietojot jau augstāk minēto "atomu" projektēšanu (3.9. att.) un zem katra preces saraksta vienuma paredzot tā konceptuālo šablonu, tiek iegūts jau detalizētāk attēlotais preču vienumu saraksts. Turpinot transformācijas procesu, katrs saraksta vienums tiek transformēts detalizētāk un iegūts jau detalizēts preču saraksts, kas redzams 3.10. attēlā.




**3.8. att. Augstākā līmeņa lietotāja saskarnes veidne, kurā tiek ievietots preču saraksts**




**3.9. att. Shematiska preces saraksta vienuma transformācija**

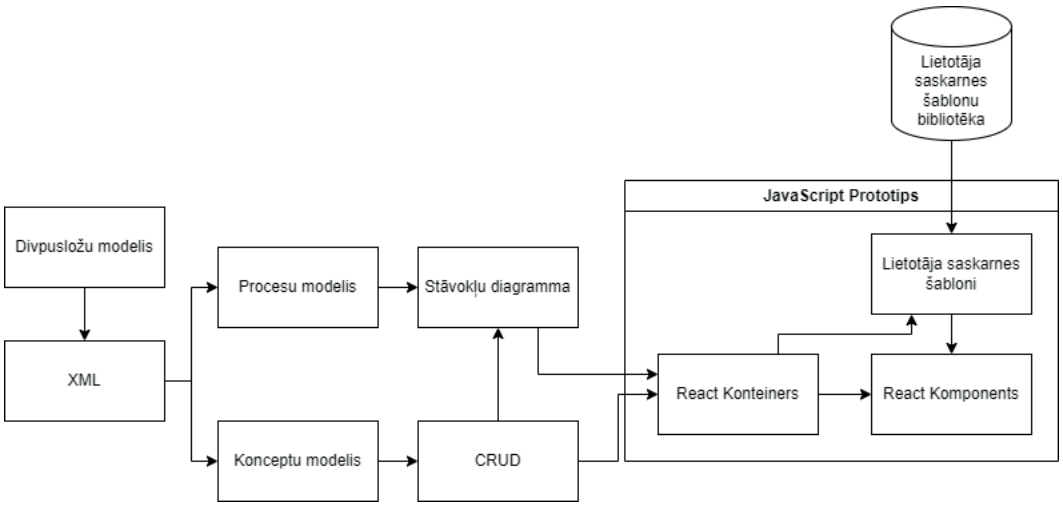



**3.10. att. Detalizēts preču saraksts**

Zinot datu modeli (konceptu) un no kurienes uz kurieni var aiziet no esošā stāvokļa (navigācija), var izmantot datu modeli, kā iespējamo lauku attēlošanu un ar esošo stāvokli noteikt kādas ir iespējamās darbības. No tā ir iespējams noteikt vai esošajā stāvoklī nepieciešams demonstrēt formu, detalizētu skatījumu vai sarakstu. Ja esošā stāvokļa lapā tiek demonstrēts koncepts, kuram ir piesaistīti citi koncepti, tad tiek demonstrēts attiecīgs šablons, kurā galvenās entītijas detalizētais skatījums tiek demonstrēts kopā ar piesaistīto entītiju sarakstu.

Izmantojot lietotāja saskarnes šablonus, kurus nosaka pēc stāvokļa nosaukuma, iespējamiem nākamajiem stāvokļiem un datu modeļa (koncepta), iespējams demonstrēt reaģējošu lietotāja saskarni (angl. *Responsive User Interface*), kas tiktu vienlīdz labi attēlota gan tīmekļa lietotnēs, gan mobilajās lietotnēs. Demonstrējamo elementu izmērs un pozīcija tiek noteikta pēc Material-UI un lietotāja saskarnes šabloniem. Etiķešu un teksta lauku pārus, režģi (angl. *Grid*),

atsevišķu etiķešu vai pogu izlīdzināšanu, utt. visu šo iespējams demonstrēt ar iepriekš definētiem lietotāja saskarnes šabloniem. 3.11. attēlā ir redzama kopējā shēma saskarnes skatījumu izgūšana no divpusložu modeļa, kas sevī kombinē gan saskarnes satura elementus (konceptus), gan navigācijas elementus. Pirmajā solī tiek izgūta XML datne no *BrainTool* rīka, kurā ir izstrādāts divpusložu modelis. Tālāk, tiek izgūti dati no XML datnes un izgūti procesa un koncepta modeļi. No koncepta modeļa tiek izveidota CRUD kartēšana un paralēli, izgūstot stāvokļa diagrammu, var izveidot saprotamus nosaukumus konteineriem un komponentiem.




**3.11. att. No divpusložu modeļa izgūstamo skatījumu kopējā shēma**

*React.js* konteiners atbild par datu un procesu kartēšanu priekš *React.js* komponenta, kas faktiski ir statisks skatījums, kurš tikai attēlo datus, bet visus notikumus apstrādā *React.js* konteiners. Izmantojot esošo stāvokļu diagrammu un koncepta modeli ar CRUD kartēšanu var izveidot lietojamu lietotāja saskarnes prototipu, tomēr, ir iespēja vēl uzlabot šos skatījumus pievienojot jaunus elementus, lai iespējotu attēlot sarežģītus un ar elementiem bagātākas lietotāja saskarnes.

Jaunais elements, kas ir parādījies šajā shēmā, ir nepieciešamība pēc lietotāja saskarnes šabloniem. Eksistē vairākas pieejas saskarnes šablonu veidošanā. Tās ir izpētītas nākamajā sadaļā.

## 3.4. Lietotāja saskarnes šabloni

No problēmvides lietošanas gadījumiem var automātiski ģenerēt lietotāja saskarnes modeļus un prototipus izmantojot transformējošu pieeju programmatūras izstrādē. Šāds programmatūras izstādes mērķis ir racionalizēt lietotāja saskarnes procesu. Metode par pamatu ņem esošo problēmvidi un lietošanas gadījumu modeļus, kurus transformējot, iegūst automātiski ģenerētus lietotāja saskarnes elementus (Cruz (da), Miguel et al., 2010a). Izmantojot šādu pieeju var samazināt manuālo piepūli, kļūdas un arī uzlabotu izstrādēs efektivitāti. Neskatoties uz to, ka šai pieejai ir daudzas priekšrocības – tai ir arī vairākas problēmas un trūkumi, pie kuriem vēl nepieciešams strādāt (Cruz (da), Miguel et al., 2010a).

Viena no galvenajām problēmām automātiskajā lietotāja saskarnes ģenerēšanā ir radoša potenciāla zaudēšana – metode ir tehniska, nevis mākslinieciska. Izmantojot automātiskus rīkus un dažādus predefinētus šablonus (Molina, Meliá et al., 2002), izstrādātāji var palaist garām kādas unikālas dizaina iespējas vai arī rezultējošā lietotāja saskarne neatbildīs gala lietotāja vajadzībām (Dubey, 2011). Tas attiecīgi var radīt situāciju, kurā gala lietotāji neizmanto šo saskarni.

Cita problēma ir automātiskā lietotāja saskarnes rīka ģenerēšanas iespējas – cik lielā mērā būs iespējams iegūt saskarni, kura pilnā mērā atbildīs gala lietotāja vēlmēm un lietojamībai. Šādi automātiski ģenerējoši rīki un metodes var noteikt kā prioritāti tehniskas definīcijas, nevis gala lietotāja ērtību (Falb, Popp et al., 2007). Šādās lietotāju saskarnēs var būt grūti orientēties, tās var mulsināt vai radīt citas lietotāja un programmatūras komunikācijas problēmas. Risinājums var būt daļēji manuāli iejaukties ģenerēšanas procesā, piešķirot procesiem šablonus vai veikt kādas citas korekcijas (Nichols, Chau et al., 2007).

Faktiski lietotāja saskarnes un prototipu kvalitāte, automātiski ģenerētajā pieejā, ir atkarīga no tā cik kvalitatīvi un pilnīgi ir ieejas dati – problēmvides un lietošanas gadījumu modeļi. Ieejas datos ir jābūt precīzi definētām lietotāja saskarnes ģenerēšanai nepieciešamajām funkcijām. Citiem vārdiem sakot, lai ģenerētu lietotāja saskarni no modeļiem ir nepieciešami pilnīgi, pareizi konstruēti un precīzi modeļi (Antović, Vlajić et al., 2012).

Neskatoties uz dažādiem izaicinājumiem, lietotāja saskarnes automātiska ģenerēšana dod vairākas vērtīgas priekšrocības, kā piemēram – samazināts lietotāja saskarnes izstrādei nepieciešamais laiks un resursi, kas ļauj izstrādes komandai koncentrēties uz funkcionalitātes

izstrādi, arhitektūru un veiktspējas optimizēšanu. Automātiska lietotāju saskarnes ģenerēšana, piemēram, izmantojot šablonus, var veicināt arī konsekvenci un standartizāciju visa projekta ietvaros, kas, savukārt, paredz saskarnes elementu atbilstību izveidotajiem dizaina modeļiem un principiem.

Šādā vai tādā veidā, programmatūras izstrādei attīstoties, lietotāja saskarnes izstrāde kļūst uz šabloniem orientēta, balstīta uz lietotāja ērtību (Diehl, Marins et al., 2022) un aizvien vairāk – automātiski ģenerēta. Eksistē dažādi lietotāja saskarnes izstrādes šabloni un tos aizvien vairāk pielieto programmatūras izstrādē. Tādēļ var teikt, ka automātiskai lietotāja saskarnes ģenerēšanai ir izšķiroša nozīme, kas paātrina programmatūras izstrādes dzīves ciklu un palielina kopējo projekta efektivitāti.

Jāpiemin, ka parasti lietotāja saskarnes šabloni tiek parādīti ar noteiktu struktūru vai uz šablona, tomēr, nav vienotas noteiktas struktūras to prezentēšanai, bet dažādi autori ir izveidojuši savu prezentācijas struktūru (Alarcon, Balcazar et al., 2022). Ir mēģinājumi strukturēt lietotāja saskarnes šablonus izmantojot šablonu valodas (Welie & Veer, 2003), kā arī sadalīt tos kategorijās un definēt to pielietošanas problēmvidi (Toxboe, 2024; Tidwell, 2020; Obrist, Tscheligi et al., 2010).

Svarīgi ir automatizēt atkārtotus un laikietilpīgus lietotāja saskarnes projektēšanas uzdevumus – daudzi no tiem jau ir tikuši izstrādāti pirms tam, tiem ir izveidotas labās prakses vai tie ir definēti kādā no lietotāja saskarnes izstrādes satvariem. Tādā veidā komandas var ātrāk veikt iterācijas un izmantojot lietotāju atsauksmes (Diehl, Martins et al., 2022) – savlaicīgi piegādāt augstas kvalitātes programmatūras produktus.

Lietotāja saskarnes modeļu un prototipu automātiska ģenerēšana paplašina efektivitātes un konsekvences uzlabošanas iespējas. Tomēr, ir svarīgi arī risināt ar tiem saistītos izaicinājumus un ierobežojumus – meklēt veidus kā uzlabot un vēl vairāk automatizēt šo procesu. Šobrīd, līdzsvarojot automatizāciju kopā ar manuālu projektēšanu var palielināt automātiskas ģenerēšanas priekšrocības un nodrošināt lietotājam draudzīgu lietotnes saskarni. Lai vēl uzlabotu šo procesu, nepieciešams pārskatīt uz lietotāju vēstos projektēšanas principus un ieguldīt precīzos un visaptverošos problēmvides un lietošanas gadījumu modeļos, papildinot tos ar dažādām labajām praksēm un procesu šabloniem.

Daži pētījumi pat aptver dažādu ierīču, mobilo pakalpojumu, platformu un operētājsistēmu dizaina šablonus, sniedzot dažādus piemērus atkarībā no lietošanas konteksta (Nguyen, Thanh-Diane et al., 2016; Engel, Herdin et al., 2012). Eksistē arī aptauju rezultāti, kuros tiek sniegtas rekomendācijas dizaina izstrādē, pamatojoties uz dažādiem lietotāju saskarņu tipiem, pielietošanas un izmantošanas tradīcijām (Diehl, Martins et al., 2022). Galvenās bažas rada tas, ka tie nesniedz nekādus metodiskus norādījumus par šo modeļu piemērošanu vai jebkādu izstrādes metodi, lai izstrādātu saistīto kodu.

Šablonus var hierarhiski strukturēt, sākot no mazākām šablonu vienībām, piemēram, sarakstiem, pogām, utt., beidzot ar veseliem procesiem, kā piemēram, interneta veikaliem. Vienā no citu autoru piedāvātajām pieejām (Kruschitz & Hitz, 2010), lietotāja saskarnes šabloni tiek strukturēti pēc līmeņiem un uzdevumiem.

Kā vienu no populārākajiem un bieži izmantotiem piemēriem ir "*Master-And-Details*" šablons (Molina, Meliá et al., 2002; Moreira, Paiva et al., 2013), kas parāda galveno sarakstu (piem., produktu sarakstu) un pēc tam detalizētu informāciju par jebkuru pašlaik atlasīto vienumu (piemēram, konkrēto produktu) (Nguyen, Thanh-Diane et al., 2016).

"*Master-And-Details*" šablonā (Moreira, Paiva et al., 2013) tiek parādīts galvenais elements (angl. *Master*), kas sastāv no virknes vienumu, kuru detaļas pēc pieprasījuma tiek nodotas. Problēmvides modelis, kurā galveno elementu raksturo savi atribūti (tipiski atribūti, piemēram, ID, nosaukums utt., kā arī privātie atribūti) un savas metodes (tipiskas metodes, kas iegūtas no CRUD) (Nguyen, Thanh-Diane et al., 2016) modelis un privātās metodes). Tāpat jebkuram galvenās kolekcijas vienumam ir savi atribūti (tipiski atribūti un specifiski vienumu atribūti) un metodes (tipiskas metodes un specifiskas vienumu metodes).

Līdzīgi "*Master-And-Details*" šablonam ir piedāvāti, piemēram, interneta veikaliem specifiski šabloni (Fernández, Liu et al., 2021):

1. Kataloga šablons (Fernández, Liu et al., 2021) – organizē informāciju par tīmekļa vietnē pārdotajiem produktiem. Kā risinājums tiek piedāvāts definēt kataloga klasi kā galveno produktu savākšanas punktu. Atdalīt dažādus interesējošos aspektus, piemēram, detalizētus produktu un saistīto produktu aprakstus dažādās klasēs. Kataloga modelis ietver konteinera/konteksta šablonu. Citi šabloni, kas parasti atrodami kopā ar katalogu, ir meklēšanas šabloni

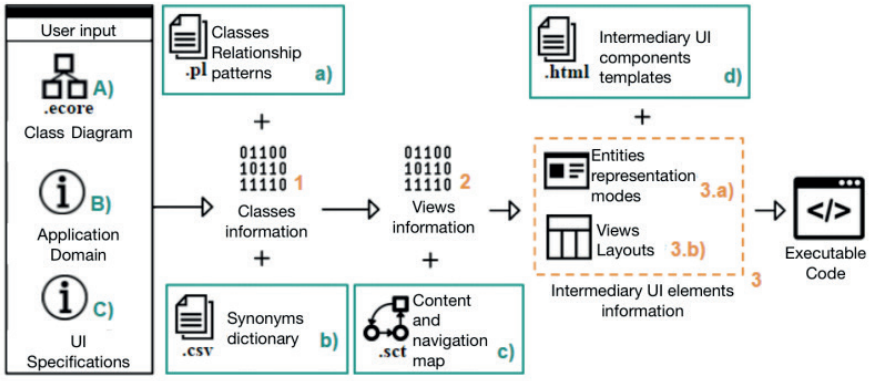
2. Iepirkšanās procesa šablons (Fernández, Liu et al., 2021) – Interneta veikalos ir daudz preču. Klienti var izvēlēties un iegādāties dažādus produktus vienā sesijā. Klients meklē sev interesējošo preci, izvēlas kaut ko, ko vēlas iegādāties, un pievieno to savam pirkumu sarakstam. Viņš jebkurā laikā var pārbaudīt un mainīt savus pirkumus.
3. Interneta veikalā var būt gan katalogi, gan rēķini, gan arī citas iespējas – piemēram, piegāde, norēķini, atlaides, utt. Daudzas no šīm iespējām var jau būt pazīstamas lietotājiem un gala sistēmas izmantotājiem – pazīstami procesi un šabloni, padara lietotāja pieredzi vieglāku un saprotamāku. Preču izvēles un norēķina procesam ir jābūt precīzi definētiem, jo ir nepieciešams parādīt klientam, kur viņš šajā procesā atrodas un kas viņu gaida tālāk. Faktiski, problēmu var definēt tā, ka nepieciešams atbildēt uz jautājumu kā precīzi aprakstīt iepirkšanās procesu. Visizplatītākais un bieži sastopamais šablons ir elektroniskais iepirkuma grozs – klientam var būt vairākas preces ar dažādiem atribūtiem (piemēram, skaits), kuras tiek glabātas iepirkuma grozā, līdz klients ir izvēlējies noformēt to kā pirkumu. Tad tiek sagatavots pasūtījums un sastādīts rēķins.

Pētījumā (Fernández, Liu et al., 2021) piedāvātajos šablons, interneta veikalu kontekstā, tiek piedāvāti arī citi šabloni, tādi kā navigācijas šablons, pasūtījuma un piegādes šabloni, norēķinu šabloni, utt. Citos pētījumos ir pieteikšanās (angl. *Login*), reģistrēšanās u.c. šabloni (Ko, Park et al., 2023).

Faktiski, tiek norādīts uz to, ka eksistē dažādi atsevišķi šabloni, kurus var apvienot vienā satvarā, tādējādi definējot kā vajadzētu izstrādāt interneta veikalus. No vienas puses tas nozīmīgi attēlo izstrādi, jo izmantojot standartizētu satvaru, var veikt transformācijas starp modeļiem vai citiem avotiem un mērķiem, zinot kas būtu jāsaņem gala rezultātā. No otras puses, standartizēts interneta veikals vienmēr nevar apmierināt gala lietotāju – prasību specifikācija var būt atšķirīga no standartizētā piedāvājuma.

Šablonu izmantošana ir pieejama dažiem risinājumiem, piemēram, *MODUS* – uz MDE un MBUID balstīta pieeja, kas nodrošina lietotāju saskarnes ģenerēšanu no modeļiem ar augstu automatizācijas līmeni. *MODUS* izmanto klases diagrammas un lietojumprogrammas problēmvidi kā pamata elementus (3.12. att.). Klases diagramma definē entītijas, kuras tiks izvietotas lietotāja saskarnē. Lietojumprogrammas problēmvide sniedz informāciju, lai veiktu vairākus pieņēmumus par lietotāja saskarni, izvairoties no nepieciešamības izveidot pilnīgus lietotāja saskarnes modeļus.

Entītiju prezentācijas režīmi un lapu izkārtojumi ievieš satura un stila atdalīšanas principu. Tādejādi veidnes (*angl. Templates*) nosaka estētiku un strukturālās detaļas, atbalstot sarežģītākas un pilnīgākas lietotāja saskarnes (Machado, Couto et al., 2017). *MODUS* darbojas kā *Eclipse* spraudnis.




**3.12. att. *MODUS* kopējā darbības shēma (aizgūts no (Machado, Couto et al., 2017))**

Mūsdienās, industrijā tiek aktīvi izmantots lietojumprogrammu izstrāde kopā ar kodu ģeneratoriem (Elec, Uyanik et al., 2020). Tas pierāda, ka var panākt efektīvu uz šabloniem balstītu koda ģeneratora integrāciju reālajā dzīvē – liela mēroga tīmekļa lietojumprogrammu izstrādē. Turklāt eksperimentējot tika parādīta automātiskās koda ģenerēšanas efektivitāte, kā arī eksperimentu laikā tika novērota koda ģenerēšana bez kļūdām (Elec, Uyanik et al., 2020).

Kā nākošais solis būtu jāizveido lietotāja saskarnes šablonu bibliotēku projektam, kura saturētu visas vairākkārt lietojamu procesu šablonus ar lietotāja saskarnes elementu kartējumu. Šai bibliotēkai nepieciešams būt organizētai pamatojoties uz "atomu" projektēšanas principiem, lai būtu viegli un ērti atrast un pielietot nepieciešamos komponentus.

Lietotāja saskarnes šabloni, iespējams, ir labāki par likumiem (angl. *Rules*) vai vadlīnijām (angl. *Guidelines*), jo tie ir skaidri saistīti ar kontekstu un ir vērsti uz problēmām (Elkoutbi, Khriss et al., 2006). Lietotāja saskarnes šabloni ir parādījušies kā iespējamais risinājums dažām problēmām, no kurām cieš vadlīnijas (Welie, Veer et al., 2000). Neskatoties uz to, ka tas ir konceptuāli, lietotāja saskarnes šablonu izveide praksē nav tik vienkārša (Elkoutbi, Khriss et al.,

2006). Lietotāja saskarnes šabloni koncentrējas uz problēmas un risinājuma kontekstu, tādējādi palīdzot dizainerim izmantot dizaina zināšanas. Arhitektūras vai programmatūras inženierijas modeļi pēc struktūras nav identiski, un lietotāja saskarnes dizains prasa arī savu lietotāja saskarnes šablonu struktūru, koncentrējoties uz lietojamību (Elkoutbi, Khriss et al., 2006). Lietotāja saskarnes šablons ne vienmēr būs strukturēts tādā pašā veidā kā arhitektūras modelis, un ir svarīgi atrast formātu, kas ir paredzēta lietotāja saskarnēm un kam ir pareizais skatījums uz svarīgām problēmām (Elkoutbi, Khriss et al., 2006). Šim nolūkam ir jādefinē jauni lietotāja saskarnes transformēšanas likumi vai jauni lietotāja saskarnes šabloni (Elkoutbi, Khriss et al., 2006; Vanderdonckt, 2001; Borchers, 2000).

### 3.4.1. Lietotāja saskarnes šablonu definēšana

Ideja par izstrādes šabloniem aizsākās astoņdesmito gadu beigās ar Kristofera Aleksanda (angl. *Christopher Alexander*) agrīnajiem darbiem (Alexander, 1979) arhitektūras jomā. Aleksandrs definēja izstrādes šablonus kā vispārīgus, nevis acīmredzamus, atkārtoti lietojamus risinājumus bieži sastopamām problēmām noteiktā kontekstā, kam ir praktisks raksturs un nozīmīgs cilvēciskais komponents, kas balstās uz izstrādes pieredzi, nevis abstraktiem principiem vai teorijām.

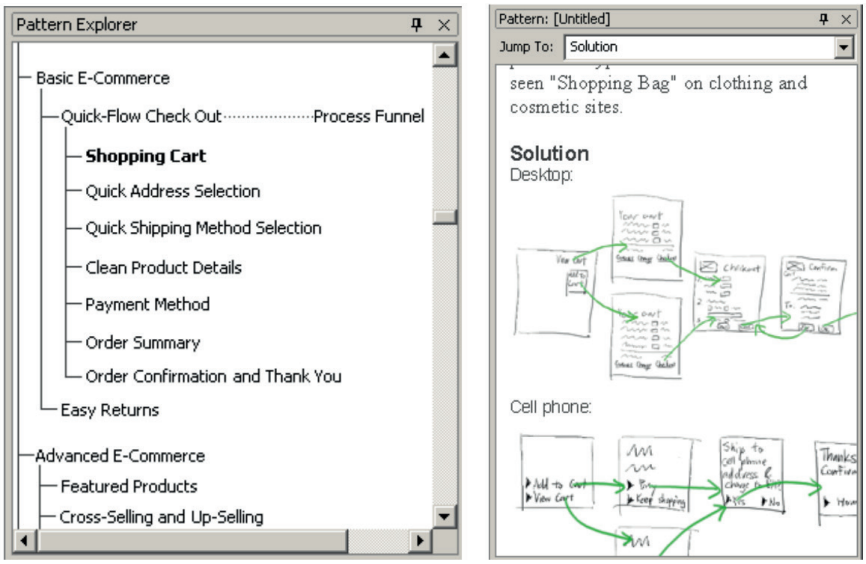
Kristofers Aleksandrs pirmo reizi definēja šablonus programmatūras arhitektūras kontekstā kā "pašreizējā labākā minējuma reprezentāciju par to, kāds fiziskās vides izvietojums darbosies, lai atrisinātu izvirzīto problēmu" (Alexander, Ishikawa et al., 1977). Pēc tam šī definīcija tika piemērota dažādās pētniecības jomās, tostarp programmatūras inženierijā.

Katrs šablons ir trīsdaļīgs likums, kas izsaka saistību starp noteiktu kontekstu, problēmu un risinājumu (Welie, Veer et al., 2000; Alexander, 1979). Aleksandrs arī definēja, ka "Katrs šablons apraksta problēmu, kas mūsu vidē atkārtojas atkal un atkal, un pēc tam apraksta šīs problēmas risinājuma būtību tādā veidā, ka jūs varat izmantot šo risinājumu miljons reižu…" (Welie, Veer et al., 2000; Alexander, 1979).

Lietotāja saskarnes šablons ir dizaina ideja vai šablons, kas tiek izmantota dažādās lietotnēs (vai dažādās vietās vienā lietotnē) (Nguyen, Vu et al., 2018). Piemēram, lielākajai daļai lietotņu ir pieteikšanās ekrāns ar lietotāja ID elementiem (piemēram, lietotājvārds, e-pasts vai tālruņa

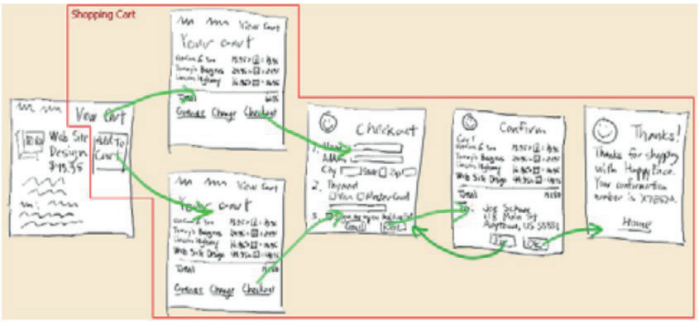
numurs), parole un dažreiz arī trešās puses autentifikācijas metodes (piemēram, pieteikšanās, izmantojot *Facebook*, *Google* vai *Twitter* kontus) (Nguyen, Vu et al., 2018).

Lietotāja saskarnes šabloni ir izrādījusies izdevīga forma zināšanu glabāšanai atkārtotai izmantošanai dažādās jomās (da Silva, Paulo, 2022; Wang, Song et al., 2022). Pētnieki un lietotāja saskarnes dizaineri apraksta šablonu noteiktā formātā, kas var atšķirties. Citā pētījumā (Welie, Veer et al., 2000), tiek doti piemēri kā definēt lietotāja saskarnes šablonus. Neskatoties uz to, ka lietotāja saskarnes šablonu vēsturē ir izveidotas vairākas kanoniskās formas (Kruschitz & Hitz, 2010; Suleri, Kipi et al., 2019), tās nav standartizētas un nav vienota formāta lietotāja saskarņu šablonu definēšanai. Piemēram specializētajā rīkā – *Damask* (Lin & Landay, 2002), ir izstrādāts lietotāja saskarnes šablonu pārlūks (3.13. att.), kurā var izvēlēties nepieciešamo lietotāja saskarnes šablonu.




**3.13. att. *Damask* rīka lietotāja saskarnes šablonu izvēles logs (aizgūts no (Lin & Landay, 2002))**

Apskatītajā pētījumā autori izvēlas iepirkuma groza risinājumu un pārvelk izvēlēto šablonu no šablonu pārlūka loga risinājuma logā. Tādējādi šīs lietotāja saskarnes šablona risinājums tiek sapludināts ar jau esošo procesa plūsmu un iepirkuma groza šablons tagad ir iekļauts esošajā projektējumā. 3.14. attēlā ir redzams Damask rīka e-komercijas vietnes versija, kurā ir ievietots iepirkumu groza šablons (Lin & Landay, 2002).




**3.14. att. *Damask* rīka e-komercijas vietnes versija, kurā ir ievietots iepirkumu groza šablons (aizgūts no (Lin & Landay, 2002))**

Aytaj Guliyeva savā maģistra darbā aprakstīja pētījumu par 120 lietošanas gadījumu analīzi, kas ietver dažādas sarežģītības pakāpes, kā arī dažādus scenārijus (Guliyeva, 2023). Guliyeva savā darbā apraksta trīs līmeņu šablonus (Guliyeva, 2023):

1. Sākotnējie šabloni (angl. *Initial Patterns*) – sākotnējā programmatūras projektu plūsma sākas no reģistrācijas/pieteikšanās plūsmas uz sistēmu, kas ļaus veikt turpmākās darbības (Guliyeva, 2023).
2. Biznesa loģikas šabloni – pēc autorizācijas nākamais iet loģiski nozīmīgākā projekta daļa, kurā aprakstīta sistēmas galvenā ideja/pakalpojumi (Guliyeva, 2023).
3. Slēguma šabloni (angl. *Closure Patterns*) – šāda veida šabloni ietver pēdējos soļus kopējā pakalpojuma posmā (Guliyeva, 2023). Ja interneta veikala projekts tiktu ņemts kā piemērs, tad piegādes vai maksājumu šabloni būtu pēdējie pakalpojuma soļi.

Analizējot dažādu rīku lietošanas gadījumu šablonus, ir konstatēts, ka tika novēroti izplatīti datu pārvaldības lietošanas gadījumi, kas arī tika izmantoti lielākajai daļai analizējamo piemēru. Rīkos esošo lietošanas gadījumu piemēru apkopojums ļauj secināt, ka dažos lietošanas gadījumos tiek izmantoti: izveidot (angl. *Create*) un atjaunināt (angl. *Update*) lietošanas gadījumi, citi apskatīt (angl. *Show*) un rediģēt (angl. *Edit*), taču tika atrasti arī piemēri, kuros tika izmantotas visas 4 darbības objektiem – šos lietošanas gadījumus var saistīt ar tehniskām CRUD operācijām (Nikiforova, Babris, et al., 2024b). Tāpēc šo lietošanas gadījumu struktūru ir vērts izveidot kā šablonu, kas var atvieglot pielāgošanu noteiktam projektam (Nikiforova, Babris, et al., 2024b).

3.13. tabulā ir attēlots pārskats par biežāk izmantotajiem šabloniem (Guliyeva, 2023), kas dod pamatojumu atlasīt visbiežāk atkārtojušies šabloni, bet pārējos šablonus no tabulas izslēgt. 3.15. attēlā ir parādīts trīju līmeņu šablona piemērs, kas ir strukturēts, balstoties uz veikto lietošanas gadījumu klasifikāciju (Nikiforova, Babris, et al., 2024b).

Lietotāja saskarnes šablonu definēšanas procesā, katram šablonam ir jārod pamatinformācija, piemēram, problēmas izklāsts (kas), konteksts (kad), risinājums (kā) un pamatojums (kāpēc) (Suleri, Kipi et al., 2019; Spero & Biddle, 2020).

**3.13. tabula**

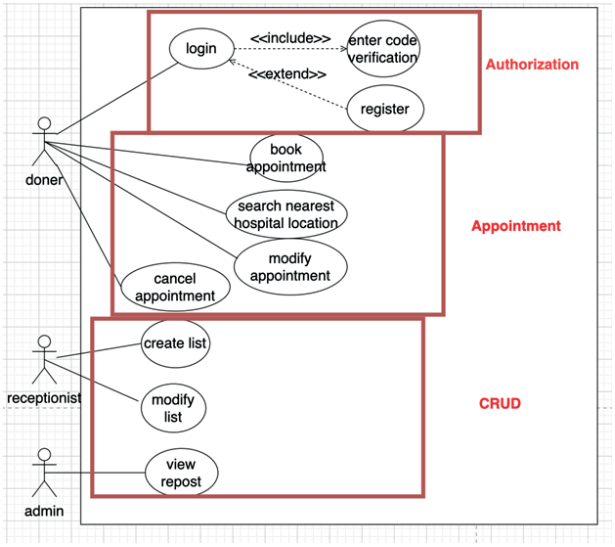
**Noteiktu šablonu pielietojumu skaits (aizgūts no (Guliyeva, 2023))**

| **Šabloni kas atkārtojas** | **Pielietojuma skaits** |
|---|---|
| Autorizācija | 45 |
| Pieraksts (angl. *Appointment*) | 15 |
| Pasūtījuma izveide | 20 |
| Banku process | 16 |
| Piegāde | 12 |
| Maksājuma process | 31 |
| Atgriezeniskā saite (angl. *Feedback*) | 13 |
| Datu pārvaldība (CRUD) | 30 |

Pētījumā par mobilo lietotņu lietotāja saskarnes šabloniem (da Silva, Paulo, 2022) ir izdalīti galvenie jautājumi uz kuriem būtu jāatbild lietotāja saskarnes šablona definīcijai:

1. Nosaukums – Nosaukuma elements ir saīsinājums, kas balstīts uz risinājumu, lai apzīmētu modeļa identifikāciju. Ieteicams, lai šis atribūts būtu īss un skaidrs, un, kad vien iespējams, tas nav pretrunā ar jau esošu konceptu (da Silva, Paulo, 2022; Welie, Veer et al., 2000; Molina, Meliá et al., 2002).
2. Problēma – problēmas elements norāda problēmu, kurai tiek meklēts risinājums, un ir atbildīgs par šķēršļa vai nekārtības īsu un tiešu parādīšanu (da Silva, Paulo, 2022; Welie, Veer et al., 2000; Molina, Meliá et al., 2002).

3. Risinājums – apraksta atbildi problēmai ar noteiktu lietotāja saskarnes šablonu. Risinājumam jāspēj parādīt, kā problēma tiek īstenota vai atrisināta, kā arī jāparāda, ko šablons ietver vai kas ir nepieciešams tā īstenošanai (da Silva, Paulo, 2022; Welie, Veer et al., 2000, Molina, Meliá et al., 2002).
4. Konteksts – attiecas uz apstākļiem, kādos šo modeli var vai nevar izmantot (da Silva, Paulo, 2022; Welie, Veer et al., 2000; Molina, Meliá et al., 2002)).
5. Saistītie šabloni – Norāda, kuri lietotāja saskarnes šabloni ir alternatīvi vai paplašināmi. Atsaucoties uz šiem šabloniem, kas saistīti ar obligātajiem vai izvēles šabloniem, aprakstīts attiecīgā šablona sastāvs (da Silva, Paulo, 2022; Welie, Veer et al., 2000; Molina, Meliá et al., 2002)).
6. Kategorija – ietver šablona klasifikāciju saskaņā ar dažiem kopīgiem kritērijiem vai aspektiem plašākā kopā (da Silva, Paulo, 2022; Welie, Veer et al., 2000; Molina, Meliá et al., 2002)).



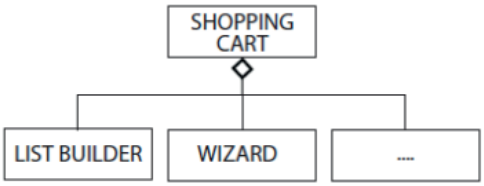

**3.15. att. Triju līmeņu šablona piemērs (aizgūts no (Nikiforova, Babris, et al., 2024b))**

Attiecības starp lietotāja saskarnes šabloniem ir svarīgas (Kruschitz & Hitz, 2010), lai pilnībā izmantotu atsevišķu lietotnes saskarnes šablonu atkārtotu izmantošanas potenciālu.

Lietotāja saskarnes šablonu attiecības parasti tiek aprakstītas ļoti īsi, tikai norādot savienojumus ar citiem šabloniem, kas var būt piemērojami konkrētai dizaina problēmai. Tomēr pareiza attiecību apsvēršana sola vēl jaudīgākas meklēšanas un navigācijas iespējas (Kruschitz & Hitz, 2010).

Lietotāja saskarnes šablons var būt vesels ekrāns, ekrāna daļa vai lietotāja saskarnes elements un tā apraksts. Lietotāja saskarnes šablonu datu kopa ir izveidota, manuāli marķējot lietotāja saskarnes ekrānus. Katrā lietotāja saskarnes šablona instancē ir ietverts lietotāja saskarnes elementu ekrānuzņēmums vai attēls. Tajā ir arī skatu hierarhijas koks, kurā tiek saglabāta lietotāja saskarnes elementu hierarhiskā struktūra. Katrs koka mezgls atbilst lietotāja saskarnes elementam, un tajā tiek saglabāta informācija par lietotāja saskarnes elementu, piemēram, elementa koordinātas ekrānos, tā atribūti, resursi, uz kuriem norāda elements, utt. Katram ekrāna lietotāja saskarnes elementam ir atbilstošs apraksts (Nguyen, Vu et al., 2018).

Iepriekš, programmatūras inženierijas pētnieki ir ierosinājuši iespējamās kategorizēšanas pieejas attiecībām OOP modeļu jomā. Turklāt, attiecības starp šabloniem var izmantot lielu atkārtotas izmantošanas repozitoriju pārlūkošanai (Hitz & Werthner, 1995). 3.16. attēlā (Kruschitz & Hitz, 2010) ir redzams piemērs iepirkuma groza lietotāja saskarnes šablonam, kurš sastāv no vairākiem cietiem lietotāja saskarnes šabloniem. Lai uzlabotu un paātrinātu atrašanas procesu, pārlūkošanas paradigmu var apvienot arī ar meklēšanas paradigmu. Vispirms lietotājs meklē konkrētu lietotāja saskarnes problēmu, un, pamatojoties uz vaicājuma rezultātiem, ir iespējams pārlūkot repozitoriju, lai noteiktu vislabāk atbilstošo risinājumu (Kruschitz & Hitz, 2010).




**3.16. att. Iepirkuma groza šablons sastāv no vairākiem citiem šabloniem (aizgūts no (Kruschitz & Hitz, 2010))**

Līdzās attiecībām, izmantojot ontoloģiju, var aprakstīt lietotāja saskarņu šablonu semantiku. Visā tīmekļa kopienā ir daudz lietotāju saskarņu šablonu, kas apraksta vienu un to pašu problēmu, bet ar atšķirīgu vārdu krājumu. Tāpēc ir grūti saprast un piekļūt šīm dizaina zināšanām

– ir nepieciešama ontoloģija vai formalizēta semantika, lai nodrošinātu kopīgu vārdu krājumu un mašīnapstrādājamu dizaina modeļu formu, ko var izmantot lietotāja saskarņu šablonu pārvaldības rīki (Kruschitz & Hitz, 2010).

Citā pētījumā par mobilo lietotņu testēšanu (Morgado & Paiva, 2015; Moreira, Paiva et al., 2013) tiek lietotāja saskarnes šabloni tiek definēti kā korteža kopa (angl. *Set of Tuples*): ***<G, V, A, C, P>*** , kas tiek aprakstīta 3.14. tabulā. Lietotāja saskarnes šablonu definēšana kā korežu piedāvā vairākas priekšrocības, jo īpaši struktūras, skaidrības un atkārtotas izmantošanas ziņā.

**3.14. tabula**

**Korteža kopas definīcija (adaptēts no (Morgado & Paiva, 2015))**

| **Elements** | **Apraksts** |
|---|---|
| **G** | Šablona mērķis vai ID |
| **V** | Ievades datu pāru kopa {mainīgais, vērtība}ar iesaistītajiem mainīgajiem lielumiem |
| **A** | Veicamo darbību secība |
| **C** | Veicamo pārbaužu kopums |
| **P** | Priekšnosacījums (būla izteiksme), kas definē nosacījumus, kādos šablons jāpiemēro |

Tādejādi, lai izpildītu šos šablonus ir nepieciešams norādīt (konfigurēt) katra mainīgā vērtību (V formātā). Tā ir iespējams izmantot vienu un to pašu šablonu vairākas reizes ar dažādiem konfigurācijas parametriem. Šablonu ir iespējams definēt kā (Morgado & Paiva, 2015):

$$G[configuration] : P- > A[V]- > C$$

kur, katra mērķa konfigurācijai G[konfigurācija], ja priekšnosacījums (P) tiek izpildīts, tiek izpildīta darbību secība (A), ar atbilstošajām ievades vērtībām (V). Noslēgumā tiek veikta pārbaudes kopa (C) (Morgado & Paiva, 2015).

3.15. tabulā tiek demonstrēta vienkāršota lietotāja saskarnes šablona definīcija, kurā neformālā veidā tiek aprakstīta definīcija.

**3.15. tabula**

**Iepirkuma groza lietotāja saskarnes šablona definīcijas piemērs**

| **Iepirkuma groza lietotāja saskarnes šablons** | |
|---|---|
| **Nosaukums** | Iepirkuma grozs |
| **Apraksts** | Apkopo informāciju par visiem klienta izvēlētajiem produktiem. Klients var pieprasīt preces savā grozā un izņemt preces no groza. Kad klients veic atlases darbību kāda veida precei, tā tiek pievienota grozam (Fernández, Liu et al., 2021). |
| **Problēma** | Nepieciešams parādīt izvēlētās preces, kuras tika pievienotas grozam, izmantojot darbību "Pievienot grozam" |
| **Saistītie šabloni** | Norēķins (angl. *Checkout*)<br>Preču katalogs (angl. *List Items*) |
| **Kategorija** | Interneta veikals |
| **Ievade** | Pievienot grozam (angl. *Add to cart*)<br>Skatīt grozu (angl. *Show cart*) |
| **Izvade** | Norēķins (angl. *Checkout*)<br>Preču katalogs (angl. *List Items*) |
| **Transformācijas** | Hierarhiskā veidā ar XML aprakstīts skatījuma saturs un izmantojamā lietotāja saskarnes satvara specifiskas koncepta transformācijas |

Lietotāja saskarnes šabloniem ir jābūt tādai pašai filozofijai kā arhitektūras vai programmatūras izveides šabloniem. Tomēr, precīzs šablona formāts ir atkarīgs no "tēmas", un tāpēc lietotāja saskarnes šabloniem arī ir nepieciešams specializēts formāts (Welie, Veer et al., 2000).

Citā lietotāja saskarnes šablona definīcijā (3.16. tabula) ir redzams, ka ievade var notikt tikai, kad ir aktivizēts norēķina notikums. Šādi ierobežojot lietotāja saskarnes šablonu ievadi un izvadi var panākt precīzāku lietotāja saskarnes šablonu izvēli un piemērotību.

**3.16. tabula**

**Norēķina lietotāja saskarnes šablona definīcijas piemērs**

| **Iepirkuma groza lietotāja saskarnes šablons** | |
|---|---|
| **Nosaukums** | Norēķins |
| **Apraksts** | Kad lietotājs izvēlas norēķināties, viņam tiek parādīts pasūtījuma preču galīgais saraksts, kā arī maksājuma iespējas. Lietotājs jebkurā laikā var izvēlēties turpināt iepirkšanos vai doties uz norēķināšanos – maksāt un pasūtīt to, kas ir iepirkumu grozā. (Fernández, Liu et al., 2021). |
| **Problēma** | Nepieciešams parādīt izvēlētās preces, kuras tika pievienotas grozam, izmantojot darbību "Pievienot grozam" |
| **Saistītie šabloni** | Pasūtījuma izveidošana<br>Preču katalogs (angl. *List Items*) |
| **Kategorija** | Interneta veikals |
| **Ievade** | Norēķins (angl. *Checkout*) |
| **Izvade** | Pasūtījuma izveidošana<br>Preču katalogs (angl. *List Items*) |
| **Transformācijas** | Hierarhiskā veidā ar XML aprakstīts skatījuma saturs un izmantojamā lietotāja saskarnes satvara specifiskas koncepta transformācijas |

### 3.4.2. Lietotāja saskarnes šablonu izvēles algoritms

Lai varētu piemērot lietotāja saskarnes šablonus izstrādājamā projekta procesiem, nepieciešams identificēt šos šablonus un kartēt ar procesiem. Šajā darbā autors apskata literatūrā sastopamos veidus kā pusautomātiski vai automātiski piemērot izstrādes šablonus. Tiek pieņemts, ka izstrādes šablonu definēšanā un izvēlē ir līdzība ar lietotāja saskarnes šablonu definēšanu.

Neskatoties uz to, ka ir definētas daudz un dažādas lietotāja saskarnes šabloni un pat bibliotēkas, vienā no pētījumiem (Wolff & Forbrig, 2010) tiek aprakstītas problēmas saistībā ar lietotāja saskarnes šablonu definēšanu, pielāgošanu un atrašanu:

1. Šajās kolekcijās nav konsekventu lietotāja saskarņu šablonu nosaukumi. Iespējams, ka būtībā vienādam šablonam tiks ievadīti vairāki ieraksti (Wolff & Forbrig, 2010).

2. Kataloga iekšējās atsauces ir reti sastopamas, starpkatalogu atsauces gandrīz neeksistē (Wolff & Forbrig, 2010).
3. Lietotāja saskarnes aspekti, kas ir detalizēti aprakstīti šablona ierakstā, atšķiras. Tas ir ne tikai nosaukumā, bet arī apjomā. (piemēram, problēmas vai risinājuma apraksts ir izlaists) (Wolff & Forbrig, 2010).
4. Abstrakcijas līmenis atšķiras atkarībā no lietotāja saskarnes šablona (Wolff & Forbrig, 2010).
5. Lietotāja saskarnes risinājums dots ļoti neformāls, pārsvarā teksts, kam pievienoti attēli ilustrācijas nolūkos (Wolff & Forbrig, 2010).

Neskatoties uz augstākminētajām problēmām, ir mēģināts atrast pieejas lietotāja saskarņu šablonu katalogu standartizēšanai. Piemēram, XML dialekts PLML (Fincher, Finlay et al., 2003) ir šāds mēģinājums. Tas tika izstrādāts, lai noteiktu kopīgu bāzi, kā aprakstīt izstrādes, lietotāja saskarnes u.c. šablonus. PLML definē šablona valodu un apmaiņas formātu. Tas tika izveidots, lai aptvertu vispārīgus šablonus, nevis tikai lietotāja saskarnes šablonus. Bez konkrētas problēmvides kā robežas nebija iespējams ierobežot valodas elementus ārpus bāzes ierobežojumiem. Līdz ar to PLML pati nepārvar iepriekš minēto formalizācijas problēmu (Wolff & Forbrig, 2010).

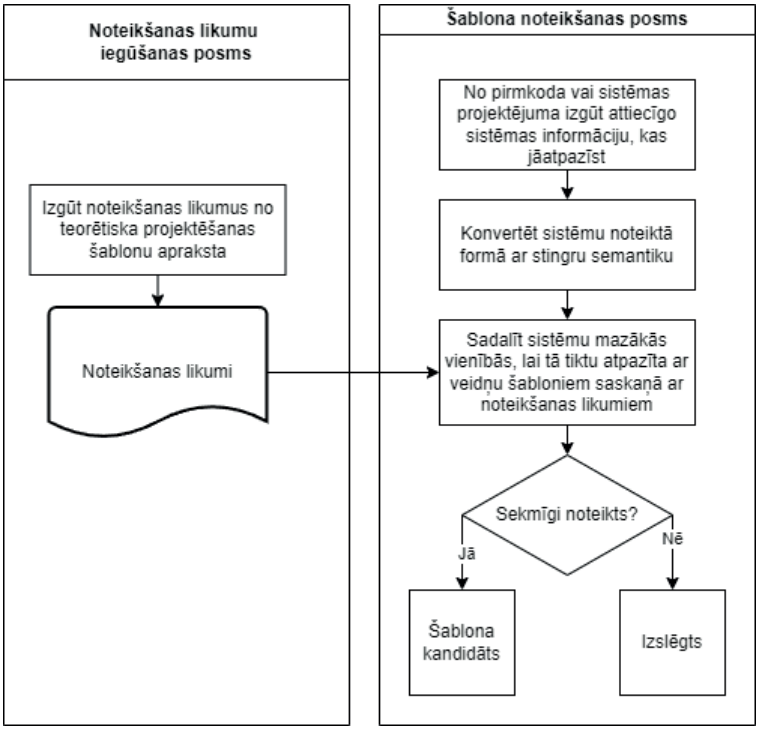
Literatūrā var izdalīt vismaz 3 veidus, kā izvēlēties izstrādes šablonus un tādejādi šīs metodes var pārnest uz lietotāja saskarnes šabloniem.

#### *3.4.2.1. UML sakņota pieeja*

Vienotajā modelēšanas valodā (angl. *Unified Modelling Language* – UML) sakņotā pieejā, piemēram (Kim & Khawand, 2007) un (Kim & Wuwei, 2008) klasificēja problēmu definīcijas avotus divās daļās: dizaina modeļa problēmas definīcija un zināmās dizaina modeļa problēmās (Hasheminejad & Jalili, 2012). Pēc tam, pamatojoties uz iepriekšminētajiem avotiem, viņi ir izmantojuši abstraktu meta modeli, lai formalizētu dizaina šablonus. Meta modelis ir iedvesmots no programmatūras dokumenta struktūras (klases diagramma) un uzvedības (sadarbības diagramma). 3.17. attēlā ir redzams izstrādes šablonu noteikšanas pamatplūsma (Wang, Song et al., 2022).

Šai pieejai ir vairāki ierobežojumi. Pirmkārt, tas ir aprobežojies ar precīzu visu šablonu problēmas definīcijas precizēšanu, jo izveidotais metamodelis dažiem izstrādes šabloniem var būt

vienāds (Hsueh, Kuo et al., 2009). Šīs grūtības iemesls ir tas, ka šo izstrādes šablonu problēmas definīcijas struktūra ir vienāda, piemēram, no UML diagrammu viedokļa, taču to mērķi ir atšķirīgi (Hasheminejad & Jalili, 2012). Otrkārt, šī pieeja nav mērogojama, jo palielinās metamodeļu līdzības lielam skaitam izstrādes šabloniem. Treškārt, katra dizaina modeļa metamodeļa izveides izmaksas ir pārāk augstas. Visbeidzot, katra dizaina modeļa metamodeļa izveides posms nav automātisks (Hasheminejad & Jalili, 2012).




**3.17. att. Izstrādes šablonu noteikšanas pamatplūsma (aizgūts no (Wang, Song et al., 2022))**

Savukārt, mašīnmācīšanās pieeja (Wang, Song et al., 2022), kura, piemēram, no UML spētu automātiski izgūt lietotāja saskarnes šablonus, balstās uz sistēmas sadalīšanu apakšsistēmās un otrā līmeņa apakšsistēmās. Apakšsistēmu un otrā līmeņa apakšsistēmu sadalīšana aizņem daudz laika un vietas atmiņā. Nepieciešamais laiks un atmiņas vieta palielinās kvadrātiski, palielinoties sistēmas klašu skaitam.

#### *3.4.2.2. Uz ontoloģiju balstīta pieeja*

Uz ontoloģiju balstīta pieejai (Hasso & Carlson, 2005) ir ierosinājuši izstrādes šablonu klasifikāciju, pamatojoties uz izstrādes šablonu problēmas definīcijas nozīmi un tās teikumu semantisko analīzi. Šajā darbā klasifikācija veikta, balstoties uz darbības vārdu nozīmi izstrādes šablonu problēmas definēšanā, izmanto lingvistiskās teorijas teikuma struktūras semantiskajā analīzē. Autori apgalvoja, ka pareizo izstrādes šablonu var noteikt, izmantojot izstrādes šablona semantisko klasifikāciju, taču viņi nav iesnieguši savas metodes novērtējumu, un tiek atzīmēts, ka klasifikācija nav automātiska (Hasheminejad & Jalili, 2012). Khoury (Khoury, Mokhtari et al., 2008) metodes pamatā ir ontoloģijas tīmekļa valoda (OWL) kas izmanto ontoloģisko saskarni programmatūras izstrādātājiem, lai izvēlētos, piemēram, Piemēram, interneta veikala iepirkuma grozu (Kruschitz & Hitz, 2010).

Šim darbam ir divi trūkumi. Pirmkārt, ir nepieciešams formalizēt drošības koncepciju specifikācijas un drošības prasības, piemēram, konfidencialitāte. Tomēr visās programmatūras projektēšanas jomās nav formālu aprakstu, un tas rada ierobežojumus šī darba pielietošanai praksē (Hasheminejad & Jalili, 2012). Citā pētījumā (Blomqvist, 2008) ierosināja metodi automātiskai pareizo izstrādes šablonu atlasei, tā izmanto izstrādes šablonu ontoloģiju, lai sakārtotu pareizos izstrādes šablonus. Kā arī šajā pieejā izmantoja dažādus kritērijus izstrādes ranžēšanai, ietver klases atbilstības, centralitāti, blīvumu un semantisko līdzību (Hasheminejad & Jalili, 2012).

#### *3.4.2.3. Teksta klasifikācijas metode*

Teksta klasifikācijas metode – piedāvātajā metodē (Hasheminejad & Jalili, 2012) tiek izmantota trešā pieeja, t.i., izstrādes šablonu problēmas definīcijas teksta klasifikācija, lai palielinātu pareizās šablonu izvēles efektivitāti un automatizāciju. Piedāvātajai metodei ir priekšrocības salīdzinājumā ar līdzīgiem darbiem. Pirmkārt, nav nepieciešamas formālas dizaina modeļa problēmas definīcijas specifikācijas. Otrkārt, tas ir ne tikai piemērojams jebkura veida dizaina modelim, bet arī to var izmantot, izvēloties bibliotēkas rutīnas. Treškārt, tas var ieteikt dažādus dizaina modeļus atkarībā no līdzības pakāpes starp izgūtā dizaina modeļa problēmas definīciju un dizaina problēmu. Ceturtkārt, piedāvātā metode piedāvā sistemātisku veidu, kā novērtēt un uzlabot pētnieku veiktās manuālās klasifikācijas neatbilstības. Piektkārt, tai ir vienkārša

modeļu atlases procesa automatizācija un zemas izmaksas salīdzinājumā ar citām pieejām. Visbeidzot, tas ir piemērojams daudziem dizaina šabloniem un tas ir mērogojams.

Šajā darbā tiek izvēlēta teksta klasifikācijas pieeja kā atlases metode lietotāja saskarnes šabloniem ar zemu izmaksu priekšapstrādei, tādējādi tā ir ātrāka, vienkāršāka un lētāka nekā citas pieejas. Piedāvātā metode tika novērtēta, izmantojot trīs dizaina modeļu grupas un pietiekami daudz reālu dizaina problēmu aprakstu (Hasheminejad & Jalili, 2012).

Izvēlētajai metodei ir četras fāzes: pirmapstrāde (angl. *Preprocessing*), dizaina modeļu klasifikatoru apguve (angl. *Learning Design Patterns Classifiers*), izstrādes šablonu klases noteikšana (angl. *Determination of a Design Patterns Class*) un dizaina modeļa (angl. *Suggestion of Design Pattern*) ierosināšana (Hasheminejad & Jalili, 2012).

#### *3.4.2.4. Uz lēmumiem balstīta metode*

Lai efektīvāk veiktu modeļa atribūtu definēšanu, transformācijas likumu veikšanu un galu galā iegūtu funkcionālu lietotāja saskarni ir nepieciešams standartizēt nosaukumus, jo nosaukums var nest semantiskas zināšanas par procesu – piemēram, izveidot, dzēst vai atjaunināt. Vienā no publicētām pieejām (Tena, Dez et al., 2013) autori demonstrē atkārtoti lietojamus dizaina ģeneratīvos šablonus. Ar šādu pieeju var efektīvi izveidot lietotāja saskarnes, kurās vienlaikus saglabāsies konsekvence, saskaņotība un mērogojamība – svarīgi elementi izstrādes procesā.

Semantiski šablonu, par kuriem iestājas autori (Tena, Dez et al., 2013), atvieglo dažādu procesu definēšanu, kā arī to pārvaldību – nodrošina pielāgošanos arī specifiskiem gadījumiem. Šabloni veicina lietotāja saskarnes izstrādes procesa efektivitāti. Lietotāja saskarnes izstrādes vai automātiskas ģenerēšanas procesā pielietoti šādi šabloni palīdz racionalizēt procesu un uzlabot lietojamību.

Pētījumā autori ir izveidojuši lēmumu koku, kurā katrā no lietotāju saskarnes detalizācijas līmeņiem tiek piedāvāta izvēle, kādu lietotāja saskarnes elementu izvēlēties pamatojoties uz attiecīgo elementu funkcionalitātes diskursu.

### 3.4.3. Lietotāja saskarnes šablonu bibliotēka

Lietotāja saskarnes šablonu bibliotēka ir kopa, kura sastāv no lietotāja saskarnes šabloniem, kurus dizaineri, izstrādātāji un informācijas arhitekti izmanto, lai izveidotu tīmekļa lietotnes

(Nikiforova, Zabiniako et al., 2021b). Šajā kolekcijā parasti ir iekļauta bieži atkārtota arhitektūra, dizaina, pirmkoda, kā arī SEO (meklētājprogrammu optimizācijas) elementi, kurus var konsekventi izmantot visā tīmekļa lietotnē. Šāda bibliotēka ir jāizveido projekta ietvaros vai kā atsevišķa, projekta neitrāla kopa.

Vienā no pētījumiem (Nguyen, Vu et al., 2018) autoru ideja ir izstrādāt un izvietot progresīvus dziļās mācīšanās modeļus, kuru pamatā ir atkārtoti neironu tīkli (angl. *Recurrent Neural Networks, RNN*) un ģeneratīvie pretrunīgie tīkli (angl. *Generative Adversarial Networks, GAN*), lai apgūtu lietotāja saskarnes šablonus no miljoniem pašlaik pieejamo mobilo lietotņu. Kad šie modeļi ir apmācīti, tos var izmantot, lai meklētu lietotāja saskarnes šablonu paraugus, ņemot vērā lietotāja sniegtos aprakstus, kas rakstīti dabiskā valodā (3.18. att.), un ģenerētu profesionāla izskata lietotāja saskarnes dizainus no vienkāršākiem, mazāk elegantiem dizaina uzmetumiem (Nguyen, Vu et al., 2018).


**3.18. att. Ieteiktais lietotāja saskarnes dizains no apraksta (aizgūts no (Nguyen, Vu et al., 2018))**

Kā piemērs tiek minēts gadījums, kad lietotnes izstrādātājs raksta vajadzīgā ekrāna aprakstu dabiskā valodā, piemēram, angļu valodā (Nikiforova, Zabiniako et al., 2021a). Pēc tam viņš izmanto šo aprakstu kā vaicājumu, lai meklētu *DeepUI* (Nguyen, Vu et al., 2018) miljoniem lietotāja saskarnes dizaina paraugu un veidņu krātuvē (bibliotēkā). *DeepUI* atrod atbilstošus eksemplārus un parāda dizaina paraugus, kuru apraksti ir līdzīgi izstrādātāja aprakstam. Sistēmai var uzdot tālāk filtrēt un ranžēt meklēšanas rezultātus, izmantojot citus kritērijus, piemēram, lietotņu vērtējumus vai kategorijas (Nguyen, Vu et al., 2018). Autori pētījumā (Nguyen, Vu et al.,

2018) izvēlējās izpētīt pieteikšanās ekrānus – ļoti populāru ekrānu, ko ietver lielākā daļa lietotņu. Viņi apkopoja pieteikšanās ekrānus 40 populārākajām *Android* lietotnēm no 4 kategorijām: saziņa, izglītība, iepirkšanās un sociālie tīkli, kas atrodas *Google Play Store* (Nguyen, Vu et al., 2018). Tika secināts, ka pieteikšanās ekrāniem, galvenokārt, ir trīs šabloni, proti: pieteikšanās ar tālruņa numuru, pieteikšanās ar lietotājvārdu un paroli un pieteikšanās tikai ar e-pastu (Nguyen, Vu et al., 2018).

Visos trīs ekrānos ir attēla skats, lai ekrāna augšdaļā parādītu lietotnes logotipu. Tālāk ir divi tekstlodziņi lietotājvārda un paroles ievadīšanai. Abos tekstlodziņos ir vietturi (angl. *Placeholders*), lai palīdzētu lietotājiem ievadīt pareizu informāciju. Šai shēmai ir arī noklikšķināma teksta etiķete "Aizmirsu paroli" un pierakstīšanās/pieteikšanās poga. Šis dizains ietver arī pierakstīšanās pogu(-as), izmantojot informāciju no citām lietotnēm, piemēram, *Facebook* un *Google*. Ekrānu apakšdaļā ir noklikšķināma teksta etiķete vai poga, lai reģistrētos, ja lietotāji ir jauni lietotnē un viņiem iepriekš nav kontu (Nguyen, Vu et al., 2018).

Vienā no pētījumiem (Zhao, Gao et al., 2012) autori aplūko problēmas, kas saistītas ar konsekventu un lietojamu lietotāja saskarnes uzturēšanu lielās programmatūras sistēmās. Tajā tiek piedāvāts risinājums, kas izmanto atkārtoti lietojamus lietotāja saskarnes šablonus un prezentācijas slāņa sistēmu (Zhao, Gao et al., 2012). Apvienojot lietotāja saskarnes šablonus, prezentācijas slāņa ietvaru (vienkāršo lietotāja saskarnes izstrādi, izmantojot iepriekš izveidotus komponentus un uz XML balstītu apraksta valodu) un uz lietotāju orientētus dizaina principus, autori panāk uzlabotu dizaina efektivitāti, atkārtoti izmantojot lietotāja saskarnes šablonus un komponentu, uzlabotu lietotāja saskarnes konsekvenci, izmantojot standartizētu pieeju un potenciālu labākai lietojamībai, izmantojot pārbaudītus dizaina šablonus (Zhao, Gao et al., 2012). Faktiski autori definē XML datnes, kurās tiek uzglabāta informācija par lietotāja saskarnes šabloniem (3.17. tabula).

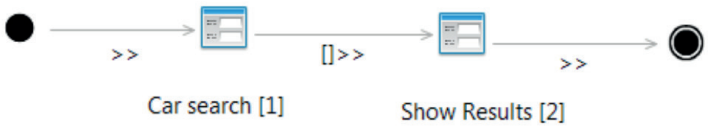
**3.17. tabula**

**Uzdevumu un lietotāja saskarnes šablonu kartēšanas tabula (adaptēts no (Zhao, Gao et al., 2012))**

| ID | Uzdevums | Lietotāja saskarnes šablons |
|---|---|---|
| 1 | Pārvaldīt informāciju par darbiniekiem, izmantojot organizācijas struktūras koku | Objekta pārvaldes šablons |

| ID | Uzdevums | Lietotāja saskarnes šablons |
|---|---|---|
| 2 | Atrast darbiniekus pašreizējā organizācijā | Objekta meklēšanas šablons |
| 3 | Ja personāls tiek atrasts, tad jātransformē par pārvaldīšanas šablonu | Objektu saraksta šablons |

Citā pētījumā (Moreira, Paiva et al., 2013) tiek testēti lietotāja saskarnes šabloni un definētas prasības *PARADIGM* valodā (3.19. att.).




**3.19. att. Meklēšanas un rezultātu definēšana (aizgūts no (Moreira, Paiva et al., 2013))**

Procesa beigās ir nepieciešams pielietot lietotāja saskarnes šablonu bibliotēku, lai izvēlētos katram procesam atbilstošu lietotāja saskarnes šablonu. Lietotāja saskarnes šabloni ir nepieciešami vairāku iemeslu dēļ. Galvenais iemesls ir nepieciešamība attēlot noteiktu procesu noteiktā veidā. Piemēram, interneta veikala preču grozs un tā funkcionalitāte ir aprakstīta daudzos avotos, lietotājam pierasta un efektīva. Nākošais iemels ir nepieciešamība projekta ietvaros saglabāt izskata un funkcionalitātes konsekvenci. Lietotāja saskarnes šabloni sniedz atkārtoti lietojamus risinājumus daudzām dizaina problēmām. Kad šabloni tiek iekļauti automātiskā lietotāja saskarnes izveidē, dizaineri var apstiprināt, ka saskarnes nodrošina vienveidību. Tas palīdz saglabāt pazīstamu izkārtojumu, mijiedarbību ar sistēmu un vizuālā plāna vienveidību. Lietotāja saskarnes šabloni apkopo labākās metodes un veiksmīgus, izplatītus dizaina risinājumus vienuviet. Tas ļauj automātiskās ģenerēšanas rīkiem izmantot gatavus modeļus, nevis izgudrot jaunus katru reizi, kad tie ģenerē lietotāja saskarni. Tas paātrina projektēšanas procesu un samazina manuālas iejaukšanās nepieciešamību. Piemēram, automātiskajiem lietotāja saskarnes ģenerēšanas rīkiem var būt parametri vai izvēles, kas ļauj mainīt šablonus dažādiem dizaina mērķiem vai lietotāju vēlmēm.

Tehnoloģijām attīstoties lietotāja saskarnes kļūst sarežģītākas un daudzveidīgākas, tādēļ lietotāja saskarnes šablonu izmantošana ir ļoti svarīga, lai risinātu dizaina sarežģītību un

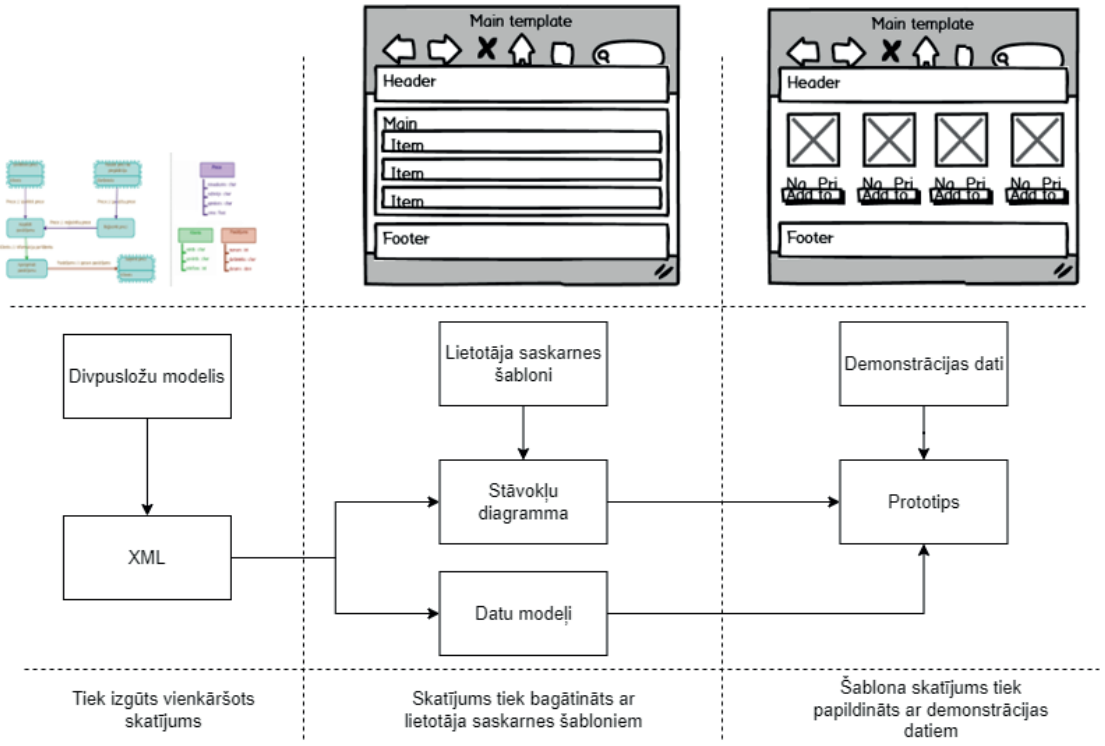
mērogojamību. Rīki, kas automātiski ģenerē lietotāja saskarnes, var izmantot šablonus, lai efektīvi izveidotu saskarnes dažādām platformām, ekrāna izmēriem un ierīcēm.

Lietotāja saskarnes šabloni darbojas kā kopīga valoda dizaineriem, izstrādātājiem un citām lietotāja saskarnes izstrādes procesā iesaistītajām pusēm. Izmantojot automātiskos lietotāja saskarnes ģenerēšanas rīkus, kas ietvertu lietotāja saskarnes šablonus, kļūst vieglāk sadarboties un saskaņot dizaina mērķus starp dažādām komandām un lomām.

Lietotāja saskarnes šabloni ir svarīgi automātiskai lietotāja saskarnes ģenerēšanai, jo tie nodrošina pārbaudītu dizaina risinājumu bāzi, kas uzlabo vienveidību, efektivitāti, lietojamību, elastību, mērogojamību, pieejamību un sadarbību starp iesaistītajām personām projektēšanas procesā.

## 3.5. Prezentācijas papildināšana ar lietotāja saskarnes šabloniem

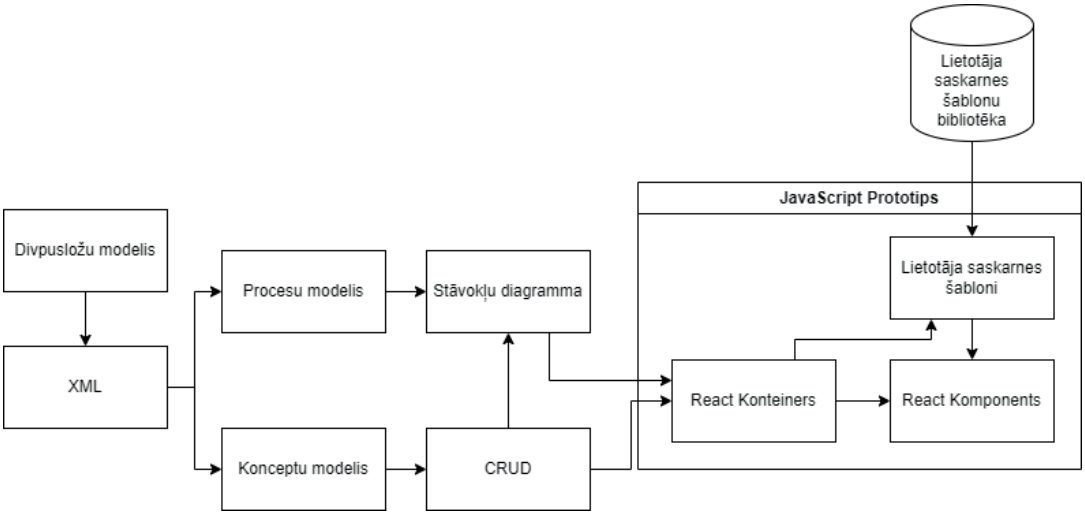
Tāpat kā programmatūras šablonu izmantošana koda ģenerēšanā uzlabo ģenerētā koda kvalitāti (Sunitha & Samuel, 2019), tā arī izmantojot lietotāja saskarnes šablonus tiek uzlabota lietojamības kvalitāti, ar cilvēkiem pierastiem maršrutiem un prezentāciju. 3.20. attēlā ir redzama kopējā shēma lietotāja saskarnes šablonu pielietošanai transformācijas procesā.




**3.20. att. Divpusložu modeļa transformācijas uz lietotāja saskarnes šabloniem kopējā shēma.**

Transformācijas procesā katram procesam tiek noteikts lietotāja saskarnes šablons izmantojot procesa nosaukumu. Piemēram, process "*Show Cart*", tiks marķēts ar "*Shopping cart*" šablonu, izmantojot "*Shopping cart*" šablona aprakstu, kas tiek apstrādāts izmantojot teksta klasifikācijas metodi. Pielietojot šablonu, tiek kartēts konceptu modelis ar šablonā iesaistītajām vērtībām, tas nozīmē, ka, piemēram, procesa "*Show Cart*" pielietotā šablona vērtības tiek saistītas ar iegūstamo datu modeli "*Item*". Kad ir izgūts lietotāja saskarnes šablons, tad ir nepieciešams aizvietot ģenerētajā JavaScript veidnē esošo saturu ar lietotāja saskarnes šablona koda saturu. Šis saturs tiek manuāli izstrādāts projektam, bet turpmākajos pētījumos, iespējams, varētu šo procesu automatizēt un izmantot tikai norādi – kādu lietotāja saskarnes bibliotēku vai satvaru izmantot. Prezentācijas plānojuma uzdevums ir iepriekšējos soļos izgūto navigācijas objektu prezentācijas definēšana, kura izpaužas kā statiskais un dinamiskais prezentācijas modelis (Koch & Mandel,

1999). Statiskais prezentācijas modelis katram navigācijas objektam piesaista vismaz vienu prezentācijas objektu, dinamiskais prezentācijas modelis apraksta šo prezentācijas objektu uzvedību (Koch & Mandel, 1999). Lietotāja saskarnes ģenerēšanas procesā (3.21. att.), pielietojot noteiktos šablonus var iegūt arī CRUD darbības (Nasiri, Rhazali et al., 2023), tādējādi katrs process un koncepts tiek attiecīgi pārveidots par lietotāja saskarnes elementu (vai elementu kopu) (Antović, Vlajić et al., 2012).



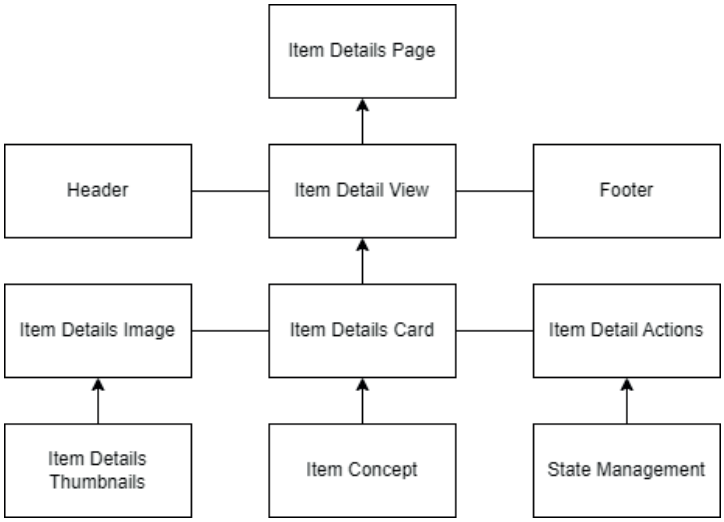

**3.21. att. Prezentācijas funkcionalitātes shēma**

Lai nodrošinātu iespēju iegūt lietotāja saskarnes elementus no divpusložu modeļa, tad esošie procesi ir jāpapildina ar metadatiem, kuros jāintegrē lietotāja saskarnes šabloni (Stahl, Voelter et al., 2006; Batdalov & Nikiforova, 2021), jāveic lietotāja saskarnes elementu kartēšanu ar procesiem un kontekstiem. Tālāk, atbilstoši transformācijas likumiem jāņem vērā atbilstošais šablons un citas iezīmes.

### 3.5.1. Eksperimenti ar lietotāja saskarnes elementu izgūšanu no divpusložu modeļa

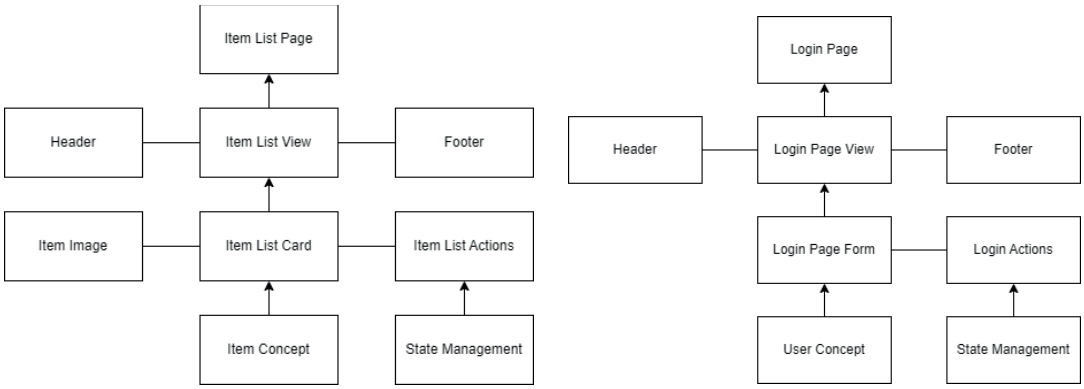
Autora piedāvātajā pieejā tiek izveidoti šabloni no citiem šabloniem tos hierarhiski apvienojot lapās. Katra lapa ir autonoma šablonu saturoša forma, kura sastāv no šabloniem koku struktūrā. Hierarhiski apvienojot šablonus lapās var sasniegt augsta līmeņa lapu struktūru. Piemēram, 3.22. attēlā, produkta kartiņa, sastāv no galvenes (angl. *Header*), produkta detalizētā

skatījuma un kājenes (angl. *Footer*). Savukārt produkta detalizētais skatījums sastāv no šim skatījumam paredzētā attēla šablona, kas sevī iekļauj arī produkta detalizētai kartiņai paredzēto sīktēlu šablonu. Produkta detalizētā skatījuma kartiņa iekļauj sevī daļu no informācijas par produkta konceptu, kurā atrodas tādi atribūti, kā produkta nosaukums, produkta apraksts, cena, utt. Lai šis skatījums būtu dinamisks, nepieciešams arī iekļaut stāvokļu pārvaldību, faktiski – uz kurieni var nokļūt no šī skatījuma tālāk.




**3.22. att. Produktu kartiņas šablona hierarhiska struktūra.**

Produktu kataloga lapa satur uzņēmuma piedāvāto produktu sarakstu. Šī skatījuma galvenā ideja ir parādīt kataloga skatījumā produktus tā, lai lietotājiem būtu viegli pārvietoties un viegli izpētīt tos. Produktu kataloga skatījumā (3.23. att. pa kreisi) ir vairākas daļas, piemēram, nelieli produktu attēli, ko sauc par sīktēliem, nosaukumi, īsi apraksti, informācija par cenām un skatījumā ir arī iespēja pievienot produktu preču katalogam, neizejot no kataloga lapas. Turklāt, pēc nepieciešamības, produktu kataloga šablons var būt saistīts ar citiem lietotāja saskarnes šabloniem, piemēram, meklēšanas, lappušu vai ieteikumu sistēmām. Savukārt, pēc struktūras vienkāršāka – pieteikšanās lapa (3.23. att. pa labi) sastāv no tās pašas galvenes, kājenes un pieteikšanās skatījuma, kurš sevī iekļauj pieteikšanās lapas formu, kurā tiek attēloti lauki no produkta koncepta, kā arī darbības šajā formā tiek iekļautas atkarībā no stāvokļa pārvaldības.




**3.23. att. Produktu kataloga šablona (pa kreisi) un pieteikšanās lapas (pa labi) hierarhiska struktūra**

### 3.5.2. Transformācijas likumi lietotāja saskarnes prezentācijas elementu ģenerēšanai

Nepieciešams izveidot transformācijas likumus, lai saskaņotu procesu nosaukumus ar šabloniem no lietotāja saskarnes šablonu bibliotēkas. Transformācijas likumu mērķis ir automātiski ieteikt lietotāja saskarnes šablonus no šablonu bibliotēkas, paļaujoties uz procesu nosaukumiem, kas norāda gan uz šablona nosaukumu, gan tā funkciju.

Pirmkārt, ir nepieciešams identificēt atslēgvārdus (Haz, Funabiki et al., 2024). Divpusložu modeļa procesa nosaukums tiek izmantots, lai izgūtu lietotāja nodomu. Piemēram, procesa nosaukums "*Show Cart*", satur sevī gan "*Show*", gan "*Cart*", no kā var secināt, ka šablons ir saistīts ar preču grozu un attēlošanas funkciju.

Otrkārt, identificētie atslēgvārdi ir jākartē ar lietotāja saskarnes šabloniem. Identificētie divpusložu modeļa procesa nosaukumu atslēgvārdi tiek kartēti ar lietotāja saskarnes šablonu bibliotēku šabloniem. Šī bibliotēka asociē atslēgvārdus vai funkcionalitāti ar noteiktiem lietotāja saskarnes šabloniem. Piemēram, "*Create*" var tikt asociēts ar formas šablonu, bet "Item" ar specifisku datu ievades formu apakššablonu lietotāja saskarnes šablonu bibliotēkā.

Treškārt, pēc izvēles, var ņemt vērā arī kontekstu jeb koncepta elementu nosaukumus, lai precīzāk kartētu lietotāja saskarnes šablonus. Piemēram, "*Search Items*" var piedāvāt "*Search Bar*" šablonu, bet ja process sevī satur filtrēšanu pēc noteiktiem kritērijiem, tad precīzāks "*Filter Panel*"

šablons iederētos labāk. Šo transformēšanas likumu efektivitāte lielā mērā ir atkarīga gan no procesa nosaukumu, gan lietotāja saskarnes šablonu bibliotēkas visaptverošuma un skaidrības.

Lai formāli strukturētu nepieciešamos veicamos soļus, tiek piedāvāts lietotāja saskarnes šablonu procesu definēt kā kortežas kopu, kas tiek aprakstīta 3.18. tabulā.

**3.18. tabula**

**Korteža kopas definīcija (adaptēts no (Morgado & Paiva, 2015))**

| **Elements** | **Apraksts** |
|---|---|
| **P** | Divpusložu modeļa procesa nosaukums |
| **K** | Atslēgvārdi izgūti no P |
| **PL** | Lietotāja saskarnes šablonu bibliotēka |
| **M** | Izvēlētais lietotāja saskarnes šablons no PL |

Tālāk ir sniegts transformācijas likumu sadalījums ar piemēriem, lai kartētu procesu nosaukumus ar lietotāja saskarnes šabloniem no iepriekš definētas bibliotēkas. Faktiski var būt vairākas situācijas, kurās nepieciešams atrast precīzāko šablonu. Šablonu bibliotēkai jābūt precīzi definētai ar skaidriem aprakstiem un piemēriem katram šablonam, jo izmantojot teksta klasifikācijas metodi, tiks izvēlēti šabloni, balstoties uz atslēgvārdiem (Hasan, Sanyal et al., 2017). Savukārt, transformācijas likumus var pielāgot, atkarībā no lietojumprogrammas problēmvides un funkcijām. Kā arī, tālākajos pētījumos var izskatīt mašīnmācīšanās paņēmienus, lai uzlabotu šablonu atlases precizitāti, analizējot iepriekšējos kartējumu un lietotāju mijiedarbību.

Pagaidām tiek uzskaitīti trīs gadījumi – atslēgvārdu identifikācija un atbilstības noteikšana (3.19. tabula), precizēšana ar kontekstu (3.21. tabula), Nenoteiktības (angl. *Ambiguity*) apstrāde (3.23. tabula).

Atslēgvārdu identifikācija un atbilstības noteikšana (3.19. tabula) ir galvenā ideja, lai procesu nosaukumus pārvērstu lietotāja saskarnes šablonos. Izmantojot atslēgvārdu atbilstību, var kartēt atslēgvārdus no procesa nosaukuma ar iepriekš definētiem šabloniem bibliotēkā. Ja, pamatojoties uz atslēgvārdiem, ir vairākas iespējamās atbilstības, lietotāju saskarnes šablonu bibliotēka var piedāvāt šablonu prioritātes līmeņus. Augstākas prioritātes modeļi parasti ir specifiskāki un tiem ir prioritāte. Šāda pieeja palīdz piedāvāt primāro lietotāja saskarnes šablonu,

kas atbilst procesa funkcionalitātei. Piemērā "*Create Customer Account*" kā sākotnējā opcija tiek ieteikta "*User Registration Form*". Efektivitāte lielā mērā ir atkarīga gan no procesu nosaukumu, gan atslēgvārdu precizitātes un detalizācijas, kas saistīti ar lietotāja saskarnes šabloniem, lietotāja saskarnes šablonu bibliotēkā.

**3.19. tabula**

**Atslēgvārdu identifikācija un atbilstības noteikšana**

| | |
|---|---|
| **Procesa nosaukums** | Create Customer Account |
| **Atslēgvārdi** | "Create", "Customer Account" |
| **Saderīgie šabloni** | "Create" kartējas ar "Form" šablonu bibliotēkā<br>"Customer Account" var tikt kartēts ar "User Registration Form" apakššablonu |

Atslēgvārdu identifikācija un atbilstības noteikšanas gadījumā (skatīt 3.20. tabulu), ievade ir P un izvade ir M, lai to panāktu, nepieciešams veikt šādus soļus:

1. Izgūt atslēgvārdus K no P, izmantojot iepriekš definētu atslēgvārdu ekstrakcijas funkciju (angl. *Keyword Extraction Function, KEF*) (Haz, Funabiki et al., 2024).
2. Par katru atslēgvārdu k no K atrast visus šablonus M no PL, kur k ir iekļauts atslēgvārdos, kas saistīti ar M.
3. Ja 2. punkta darbībā sakrīt vairāki šabloni M, atgrieziet šablonu ar augstāko prioritāti, kas definēta PL.
4. Ja 2. darbībā neviens paraugs neatbilst, tad atgriezt "Nav atrasta atbilstība".

**3.20. tabula**

**Formāls piemērs atslēgvārdu identifikācijai un atbilstības noteikšanai**

| | |
|---|---|
| **P** | Create Customer Account |
| **K** | {"Create", "Customer Account"} |
| **PL** | {"Form": (keywords: {"Create", "Edit"}), "User Registration Form": (keywords: {"Create", "User Account"})} |
| **M** | "User Registration Form" (pamatojoties uz atslēgvārdu atbilstību un, iespējams, augstāku prioritāti noteiktām formām) |

Atslēgvārdu identifikācija un atbilstības noteikšana ir primārais veids, kā noteikt lietotāja saskarnes šablonus, izmantojot procesu nosaukumus. Taču dažos gadījumos varētu būt noderīgi iegūt sīkāku informāciju par procesa definīciju, lai labāk atlasītu un kartētu lietotāja saskarnes šablonu. Tas nozīmē, ka jāaplūko papildu informācija, kuras konteksts var būt dažādos formātos:

1. Procesu apraksti – procesa nosaukums ir viena lieta, taču dokumentācija vai apraksti, kas tam pievienoti (divpusložu modeļa gadījumā tie ir <*comments*> un <*description*> procesa modeļa atribūti, var sniegt papildus informāciju. Piemēram, ja ir apraksts "*Search Products*" (3.21. tabula), tas varētu sniegt informāciju, ka jāmeklē lietotāja saskarnes šablons ar iespēju meklēt preces pēc noteiktiem kritērijiem.
2. Datu modelis – process ir saistīts ar noteiktiem konceptiem divpusložu modelī. Zināšanu iegūšana par konceptu, jo īpaši par to, kāda veida dati tiek meklēti vai filtrēti, var ietekmēt lietotāja saskares šablona izvēli.

**3.21. tabula**

**Precizēšana ar kontekstu**

| | |
|---|---|
| **Procesa nosaukums** | Search and Filter Products |
| **Atslēgvārdi** | "Search", "Filter", "Product" |
| **Saderīgie šabloni** | "Search" sākotnēji tiek kartēts ar "Search Bar" šablonu<br>Ņemot vērā "Filter" un produkta kontekstu, var precizēt atlasi līdz "Advanced Search" vai "Filter Panel" šablonam, kas ļauj filtrēt pēc papildu kritērijiem. |

Atslēgvārdu precizēšnai ar kontekstu (skatīt 3.22. tabulu), ievade ir P un C, bet izvade ir M, lai to panāktu, nepieciešams veikt šādus soļus:

1. Izmantojot augstākminēto pamata atslēgvārdu atbilstības izgūšanu, iegūt sākotnējo atbilstības modeli M.
2. Ja C sniedz papildu informāciju, kas attiecas uz lietotāja saskarnes šablona izvēli (piemēram, filtrēšanas kritēriji), nepieciešams precizēt "M", pamatojoties uz "C". Konkrēta pilnveidošanas loģika ir atkarīga no C rakstura un PL struktūras.
3. Atgrieziet precizēto šablonu "M".

**3.22. tabula**

**Formāls piemērs precizēšanai ar kontekstu**

| | |
|---|---|
| **P** | Search and Filter Products |
| **C** | {“Filter by Category”, “Filter by Price Range”} |
| **K** | {“Search Bar”: (keywords: {“Search”}), “Advanced Search”: (keywords: {“Search”, “Filter”})} |
| **PL** | {“Form”: (keywords: {“Create”, “Edit”}), “User Registration Form”: (keywords: {“Create”, “User Account”})} |
| **M** | “Advanced Search” (precizēts no sākotnējās “Search Bar”, pamatojoties uz filtrēšanas kontekstu) |

Ne visi procesu nosaukumi ir perfekti definēti – nenoteiktība var rasties dažādu iemeslu dēļ. Dažkārt, var būt sarežģīti kartēt lietotāja saskarnes šablonus, izmantojot tikai atslēgvārdus. Šādā gadījumā (3.23. tabula) ir jāatpazīst procesu nosaukumi, kuriem var būt vairākas interpretācijas, kas var būt balstīts uz:

1. Neskaidrība – procesu nosaukumi, piemēram, “*Manage User Permissions*”, rada neskaidrību interpretācijai. Vai šim lietotāja saskarnes šablonam ir jāietver lietotāja tiesību skatīšanās režīms, to rediģēšana vai abus?
2. Vairākas nozīmes – tādi termini kā “*Update*” var attiekties uz vispārīgu datu pārveidošanu vai konkrētu darbību, piemēram, jaunas datnes augšupielādi.

Izmantojot papildu informāciju par divpusložu modeļa procesu, var mēģināt atrisināt nenoteiktību:

1. Ar piemēru – piemēram, procesu “*Manage User Permissions*”. Pamatojoties uz kontekstu, varam noteikt, ka tas ietver lietotāju atlasi un konkrētu atļauju līmeņu piešķiršanu no iepriekš definēta saraksta.
2. Ar uzlabotu kartēšanu – papildu konteksts liecina, ka lietotāja saskarnes šabloni, piemēram, “*Editable Table*” ar izvēles rūtiņām vai nolaižamajām izvēlnēm atļauju atlasei, varētu būt labāk piemērota, salīdzinot ar vienkāršu “*List View*” šablonu.

Nenoteiktības apstrāde var atrisināt problēmas ar nekonkrētiem procesu nosaukumiem, tādējādi atbalstot precīzāku lietotāja saskarnes šablonu kartēšanu. Tam ir nepieciešama sīkāka informācija, lai izdarītu izvēli, taču tas ne vienmēr var atrisināt visas neskaidrības. Tomēr, dažos gadījumos pat ar papildu informāciju nenoteiktība var saglabāties. Tas varētu būt saistīts ar:

1. Ierobežotu kontekstu – procesa definīcijā vai konteksta definīcijā var trūkt pietiekami daudz informācijas.
2. Pēc būtības neskaidri nosaukumi – daži nosaukumi pēc būtības var būt neviennozīmīgi.
3. Atzīmēti pārskatīšanai – ja nenoteiktība paliek neatrisināta, procesa nosaukumu var atzīmēt manuālai pārskatīšanai, lai lietotāja saskarnes dizaineris vai izstrādātājs pēc procesa beigšanas var pārskatīt procesa definīciju un izvēlēties vispiemērotāko lietotāja saskarnes šablonu, pamatojoties uz cilvēka izpratni.

Nenoteiktības apstrāde palīdz kartēt procesu nosaukumus lietotāja saskarnes šabloniem, pirms laika atrodot nenoteiktību un izmantojot vairāk informācijas, kas padara visu transformāciju labāku.

**3.23. tabula**

**Nenoteiktības apstrāde**

| | |
|---|---|
| **Procesa nosaukums** | Manage User Permissions |
| **Atslēgvārdi** | “Manage”, “User”, “Permissions” |
| **Saderīgie šabloni** | “Manage” var tikt kartēts ar dažādiem šabloniem, piemēram, “List View” vai “Editable Table” atkarībā no lietotāja saskarnes šablonu bibliotēkas.<br>Šādā gadījumā var būt nepieciešama papildu informācija par to, kā tiek pārvaldītas atļaujas (piemēram, izvēles rūtiņas, izkrītošie saraksti), lai izvēlētos vispiemērotāko lietotāja saskarnes šablonu |

Nenoteiktības apstrādes noteikšanas gadījumā (skatīt 3.24. tabulu), ievade ir tikai P, bet izvade ir M vai arī saraksts ar attiecīgajiem lietotāja saskarnes šabloniem, lai to panāktu, nepieciešams veikt šādus soļus:

1. Izmantojot augstākminēto pamata atslēgvārdu atbilstības izgūšanu, iegūt sākotnējo atbilstības modeli M.

2. Ja M kardinalitāte ir lielāka par 1 (vairākas iespējamās atbilstības)
    a. Analizēt PL par šabloniem ar noteiktākiem atslēgvārdiem vai papildināt informāciju no procesa definīcijas, ja tāda ir pieejama.
    b. Ja tālāka konkretizēšana nav iespējama, atgrieziet iespējamo lietotāja saskarnes atbilstības sarakstu M, ar norādi par nenoteiktību.
3. Atgrieziet vienu atbilstošo lietotāja saskarnes šablonu M no M kopas.

**3.24. tabula**

**Formāls piemērs nenoteiktības apstrādei**

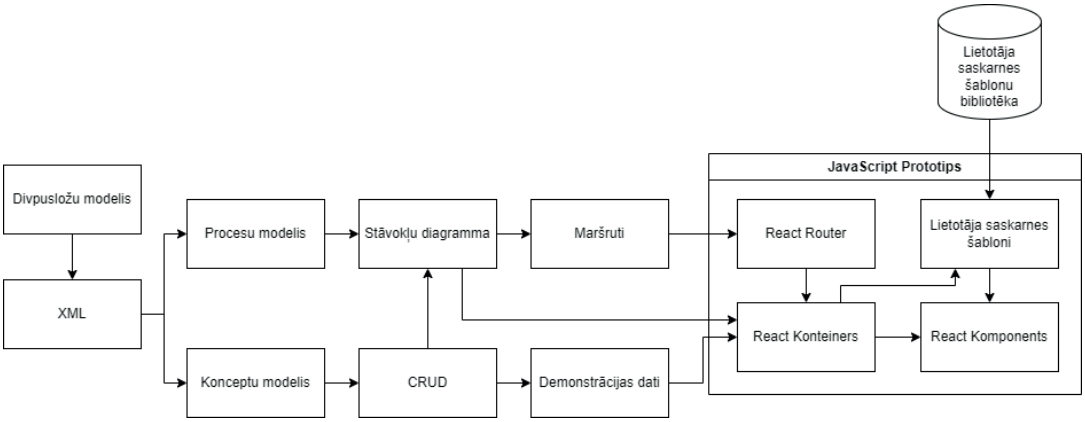
| | |
|---|---|
| **P** | Manage User Permissions |
| **PL** | {“List View”: (keywords: {“View”, “List”}), “Editable Table”: (keywords: {“Edit”, “Table”})} |
| **M** | Potenciālo šablonu saraksts: {“List View”, “Editable Table”} |

Izmantojot šo pieeju var veikt lietotāja saskarnes dizaina automatizāciju, pamatojoties uz procesu definīcijām, kas, savukārt, var paātrināt lietotāja saskarnes izstrādi. Kā arī šāda pieeja veicina lietotāja saskarnes šablonu konsekvenci visā lietojumprogrammā.

Tomēr, kā trūkumi ir jāatzīst, ka sarežģītos procesos var būt nepieciešama manuāla iejaukšanās, lai izvēlētos vispiemērotāko šablonu un šīs pieejas efektivitāte ir atkarīga no procesu nosaukumu un lietotāja saskarnes šablonu bibliotēkas kvalitātes.

## 3.6. Kopējais modeļu transformācijas algoritms

Lai uzsāktu transformācijas procesu (3.24. att.) sākumā ir nepieciešams divpusložu modelis. Visbiežāk, tas tiek veidots *BrainTool* rīkā un sastāv no procesa modeļa un koncepta modeļa. *BrainTool* piedāvā eksportēt šo modeli uz XML datnes formātu. Šo XML datni apstrādājot tiek iegūts gan procesu modelis, gan koncepta modelis ārpus *BrainTool* rīka apstrādājamā formā.




**3.24. att. Risinājuma kopējā shēma**

Koncepta modelis tālāk tiek kartēts ar *CRUD*, lai izgūtu datu modeļa entītijas un izlemtu, kādiem parametriem ir jābūt nodotiem. Iegūtie objekti un mainīgie tālāk tiek kartēti ar demonstrācija datiem, lai prototipos būtu aizpildīti skatījumi un varētu redzēt kā aptuveni izskatīsies gatava lietotāja saskarne. Mainīgie ar uzstādītām vērtībām tālāk tiek definēti *React.js* konteinerā. Savukārt no procesu modeļa tiek izgūta stāvokļa diagramma, kura tālāk tiek kartēta ar *React Router* maršrutiem. Maršruti tiek definēt atsevišķi, kopā ar *React.js* konteinera izsaukumiem, lai norādītu, kāds konteiners pie kāda maršruta izsaukuma tiek parādīts gala lietotājam. Kad ir sagatavotie gan mainīgie, gan maršruti, tiek izveidots *React.js* konteinera komponents, kurš tiek izvēlēts pamatojoties uz procesa modeļa vienuma nosaukuma izmantojot teksta klasifikācijas metodi – izgūts no lietotāju saskarnes šablonu bibliotēkas. Tādejādi tiek izgūts JavaScript prototips no divpusložu modeļa.

## 3.7. Nodaļas rezultāti un secinājumi

Divpusložu modeļa transformācija lietotāja saskarnes prototipos izmantojot modeļvadāmu pieeju ir sistemātisks process – tiek mainīts izskats, no procesiem un konceptiem uz JavaScript tīmekļa lietojumprogrammas prototipu. Šajā transformēšanas procesā tiek izmantots algoritms, kas ir izstrādāts, lai iespējotu lietotāja saskarnes šablonu pielietošanu, lai panāktu tīmekļa lietotne konsekvenci un atvieglotu gan programmētāja un lietotāja, gan arī visu citu iesaistīto personu

mijiedarbību, lai jau programmatūras izstrādes sākumā varētu pieņemt lēmumus par gala lietotāja saskarni.

Izmantojot šo automatizāciju, izstrādātāji var paātrināt lietotāja saskarnes prototipu izveidi, jo tie ir ļoti līdzīgi gala lietotāja saskarnei, nodrošinot viendabīgumu un precizitāti iegūtā lietotāja saskarnes dizainā. Turklāt šī metode ļauj lietotāja saskarnes veidotājiem viegli vizualizēt savas dizaina idejas. Var ātri izmēģināt dažādus stilus, krāsas un izkārtojumus, lai atrastu vislabāko lietotāja pieredzi. Šis process atbalsta ātru lietotāja saskarnes koncepciju iterāciju un apstiprināšanu, palīdzot efektīvi izstrādāt un uzlabot lietotāju pieredzi.

No otras puses, ir nepieciešams izstrādāt plašu un pietiekami elastīgu lietotāja saskarnes šablonu bibliotēku, kurā katrs no šabloniem būtu savienojams ar citiem un izstrādāts ar spēju iekļauties kopējā stila un izskata koncepcijā.

Ir ļoti svarīgi izstrādājot divpusložu modeli, lietot procesa nosaukumos semantiski atbilstošus lietotāja saskarnes šabloniem, lai efektīvi varētu izmantot visu šablonu piedāvātās priekšrocības un gala lietotājs saņemtu izstrādātāja plānotu prototipu.

Var piedāvāt arī paplašināt *BrainTool* rīka funkcionalitāti ar uz esošā procesa balstītu nākamo procesu prognozēšanu. Izveidojot procesu ar nosaukumu, kas ir pietiekami līdzīgs lietotāja saskarnes šablonu bibliotēkā atrodamajam, piedāvāt no izvēles iespējām, saderīgu saistīto procesu, kas izstrādātājam atvieglotu darbu un paaugstinātu risinājuma efektivitāti.

# 4. RISINĀJUMA APROBĀCIJA UN PĀRBAUDE

Promocijas darba ietvaros izstrādāta risinājuma pirmais solis ietver izpratni par konkrēto problēmvidi, kuram tiek izstrādāts lietotāja saskarnes prototips, un avota modeļa (divpusložu modeļa) izstrādi. Tālāk izveidotais avota modelis ir padots transformācijas likumu dzinējam un rezultātā ir iegūts lietotāja saskarnes prototips kā ekrānformu pirmkoda failu kopa, kas implementē problēmvides funkcionalitāti un lietotāja mijiedarbību problēmvidē. Automātiskā lietotāja saskarnes prototipu ģenerēšana izmanto iepriekš definētus lietotāja saskarnes šablonus kā dizaina pamatu. Lietotāja saskarnes šablonos ir izveidoti dizaina risinājumi izplatītiem lietotāja saskarnes elementiem un funkcijām, veicinot intuitīvu un lietotājam draudzīgu saskarni (Nikiforova, Babris et al., 2024b). Iepriekš definētu šablonu izmantošana var paātrināt prototipu veidošanas procesu, salīdzinot ar lietotāja saskarnes izveidi no jauna. Lietotāja saskarnes šabloni nodrošina konsekvenci starp prototipiem un lietotāja mijiedarbībā visā lietojumprogrammā. Iegūta transformācijas rezultāta atbilstība sagaidāmam rezultātam apliecina risinājuma darbspēju.

Lai validētu izstrādāto risinājumu iegūta rezultāta kvalitātes ziņā ir nepieciešams analizēt transformācijas rezultātu divos aspektos:

1) Iegūta koda kvalitāte, kas attiecas uz koda vispārējo efektivitāti un uzturēšanu. Tas attiecas ne tikai uz koda pareizu darbību, bet arī uz to, cik labi tas ir uzrakstīts, strukturēts un dokumentēts (Nikiforova, Zabiniako et al., 2021a). Kodam jābūt viegli saprotamam ikvienam izstrādātājam, kurš pārzina izmantoto programmēšanas valodu. Tas ietver pareizas atkāpes, jēgpilnus mainīgo nosaukumus un komentārus, kas izskaidro sarežģītu loģiku, ja tādi ir nepieciešami.
2) Iegūta tīmekļa lietojuma atbilstība problēmvides biznesa procesam un sagaidāmiem lietošanas scenārijiem, kas attiecās uz koda funkcionēšanu. Lietojuma funkcionēšanu ir iespējams pārbaudīt ar pieņemšanas testēšanas metodi (ISO/IEC/IEEE 29119, 2013), ar ko var apstiprināt, vai iegūtais prototips darbojas korekti un atbilst problēmvidē identificētiem biznesa procesiem.

Šajā sadaļā ir gan demonstrēta izstrādāta risinājuma pielietošana uz neliela abstrakta aprobācijas piemēra, kas apliecina risinājuma darbspēju, gan ir aprakstīta transformācijas rezultāta pieņemšanas testēšana, kas validē risinājuma pielietošanas rezultātu.

## 4.1. Problēmas vides apraksts un sagaidāmais rezultāts

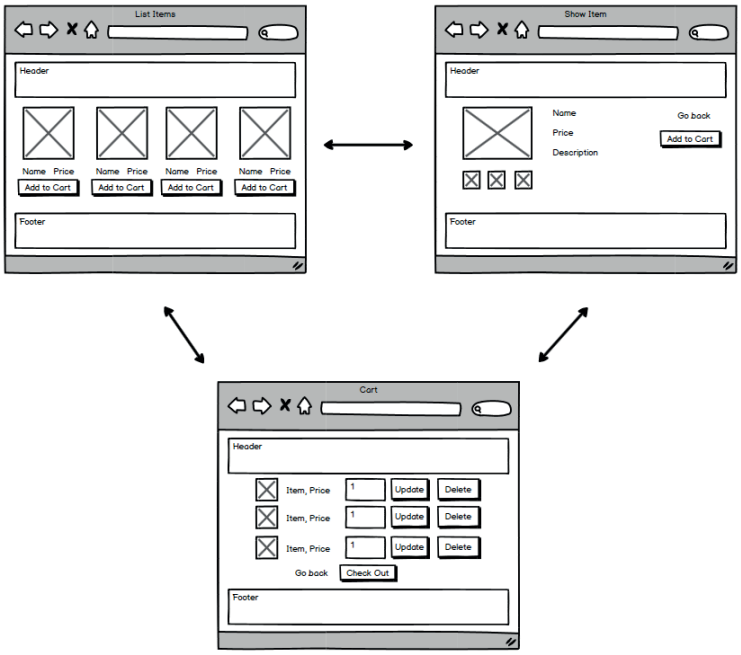
Lai demonstrētu piedāvāto pieeju tika izvēlēts piemērs. Šajā gadījumā piemērs ir interneta veikala lietošanas process uzņēmumam, kur var iegādāties abonementu uz kādu no uzņēmuma produktu. Uzņēmums nodarbojas ar biznesa analītikas risinājumiem priekš CRM un ERP sistēmām. Pārdodamais produkts ir moduļa veidā izstrādāta sistēma, kuru var viegli uzstādīt uz izvēlētajām platformām, neveicot kādas specifiskas pielāgošanas procesus. Produktu var iegādāties vai nu uz mēnesi vai abonēt uz veselu gadu, turklāt jāparedz iespēja šo abonementu automātiski pagarināt, ja ir izvēlēta tāda opcija. Šajā procesā lietotājs var reģistrēties, pieteikties, apskatīt preces, pievienot tās grozam un galu galā – izveidot pasūtījumu. Piedāvātais process iezīmē tipisku (Fernández, Liu et al., 2021) lietotāja mijiedarbību ar interneta veikalu. Process var sākties no reģistrācijas, pieteikšanās un produktu saraksta apskates. Pēc preču pārlūkošanas, iespējams preci pievienot iepirkumu grozam. Iepirkuma grozu var pārvaldīt – noņemt preces vai mainīt to daudzumu. Pēc tam ir iespējams noformēt pasūtījumu. Process ir aprakstīts sīkāk šādi (Babris & Nikiforova, 2024b):

1. Lietotāja reģistrēšana – Lietotājs apmeklējot interneta veikala vietni aktivizē pogu "Reģistrēties" un lietotājs tiek novirzīts uz reģistrācijas formu, kurā viņš ievada savus personas datus, piemēram, vārdu, e-pasta adresi, paroli. Pēc nepieciešamās informācijas aizpildīšanas lietotājs iesniedz formu. Sistēma izveido jaunu lietotāja kontu.
2. Lietotāja pieteikšanās – Ja lietotājam jau ir konts, viņš var pieteikties, noklikšķinot uz pogas "Pieteikties". Lietotājs pieteikšanās formā ievada savu e-pasta adresi un paroli. Pēc iesniegšanas sistēma pārbauda akreditācijas datus un, ja tie ir pareizi, piešķir piekļuvi lietotāja kontam.
3. Preču pārlūkošana – lietotājs var pārlūkot interneta veikala preču katalogu. Lietotājs var izmantot meklēšanas funkcionalitāti, lai atrastu konkrētus vienumus, kas viņus interesē. Katra prece tiek parādīta ar tās nosaukumu, cenu, aprakstu un iespēju to pievienot grozam.
4. Preču pievienošana grozam – Kad lietotājs atrod preci, kuru vēlas iegādāties, viņš noklikšķina uz pogas "Pievienot grozam". Atlasītā prece tiek pievienota lietotāja

iepirkumu grozam, kurā redzams visu līdz šim pievienoto preču kopsavilkums. Lietotājs var pielāgot katras preces daudzumu grozā vai izņemt preces pavisam.

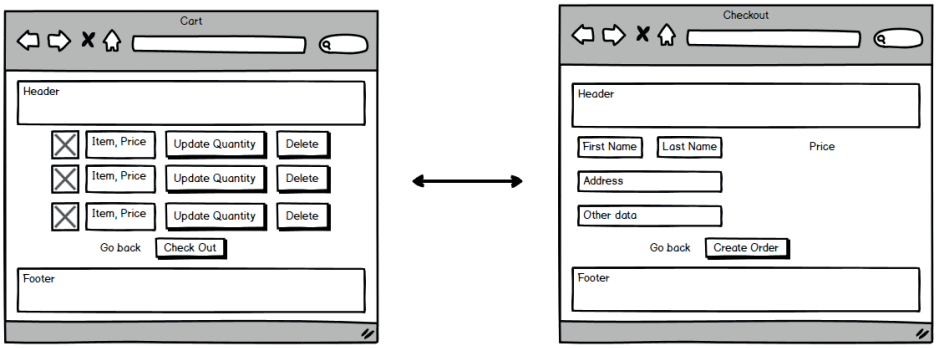
5. Groza pārvaldīšana – Groza skatā lietotājs var redzēt visas pievienotās preces, kā arī to daudzumus un cenas. Lietotājs var atjaunināt preču daudzumu, pielāgojot daudzuma lauku, vai noņemt preces, noklikšķinot uz pogas "Noņemt".
6. Norēķinu process – Pēc izvēles pabeigšanas lietotājs pāriet uz norēķināšanās skatījumu, noklikšķinot uz pogas "Izrakstīties". Lietotājam tiek parādīta norēķinu skatījuma forma, kurā viņš apstiprina savu piegādes adresi. Kad nepieciešamā informācija ir iesniegta, lietotājs iesniedz norēķinu formu un sistēma apstrādā pasūtījumu.
7. Pēc pasūtījuma veikšanas, lietotājs tiek pāradresēts uz savu profilu.

Lietotāji pāriet no preču kataloga, lai skatītu detalizētu informāciju par konkrētām precēm, kuras vēlas apskatīt. Tas ļaujot lietotājiem viegli izpētīt produkta informāciju, piemēram, aprakstus, attēlus un specifikācijas. Pēc preces informācijas pārbaudes un lēmuma par pirkšanu lietotāji var netraucēti pāriet uz iepirkumu groza zonu, lai pievienotu izvēlētās preces. Kā arī var no preču kataloga uzreiz pievienot preci grozam. 4.1. attēlā tiek demonstrēta preču kataloga, preču vienuma un preču groza shēma, kā var pārvietoties pa šiem elementiem.



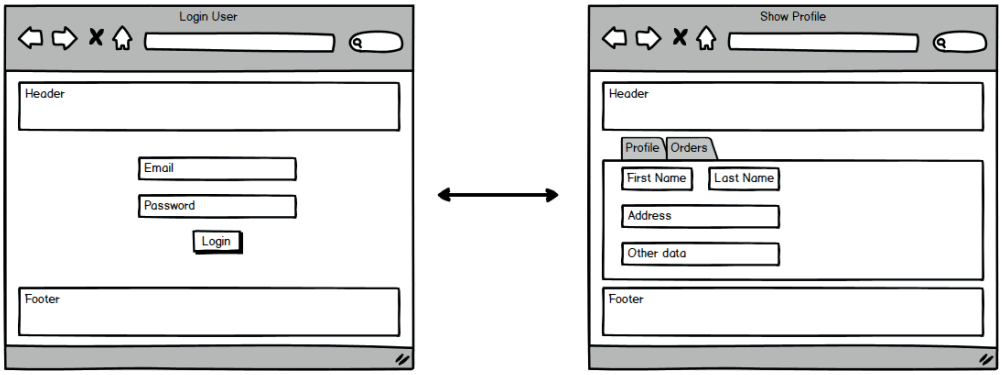

**4.1. att. Preču kataloga, vienuma un preču groza shēma**

Savukārt no preču groza var nokļūt vai nu uz norēķina sadaļu, kur nepieciešams ievadīt rekvizītus uz kuru sastādīt rēķinu, izvēlēties maksājuma veidus, utt., vai arī atpakaļ – uz preču katalogu (4.2 attēlā ir redzama preču groza un norēķina shēma).




**4.2. att. Preču groza un norēķina shēma**

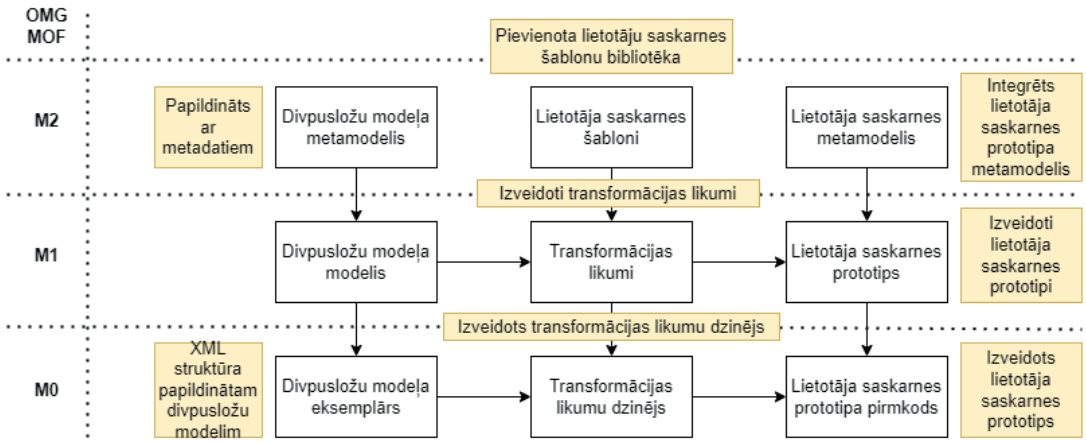
Savukārt, pabeidzot pasūtījumu, lietotājs tiek novirzīts uz lietotāja profilu (4.3. att.). Uz lietotāja profilu lietotājs tiek novirzīts arī gadījumā, ja ir notikusi lietotāja pieteikšanās sistēmā. Atslēdzoties no sistēmas, lietotājs tiek novirzīts uz pieteikšanās skatījumu.




**4.3. att. Lietotāja pieteikšanās un lietotāja profila shēma**

## 4.2. Risinājuma aprobācijas piemērs

Promocijas darba ietvaros izstrādātais lietotāja saskarnes prototipa ģenerēšanas risinājums no divpusložu modeļa ir parādīts 4.4. att., attēlojot risinājuma komponentes saskaņā ar objektu vadības grupas meta objektu spējas (angl. *Meta Object Facility*, *MOF*) ietvaru (https://www.omg.org/mof/). Ar blokiem baltā krāsā ir attēloti risinājuma komponenti. Ar komentāriem uz dzeltenā fona ir aprakstīti veiktie papildinājumi, modifikācijas vai jaunizstrādājumi, kas ir veikti promocijas darba ietvaros esošajiem artefaktiem.




**4.4. att. Risinājuma konceptuālā shēma un galvenie komponenti**

Risinājums ietver šādas komponentes:

1. **Modeļu transformācijas komponentes:**
   1.1. Transformācijas likumi, kas definē kā no divpusložu modeļa elementiem tiek veidoti lietotāja saskarnes elementi, ievērojot attiecīgos lietotāja saskarnes šablonus.
   1.2. Lietotāja saskarnes šabloni, kas ir promocijas darba jauninājums to lietošanas kontekstā, jo ir piedāvāts atkarībā no procesa metadatiem izvēlēties attiecīgo iepriekš definēto lietotāja saskarnes šablonu.
   1.3. Transformācijas likumu dzinējs, programmatūra, kas palaiž transformācijas likumus divpusložu modeļa XML datnei un rezultātā izveido tīmekļa lietotnes priekšgala komponenšu pirmkodu JavaScript programmēšanas valodā (promocijas darba demonstrācijai transformācijas likumi ir aprakstīti PHP programmēšanas valodā).
2. **Avota komponentes:**
   2.1. Divpusložu modeļa metamodelis, kas ir papildināts ar procesu metadatiem un ir realizēts UML klašu diagrammas veidā.
   2.2. Divpusložu modelis, kas ir veidots *BrainTool* rīkā un attēlo problēmvidi bez procesu metadatiem.

2.3. Divpusložu modeļa pirmkods XML valodā, kur XML struktūra ir izstrādāta, ievērojot veiktus divpusložu modeļa notācijas papildinājumus.

**3. Mērķa komponentes:**

3.1. Lietotāja saskarnes metamodelis, kas ir veidots, integrējot esošos saskarnes metamodeļus un pārstrukturējot tos attiecībā uz darbā veikto lietojumsistēmas priekšgala komponentu kartēšanu un sagaidāmo tīmekļa lietotnes lietotāja saskarnes elementu kopu (aprakstīta UML klašu diagrammas veidā).

3.2. Lietotāja saskarnes prototips, kas pēc būtības ir strādājoša tīmekļa lietotne kā mijiedarbojošās ekrānformas palaistas tīmekļa pārlūkā.

3.3. Tīmekļa lietotnes priekšgala komponenšu pirmkods, kas ir palaižams kādā integrētajā izstrādes vidē ar realizācijai izvēlēta satvara atbalstu (promocijas darbā demonstrācijas piemēra ir izmantotas *React*.js, *Redux.js*, *XState* bibliotēkas).

Attiecībā uz divpusložu modeļa notāciju un metamodeli ir veiktas minimālas metamodeļa izmaiņas, galvenokārt – procesu modelī, faktiski neizmainot tā projektējumu. Izmaiņas norāda tikai uz to vai process ir redzams lietotājam vai arī šis process priekš lietotāja saskarnes ir jāignorē un jāizmanto nākošais process procesu kartē.

Divpusložu modeļa metamodelis attēlo modeļa augstāka līmeņa abstrakciju. Šajā kontekstā divpusložu metamodelis ir ticis papildināts ar metadatiem. Piemēram, procesa modelis ir papildināts ar marķējumu vai process ir jāredz lietotājam vai nē – vai tas ir lietotāja saskarnes skatījums vai nē. Kā arī konceptu modelis ir papildināts ar koncepta atribūtu datu tipiem, lai vēl precīzāk varētu transformēt koncepta datus.

Lietotāja saskarnes šablonu bibliotēkas izmantošana sniedz izstrādātājiem gatavus un papildināmus risinājumus tipiskām dizaina problēmām. Tas padara viegli saprotamu un konsekventu saskarņu izveides procesu daudz vienkāršāku. Šajos šablonos var iekļaut labākās metodes un dizaina idejas, kas ļauj izstrādātājam izvēlēties un mainīt tās, ja nepieciešams konkrētām dizaina situācijām.

Transformācijas likumi izveido organizētas saiknes starp sarežģītākām dizaina idejām, piemēram, procesu nosaukumiem un faktiskajiem lietotāja saskarnes šabloniem. Šie likumi palīdz pārvērst abstraktā dizaina idejas faktiskās saskarnes daļās, padarot projektēšanas procesu sistemātiskāku un saglabājot dizainu vienveidību. Risinājums var ietvert arī ieteikumus par

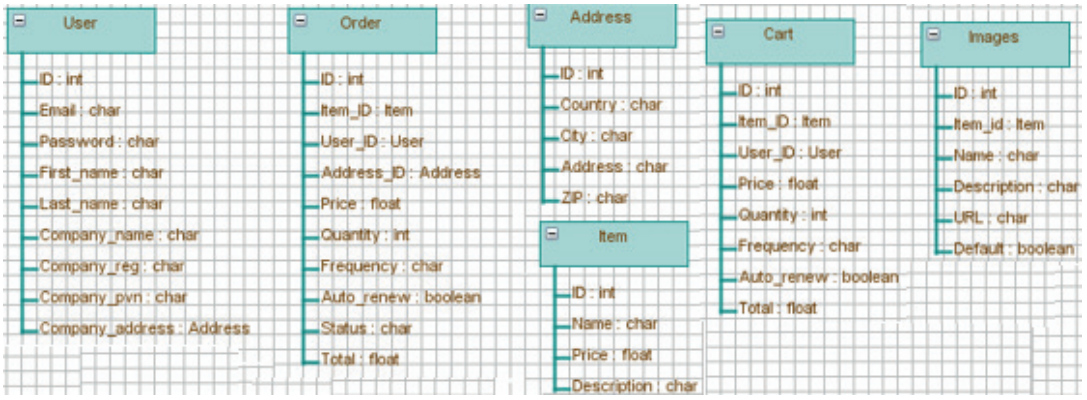
lietojamību, pieejamību un reaģējošu dizainu, ko var izmantot izveides posmā, lai pārliecinātos, ka saskarnes ir piemērotas dažādiem lietotājiem un platformām. Apvienojot metamodeļu izmaiņas ar lietotāja saskarnes šabloniem, transformācijas likumiem utt. izstrādātāji var izveidot risinājumus, kas labi darbojas, padarot saskarnes viegli lietojamas, vienlaikus precīzi un laicīgi saskaņojot izskatu ar gala lietotāju.

### 4.2.1. Problēmvides divpusložu modelis kā avota modelis

4.5. att. ir parādīta risinājuma demonstrācijai un aprobācijai izvēlētas problēmvides koncepti un 4.6. att. – tās galvenie procesi, kas kopā tiek apvienots divpusložu modelī, *BrainTool* rīkā uzdodot attiecības starp attiecīgajām procesu plūsmām un konceptiem. Par problēmvidi ir izvēlēts tipisks interneta veikals, kurš var tikt lietots dažu produktu izvēlei un abonēšanai, tas nesastāv no kategoriju hierarhijām vai citādām kompleksām klasifikācijas sistēmām. Galvenā prasība ir, lai potenciālais vai esošais uzņēmuma lietotājs varētu apmaksāt sistēmas abonementu par viņam izvēlētu laika periodu.

Galvenie sistēmas objekti ir lietotājs, prece, iepirkumu grozs un norēķins jeb pasūtījuma apstiprināšana no lietotāja puses. Šajā sistēma nav paredzēts integrēt maksāšanas sistēmas.

Preces (angl. *Items*) – entītija apzīmē preces, kuras var iegādāties interneta veikalā. Katrs vienums parasti ietver tādus atribūtus kā nosaukums, apraksts, cena un attēls. Attēlam ir informatīva nozīme un tie var būt vairāki, lai pilnīgāk sniegtu informāciju par potenciāli iegādājamo produktu (1.pielikums 1. tabula). Dotajā piemērā, preces abonēšanas cena ir atkarīga no daudzuma – šajā gadījumā lietotāju skaita, kuri lietos abonējamo produktu.




**4.5. att. Problēmvides divpusložu modeļa koncepta modelis**

Lietotāji (angl. *Users*) – Lietotāji ir subjekti, kas mijiedarbojas ar interneta veikalu. Tie ietver tādus atribūti kā lietotājvārds, e-pasts, parole (1.pielikums 2. tabula). Turklāt lietotājam ir nepieciešams norādīt uzņēmuma rekvizītus. Lietotājiem ir paredzēta iespēja pieteikties (angl. *Login*), kā arī reģistrēties (angl. *Register*). Sakarā ar to, ka lietotājs, galvenokārt, ir paredzēts kā kādas juridiskas personas pārstāvis (produkts ir paredzēts uzņēmumiem), tad pie lietotāja ir paredzēts ievadīt arī uzņēmuma informāciju – uzņēmuma nosaukumu, reģistrācijas numuru, PVN numuru un uzstādīt uzņēmuma lietotāja juridisko adresi.

Attēli ir svarīgs preces apraksta elements (1.pielikums 3. tabula) un tam ir nepieciešams izvēlēties galveno attēlu (Default) un paskaidrojošos attēlus, kurus var attēlot galerijas veidā pie produkta detalizētā skatījuma.

Adrese ir nepieciešama, lai norādītu uzņēmuma lietotāja juridisko adresi uz kuru saņemt rēķinu, ja tas ir nepieciešams un faktisko adresi, ja tās atšķiras 1.pielikums 4. tabula). Adrese tiek piesaistīta gan pie lietotāja entītijas, kā galvenā – juridiskā adrese, gan pie pasūtījuma, kā faktiskā.

Grozs (angl. *Cart*) – Grozs ir pagaidu krātuve precēm, ko lietotāji atlasījuši iegādei pirms norēķināšanās. Tas ietver asociācijas ar lietotāju pievienotajām precēm, kā arī daudzumiem. Groza vienība atvieglo preču pārvaldību visas iepirkšanās sesijas laikā. Groza funkcionalitātē (1.pielikums 5. tabula) ietilpst iespēja mainīt abonējamā produkta lietotāju daudzumu var izņemt produktu no groza. Grozs ir paredzēts tikai priekšgala lietotāja saskarnes sistēmā lietotāja sesijas laikā.

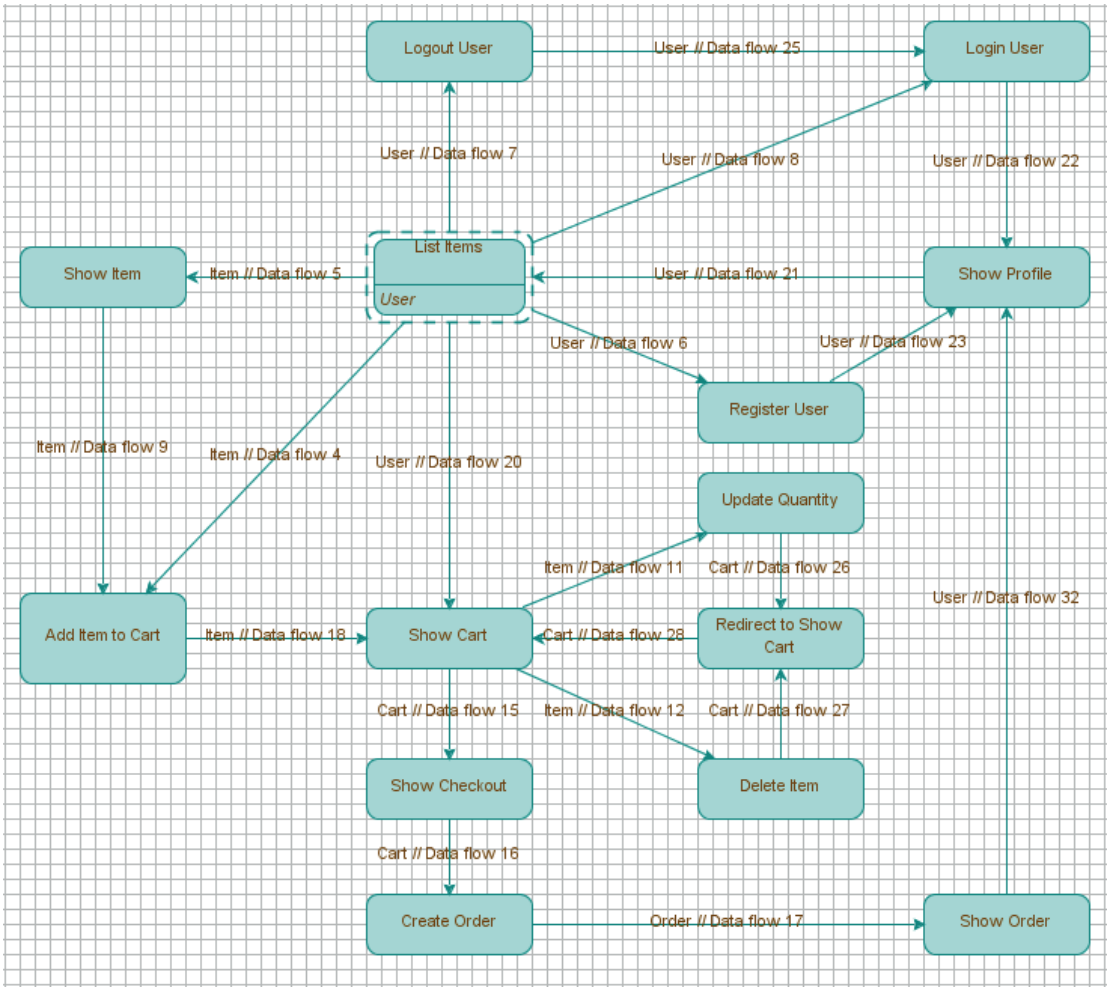
Pasūtījums ir gala rezultāts lietotāja iepirkšanās procesam norādītajā interneta veikalā. Pasūtījums sevī ietver gan groza saturu, gan norādi uz lietotāju, gan arī adreses saturu 1.pielikums 6. tabula). Kā arī pasūtījums tiek papildināts ar statusa atribūtu, lai atšķirtu jaunus pasūtījums no jau apkalpotajiem.

4.6. att. ir parādīti divpusložu modeļa procesi – sava veida lietošanas gadījumus, kas definē to, kā lietotājs mijiedarbojas ar sistēmu un no kura procesa lietotājs var turpināt sistēmas navigāciju uz nākamajiem procesiem.

Interneta veikals jeb e-komercijas platforma ietver dažādas vienības un procesus, kas paredzēti efektīvai iepirkšanās veicināšanai tiešsaistē. "*List Items*" process ietver pieejamo produktu kataloga parādīšanu, kas bieži tiek iedalīts kategorijās, lai atvieglotu navigāciju. Katru produktu var izpētīt tālāk, izmantojot "*Show Item*" procesu, kurā tiek sniegta detalizēta informācija, piemēram, apraksti, cenas, attēli utt. Pēc tam klienti var pievienot vēlamos produktus saviem virtuālajiem iepirkumu groziem, izmantojot funkciju "*Add item to cart*". "*Show cart*" process ļauj lietotājiem pārskatīt izvēlētās preces, kur viņi var atjaunināt daudzumus vai noņemt preces pēc vajadzības.

Kad klienti ir gatavi pirkt, "*Show Checkout*" process palīdz viņiem ievadīt piegādes un maksājumu informāciju. Pabeidzot šo darbību, tiek aktivizēts pasūtījuma izveides process, kas pabeidz pirkumu un ģenerē pasūtījuma apstiprinājumu. Klienti var skatīt savus iepriekšējos pirkumus, izmantojot funkciju "*Show orders*", sniedzot tādu informāciju kā, piemēram, pasūtījuma statuss

Lietotāju pārvaldības procesi arī ir neatņemama interneta veikala pieredze. "*Login User*" process pārbauda lietotāja pieteikšanās datus, piešķirot piekļuvi personalizētām funkcijām, piemēram, funkcijai "*Show Profile*", kurā lietotāji var skatīt un atjaunināt savu personisko informāciju, adreses utt. "*Logout User*" process droši pabeidz lietotāja sesiju, nodrošinot viņa konta aizsardzību. Kopā šīs vienības un procesi rada vienotu un lietotājam tipisku vidi iepirkšanās tiešsaistē.




**4.6. att. Problēmvides divpusložu modeļa procesa modelis**

Lietotāja pieteikšanās, reģistrācijas un atteikšanās process ir ļoti svarīgs piedāvātajā interneta veikalā. Tas palīdz nodrošināt drošu piekļuvi lietotāja kontam un viņa veiktajiem pasūtījumiem.

Lietotāju reģistrācija sākas ar formu, kurā cilvēkiem tiek lūgts ievadīt nepieciešamo informāciju – dati par lietotāju, uzņēmuma rekvizīti un juridiskā adrese. Aizpildot šo nepieciešamo informāciju sistēma piereģistrē lietotāju un pāradresē uz lietotāja kontu.

Lai pieteiktos (angl. *Login*) jau reģistrētam lietotājam, nepieciešams ievadīt e-pastu un paroli un veiksmīgas autentifikācijas gadījumā lietotājs tiek pāradresēts uz lietotāja konta sadaļu.

Lai veiktu atteikšanos (angl. *Logout*) no sistēmas, nepieciešams izvēlēties attiecīgo aktivitāti no lietotāja saskarnes. Tas ietver notiekošās sesijas pārtraukšanu, visu sesijas datu noņemšanu un lietotāja novirzīšanu uz lietotāja pieteikšanās formu.

Neatkarīgi no tā vai lietotājs ir reģistrējies un pieteicies vai sistēma tiek pārlūkota izmantojot nereģistrētu lietotāju ir iespējams apskatīt piedāvātos produktus. Produktu katalogā ir redzams produktu saraksts ar tā nosaukumu, attēlu, cenu un iespēju pievienot šo produktu preču grozam.

Atverot noteiktu produktu preču detalizētajā skatījumā ir iespējams aplūkot produkta nosaukumu, aprakstu, attēlu, vairākus miniatūrus attēla eksemplārus un iespēju pievienot šo produktu preču grozam.

Preču grozā var aplūkot izvēlētos produktus, lietotāju skaitu katram produktam un gala vērtību visu produktu un daudzuma kopsummu. Preču grozā var mainīt lietotāju skaitu vai vispār izņemt produktu no preču groza.

Nākamais solis procesā ir norēķins (angl. *Checkout*), kurā tiek pieprasīts ievadīt pasūtījumam nepieciešamo informāciju. Kad visa informācija ir ievadīta, tad tiek izveidots pasūtījums un lietotājs pāradresēts uz lietotāja kontu, kur var redzēt visus lietotāja pasūtījumus.

Tā kā divpusložu modelis ir veidots *BrainTool* rīkā, informāciju par tajā esošajiem konceptiem, procesiem un to savstarpējām saistībām ir iespējams transformēt apstrādājamā formā, t.i. eksportēt attiecīgo XML (paplašināmā iezīmēšanas valoda) datni.

XML ir iezīmēšanas valoda, kas nosaka noteikumu kopumu dokumentu kodēšanai formātā, kas ir gan cilvēkiem, gan mašīnlasāms. To parasti izmanto strukturētu datu glabāšanai hierarhiskā formātā. Divpusložu modeļa avota modelis, parasti ir XML datne, kas attēlo procesus un konceptus. Šajā divpusložu modelī avota modeļa XML datnes struktūra definē elementus un atribūtus, kas var tikt izmantota lietotāja saskarnes komponentu ģenerēšanā, kā arī to īpašības, attiecības un uzvedību. Divpusložu modeļa ietvaros atrasto sadalījumu var redzēt avota modeļa XML datnes struktūrā (4.1. tabula). Tajā esošie elementi tiek izmantoti, lai definētu abas daļas, kas saistītas ar konceptuālajiem aspektiem, kā arī procesa attēlojumu lietotāja saskarnēm. Divpusložu modelī XML elementi varētu aprakstīt entītijas, funkcijas, saites un darbības, kuras tālāk tiktu

izmantotas darbā piedāvātajā risinājumā. No divpusložu modeļa ir iespējams arī ģenerēt secību diagrammas (Nikiforova, Kozacenko et al., 2013) un UML klašu diagrammas (Nikiforova & Pavlova, 2011), kas liecina par to, ka divpusložu modelis ir pietiekami funkcionāls, lai no tā varētu ģenerēt lietotāja saskarnes elementus.

**4.1. tabula**

**Divpusložu modeļa XML datne**

| **Elements** | **Apraksts** |
|---|---|
| models | Satur divpusložu modeļus, kuri tiek attēloti |
| diagramObjects | Visi elementi, kuri tiek attēloti diagrammās |
| concept | Koncepta diagrammas vienums un tā saturs |
| internalProcess | Iekšējā procesa definīcijas |
| externaProcess | Ārējā procesa definīcija |
| dataFlow | Datu plūsmas definīcija |
| diagrams | Satur procesu un konceptu diagrammas |
| conceptualDiagram | Konceptuālās shēmas definīcija |
| processDiagram | Procesu diagrammas definīcija |

Divpusložu modelī XML var šo struktūru paplašināt pamatojoties uz divpusložu modeļa pievienotajām īpašajām prasībām un elementiem.

### 4.2.2. Mērķa modeļa pirmkoda datne un datu formāts

Risinājuma izejas dati tiek pasniegti kā *React.js* JavaScript pirmkods. Katram satvaram vai bibliotēkai var tikt izstrādāta savi modeļa transformēšanas likumi, lai iegūtu pilnvērtīgu lietotāja saskarni.

Ņemot vērā, ka viena no detalizētākajām lietotāja saskarnes modelēšanas pieejām ir IFML (angl. *Interaction Flow Modeling Language, IFML*), tad arī piedāvātajā risinājumā tiek aizgūti tādi elementi kā *Container* un *Component*, lai atdalītu funkcionālos un konteksta elementus no prezentācijas elementiem. Piemērojot šādu arhitektūru var iegūt *React.js* un *Redux.js* kodu. Sakarā ar to, ka paši konteineru un komponentu veidnes tiek manuāli izveidotas, tad tālāk 4.2. tabulā var

redzēt konteinera veidnes satura galvenos elementus, savukārt, 4.3. tabulā var redzēt komponenta veidnes satura galvenos elementus. Šīs veidnes tiek izmantotas transformācijas procesā, lai funkcionāli izvietotu lietotāja saskarnes šablonus un datu modeļus.

**4.2. tabula**

**Konteinera veidnes saturs**

| **Elements** | **Apraksts** |
|---|---|
| <ComponentName> | Piesaistes komponenta nosaukums |
| <ContainerDependencies> | Konteinera atkarības |
| <ContainerConceptDefinitions> | Konteinera koncepta vērtību objekts |
| <ContainerConcepts> | Konteinera konceptu sarakts |
| <ContainerMethods> | Konteinera metodes, ja tādas ir |

*React.js* un *Redux.js* tīmekļa lietojumprogrammā ir daudzas daļas, kas palīdz organizēt un kontrolēt lietotāja saskarni, kā arī stāvokļa pārvaldību. Piemēram, <*ComponentName*> ir tehniskais nosaukums *React.js* komponentam, ko var izmantot atkārtoti dažādās projekta vietās. Šis nosaukums ir atbildīgs par konkrētas lietotāja saskarnes daļas rādīšanu ekrānā, vienlaikus kontrolējot, kā šis apgabals darbojas. Šie padara tos viegli atkārtoti lietojamus un atdalāmus no citām daļām.

Elements ar nosaukumu <*ContainerDependencies*> satur norādes uz izmantojamiem komponentiem vai konteineriem, tas var saturēt arī norādes uz kādu noteiktu bibliotēku lietošanu.

Elements <*ContainerConceptDefinitions*> satur galvenās datu struktūras vai entītijas, ko apstrādā tīmekļa lietojumprogramma. Izmantojot demonstrācijas datu ģeneratoru, var aizpildīt šīs datu struktūras tādā veidā, lai lietotājam šos datus būtu viegli uztverams un radītu prototipu, kas būtu maksimāli tuvs reālai lietotāja saskarnei.

Elements <*ContainerConcepts*> ir iepriekš definēto <*ContainerConceptDefinitions*> elementu saraksts, kas tiek nodots komponentam, no konteinera definīcijas. Faktiski <*ContainerConcepts*> ir <*ContainerConceptDefinitions*> vienkāršojums.

Pēdējais elements <ContainerMethods> var ietvert dažādas metodes vai funkcijas, kas pieder šim konteinera komponentam. Parasti šīs metodes apstrādā tādas lietas kā lietotāja darbības un datu apstrāde, kā arī navigāciju (4.7. att.).

Arī komponenta gadījumā <ComponentName> ir komponenta tehniskais nosaukums un elements ar nosaukumu <ContainerDependencies>, līdzīgi kā <ContainerDependencies> satur norādes uz izmantojamiem komponentiem vai konteineriem, kā arī uz šablonu bibliotēkām vai citām nepieciešamajām atkarībām.

**4.3. tabula**

**Komponenta veidnes saturs**

| **Elements** | **Apraksts** |
|---|---|
| <ComponentName> | Komponenta nosaukums |
| <ComponentDependencies> | Komponenta atkarības |
| <ComponentConcepts> | Komponentu koncepta vērtību saraksts |
| <ComponentContent> | Komponenta saturs |

Elements, <ComponentConcepts> tāpat kā <ContainerConcepts> ir iepriekš definēto konceptu elementu saraksts, kas tiek nodots komponentam, no konteinera definīcijas. Komponents šādā veidā var tikt lietots atkārtoti arī citās projekta daļās (4.8. att.).

Pēdējais elements <ComponentContent> ir vissvarīgākais elements, jo satur sevī lietotājam attēlojamo elementu kopumu, kas definēts lietotāja saskarnes šablonā.

Sakarā ar to, ka tiek ģenerēti prototipi, tad bez datiem, nebūs pilnīgas iespējas gala lietotājam saprast kā rezultāti izskatīsies lietotāja saskarnē. Lai risinātu šo problēmu, tiek piedāvāts izmantot demonstrācijas datus, kuri nenes nekādu informāciju, bet tiek izmantoti tikai, lai lietotājam demonstrētu maksimāli reālu prototipu.

```
import React from 'react';
import { connect } from 'react-redux';
import { faker } from '@faker-js/faker';
import { goNext, getStateMeta } from "./shared"

<ContainerDependencies>

import <ComponentName> from "./<ComponentName>"


function mapStateToProps(state, props) {
    const currentState = props.stateActor.getSnapshot();
    const meta = getStateMeta(currentState);

    <ContainerConceptDefinitions>

    return {
        title: meta.title,
        <ContainerConcepts>
    };
}

function mapDispatchToProps(dispatch, props) {
    return {
        goNext: (nextStateName) => {
            return goNext(nextStateName, props.stateActor)
        },
        <ContainerMethods>
    }
}

export default connect(
    mapStateToProps,
    mapDispatchToProps
)(<ComponentName>);
```

**4.7. att. Minimālā konteinera veidne**

```
import * as React from "react";
import { useNavigate } from "react-router-dom";

<ComponentDependencies>

export default function <ComponentName>(props) {
    const navigate = useNavigate();
    const { goNext, title <ComponentConcepts>} = props;

    return (<ComponentContent>)
}
```

**4.8. att. Minimālā komponenta veidne**

Pamatojoties uz popularitāti un atbalsta klāstu, par demonstrācijas datu ģenerēšanai tika izvēlēta JavaScript bibliotēka *Faker* (https://fakerjs.dev/). *Faker* datu kopas tiek ģenerētas izmantojot divpusložu modeļa konceptus, atribūtus un tipus (4.4. tabulā). Var pielietot arī citas bibliotēkas datu ģenerēšanai, ja tās var kartēt ar konceptiem. Bez šādas kartēšanas var pielietot

manuālu aizpildīšanu pēc divpusložu modeļa koncepta atribūtu tipiem, tomēr, šādā gadījumā ģenerētie dati zaudētu savu nosacīto, bet tomēr – kontekstu. Lietotāja saskarnes prototipiem, tomēr, ir vēlams būt maksimāli pietuvinātiem reālā gala lietotāja saskarnei, lai gūtu maksimālu labumu no izveidotajiem prototipiem. Tieši specializētas datu ģenerēšanas bibliotēkas nodrošina šādu iespēju.

**4.4. tabula**

**Divpusložu modeļa konceptu kartēšana ar *Faker* bibliotēkas elementiem**

| **Divpusložu modeļa koncepta elements** | **Faker elements** | **Kartēšanas elements** |
|---|---|---|
| **Item** | | |
| ID (Int) | faker.number.int() | ID |
| Name (Char) | faker.commerce.product() | Name |
| Price (Float) | faker.commerce.price() | Price |
| Description (Char) | faker.commerce.productDescription() | Description |
| **Image** | | |
| ID (Int) | faker.number.int() | ID |
| Item_ID (Item) | Item.id | Item.ID |
| Name (Char) | faker.commerce.product() | Name |
| Description (Char) | faker.commerce.productDescription() | Description |
| URL (Char) | faker.image.urlLoremFlickr({category: abstract}) | ImageURL |
| Default (Int) | faker.datatype.boolean() | Default |

Kā redzams 4.4. tabulā (Divpusložu modeļa konceptu kartēšana ar *Faker* bibliotēkas elementiem), tad *Faker* automātiskai elementu kartēšanai var piemērot līdzību ar lietotāja saskarnes šablonu bibliotēku un šablona atlasīšanas pieeju (teksta klasificēšana). Piemērojot divpusložu modeļa koncepta diagrammas elementu nosaukumu kopā ar atribūtu nosaukumu, iespējams pietiekami precīzi noteikt jau iepriekš kartētus *Faker* elementus. Piemēram, "*faker.commerce.product()"* var kartēt kā "*Name*" tādejādi iegūstot projekta ietvaros kartētu atribūtu "Name", kuru var atlasīt no demonstrāciju datu kartētiem elementiem (4.9. att.).

```
import React from 'react';
import { connect } from 'react-redux';
import { faker } from '@faker-js/faker';
import { goNext, getStateMeta } from "./shared"

import Item from "./Item"

function mapStateToProps(state, props) {
    const currentState = props.stateActor.getSnapshot();
    const meta = getStateMeta(currentState);

    const item = {
        id: faker.number.int(),
        name: faker.commerce.product(),
        price: faker.commerce.price(),
        description: faker.commerce.productDescription()
    }

    const images = new Array(5).fill().map(() => {
        return {
            id: faker.number.int(),
            item_id: item.id,
            name: faker.commerce.product(),
            description: faker.commerce.productDescription(),
            url: faker.image.urlLoremFlickr({ category: 'abstract' }),
            default: faker.datatype.boolean()
        }
    })

    return {
        title: meta.title,
        item: item,
        images: images
    };
}

function mapDispatchToProps(dispatch, props) {
    return {
        goNext: (nextStateName) => {
            return goNext(nextStateName, props.stateActor)
        },
    }
}

export default connect(
    mapStateToProps,
    mapDispatchToProps
)(Item);
```

**4.9. att. "*ShowItem*" konteinera kods**

Gadījumā, kad divpusložu modeļa koncepti ir savstarpēji saistīti ar relāciju, piemēram, kā gadījumā ar konceptiem "*Item*" un "*Image*" (vienai precei var būt piesaistīti vairāki attēli), tad šādā gadījumā tiek brīvi noteikts saistīto elementu daudzums ar noteiktu konstanti, kas tiek izmantota

transformēšanas procesā. Šajā gadījumā konstantes vērtība ir 5, bet šī konstante var būt brīvi izvēlēta. "*ShowItem"* komponenta kodu var redzēt 4.10. attēlā.

```
import * as React from "react";
import { useNavigate } from "react-router-dom";

import Box from '@mui/material/Box';
import Card from '@mui/material/Card';

import ItemDetails from "./ItemDetails"
import Buttons from "./Buttons"

export default function ShowItem(props) {
    const navigate = useNavigate();
    const { goNext, title, item, images } = props;

    return (
        <div>
            <h2>{title}</h2>

            <Box sx={{
                flexGrow: 1,
                border: '1px dashed grey',
                marginBottom: "10px",
                padding:"10px"
            }}>
                <Card sx={{ display:"flex", width:"100%" }}>

                    <ItemDetails
                        item={item}
                        images={images}
                    />

                    <Buttons
                        backLabel="Back"
                        nextLabel="Add Item To Cart"
                        onBackClick={() => navigate(goNext("ON_LIST_ITEMS"))}
                        onNextClick={() => navigate(goNext("ON_ADD_ITEM_TO_CART"))}
                    />

                </Card>
            </Box>
        </div>
    )
}
```

**4.10. att. "*ShowItem"* komponenta kods**

Kā redzams 4.9. attēlā un 4.10 attēlā, tad var secināt, ka iegūtie elementi ir ļoti tuvi gala lietotāja saskarnei, faktiski, nepieciešams ir konteineram nodot reālos datus un izmaiņas komponentā gandrīz nav jāveic. Protams, viss ir atkarīgs no tā vai gala lietotāju apmierina no divpusložu modeļa iegūtais prototips – ja ir nepieciešams veikt kādas izmaiņas, tad šīs izmaiņas

var veikt pašā šablonā (no šablonu bibliotēkas) un veikt atkārtotu transformāciju no divpusložu modeļa.

### 4.2.3. Avota un mērķa modeļu kartēšana

Avota un mērķa modeļa kartēšana ar *React Router*, konteineriem un komponentiem nozīmē arhitektūras modeļa attēlojuma pārveidošanu maršrutu un komponentu izkārtojumā *React.js* tīmekļa lietojumprogrammā. *React Router* ir rīks, kas palīdz savienot katru konteineru un komponentu ar noteiktu maršrutu *React.js* tīmekļa lietojumprogrammā. Tādēļ nepieciešams definēt maršrutus, kas atbilst dažādiem divpusložu modeļa procesiem. Veicot šīs darbības, izstrādātāji var kartēt divpusložu modeļa procesu plūsmu ar *React Router* maršrutētāju, konteineriem un komponentiem (4.5. tabula). Tādējādi tiek izveidots vienots, funkcionāls tīmekļa lietotnes prototips, ar ko lietotājs var mijiedarboties.

**4.5. tabula**

**Divpusložu modeļa procesu kartēšana ar React.js elementiem**

| **Divpusložu modeļa process** | **Lietotāja saskarnes šablons** | **React.js Konteiners** | **React.js Komponents** | **React Router Maršruts** |
|---|---|---|---|---|
| List Items | Internet shop catalogue | ListItems | ListItems | /list_items |
| Show Item | Internet shop Item view | ShowItem | ShowItem | /show_item |
| Show Cart | Internet shop item cart | ShowCart | ShowCart | /show_cart |
| Delete Item | - | - | - | /show_cart |
| Update Quantity | - | - | - | /show_cart |
| Add Item to Cart | - | - | - | /show_cart |
| Show Checkout | Internet shop checkout | ShowCheckout | ShowCheckout | /show_checkout |
| Login User | Login user form | LoginUser | LoginUser | /login_user |
| Show Profile | User profile | ShowProfile | ShowProfile | /show_profile |
| Register User | User registration form | RegisterUser | RegisterUser | /register_user |
| Logout User | - | - | - | /login_user |

Kā arī nepieciešams kartēt (4.6. tabula) divpusložu modeļa konceptus ar *React.js* konteineriem un komponentiem, lai varētu zināt, kādi datu modeļi nepieciešami katrā konteinerā, lai varētu veikt to definēšanu. Kā var ievērot, tad ja mainīgā tips ir masīvs, tad divpusložu modeļa koncepts ir jāizmanto daudzskaitlī un pie transformācijas tiek pārvērts ar objektu masīvu. Savukārt, ja sagaidāmā mainīgā tips ir objekts, tad mainīgais tiek saukts vienskaitlī un atstāts tāds kā divpusložu modeļa koncepta nosaukums. Mainīgo tipi tiek definēti pie lietotāja saskarnes šablona, tas tiek darīts manuāli veidojot šablonu.

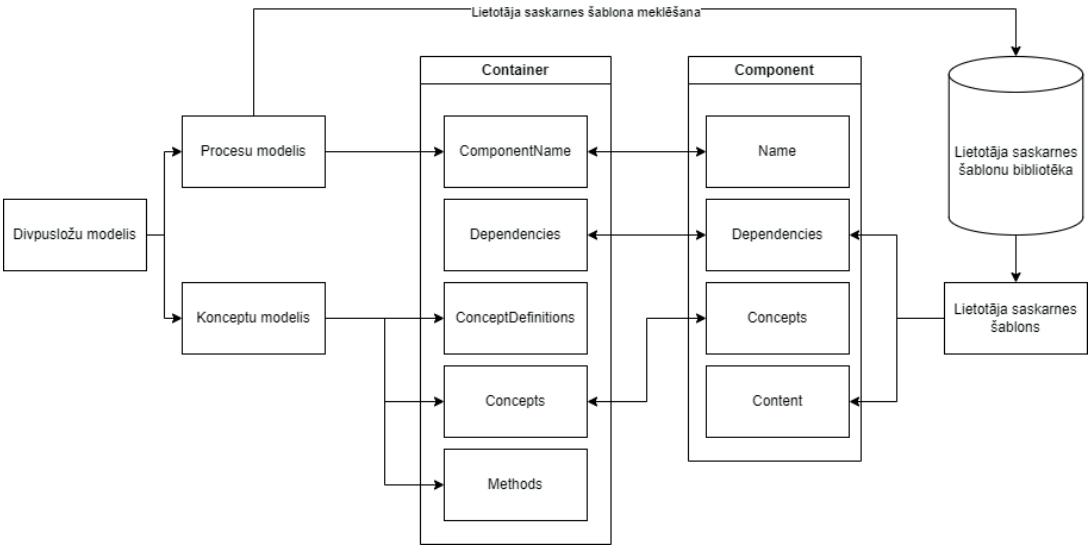
**4.6. tabula**

**Divpusložu modeļa konceptu kartēšana ar React.js elementiem**

| **Divpusložu modeļa koncepts** | **Divpusložu modeļa process** | **React.js Konteiners** | **Mainīgā tips** | **React Router Maršruts** |
|---|---|---|---|---|
| Item | List Items | ListItems | Object | /list_items |
| Item<br>Images | Show Item | ShowItem | Object<br>Array | /show_item |
| Items | Show Cart | ShowCart | Array | /show_cart |
| User<br>Address<br>Items | Show Checkout | ShowCheckout | Object<br>Object<br>Array | /show_checkout |
| User<br>Orders | User profile | ShowProfile | Object<br>Array | /show_profile |
| User | User registration form | RegisterUser | Object | /register_user |

4.11. attēlā ir redzama kopējā transformācijas likumu pielietošanas shēma starp visiem promocijas darba ietvaros izstrādāta risinājuma artefaktiem lietotāja saskarnes prototipa komponentu pirmkoda ģenerēšanai no divpusložu modeļa elementiem.

No divpusložu modeļa tiek izgūti apstrādājamie procesu modelis un konceptu modelis. Katram no šiem modeļiem ir sava loma lietotāja saskarnes prototipu ģenerēšanā. Tā, piemēram, procesu modelis definē konteinera un komponenta nosaukums. Gadījumā, kad ir process ar nosaukumu, piemēram, "*List Items*", tad izmantojot procesa nosaukumu tiek ģenerēts "*ListItemsContainer*" konteiners un "*ListItems*" komponents.




**4.11. att. Divpusložu modeļa loma konteinera un komponenta izveidē**

Tālāk, balstoties uz šo nosaukumu tiek meklēts lietotāja saskarnes šablonu bibliotēkā attiecīgais šablons. Kad tiek atrasts (piemēram, "*List Items Page*") šablons, tad tas izveido komponenta saturu un nosaka nepieciešamās atkarības, kādas ir jādefinē komponentā un konteinerā. Tālāk, no koncepta modeļa tiek izgūti visi pieejamie koncepti un tie tiek kartēti ar konteineru, nosakot tajā nepieciešamos mainīgos un metodes, ja tādas ir.

### 4.2.4. Transformācijas rezultāts

Automātiskās lietotāja saskarnes prototipu izveides kontekstā, modeļu ģenerēšanas rezultāts nozīmē to, kas tiek iegūts no automatizētās procedūras. Abstrakti modeļi tiek pārvērsti faktiskos lietotāja saskarnes prototipos. Transformācijas process pārvērš darbību un mijiedarbību no augsta līmeņa attēlojuma par kaut ko taustāmāku – vizuālās saskarnes, ar kurām lietotāji var mijiedarboties, lai atdarinātu paredzēto lietotāja pieredzi. Modeļu ģenerēšanas rezultāts parasti ietver dažus prototipu objektus, kas norādīti šajos avota modelī. Prototipu izveides galvenais mērķis ir parādīt vizuālu galīgo lietotāja saskarnes dizaina piemēru. Tādā veidā tas ļauj dizaineriem un citiem iesaistītajiem cilvēkiem redzēt, kā sistēmas lietotāja saskarne izskatīsies un darbosies, pirms viņi sāk to pārveidot par gala lietotāja saskarni. Modeļa ģenerēšanas rezultātam var būt daudz

dažādu daļu, piemēram, komponenti, izkārtojumi, navigācijas ceļi un mijiedarbības. Tas parāda, kas ir plānots ievades modeļos, parādot to struktūru vai funkciju atbilstoši prasībām, kas iestatītas modelēšanas posmā. Kad tiek automatizēti šablonu integrēšana prototipu saskarnēs, tiek palīdzēts dizaineriem paātrināt dizaina izmaiņas, pārbaudīt, vai viņi izdara pareizu izvēli attiecībā uz dizainu, un saņemt lietotāju viedokļus agrīnā izstrādes laika skalā, kas palīdz uzlabot gala lietotāja saskarnes produkta kvalitāti un lietojamību.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<patterns>
    <pattern>
        <name>Item Page</name>
        <description>Item details view gor shopping item</description>
        <input>
            <input>List Items</input>
            <input>Show Cart</input>
        </input>
        <output>
            <output>Show Cart</output>
        </output>
    </pattern>
    <pattern>
        <name>Item Details</name>
        <description>Item details with images and description for item</description>
        <parents>
            <parent>Item Page</parent>
        </parents>
    </pattern>
    <pattern>
        <name>Item Description</name>
        <description>Item description for item</description>
        <parents>
            <parent>Item Details</parent>
        </parents>
    </pattern>
    <pattern>
        <name>Item Image Thumbnails</name>
        <description>Item image thumbnails</description>
        <parents>
            <parent>Item Description</parent>
        </parents>
    </pattern>
    <pattern>
        <name>Buttons</name>
        <description>Buttons for shopping cart, checkout and item</description>
        <parents>
            <parent>Item Description</parent>
            <parent>Shopping Cart</parent>
            <parent>Checkout</parent>
        </parents>
    </pattern>
</patterns>
```

**4.12. att. vienkāršota lietotāja saskarnes šablonu bibliotēka**

Autora pētījuma gadījumā kā transformācijas rezultātā tiek iegūti lietotāja saskarnes prototipi, kuri tiek bāzēti uz JavaScript valodas, izmantojot *React.js* bibliotēku un *Material-UI* kā lietotāja saskarnes komponentu satvaru. Transformācijas procesam tiek izveidota lietotāja

saskarnes šablonu bibliotēka, kura glabā sevī norādes uz lietotāja saskarnes šabloniem. 4.12. attēlā ir redzams piemērs kā var definēt lietotāja saskarnes šablonu bibliotēku.

Papildus šīm bibliotēkām tiek izmantots *XState*, kā stāvokļu diagrammu realizācija. Transformācija noris izmantojot PHP programmēšanas valodu, bet tā var tikt aizstāta ar jebkuru citu. Transformācijas procesā tiek izmantotas iepriekš ģenerēti lietotāja saskarnes šabloni, kuri tiek atlasīti izmantojot PHP bibliotēku *TNTSearch* (https://github.com/teamtnt/tntsearch), kura nepieciešama, lai varētu izmantot teksta klasifikācijas metodi lietotāja saskarnes šablonu atlasīšanai. Kad transformācija ir gandrīz pabeigta, tad ir nepieciešams papildināt lietotāja saskarnes prototipus ar datiem, lai lietotāja saskarnes dizaineri vai citas projektā iesaistītās personas varētu pārbaudīt, cik izdevīgs būs ģenerētais prototips no lietotāja pieredzes puses un cik labi attēlojas dati. Šajā gadījumā demonstrācijas datu ģenerēšanai tiek izmantota JavaScript bibliotēka *Faker*, bet to var aizstāt ar jebkuru citu. *Faker* datu kopas tiek ģenerētas izmantojot divpusložu modeļa koncepta diagrammu, pamatojoties uz atribūtiem un to tipiem.

Lietotāja saskarnes šablonu bibliotēka ir saraksts ar pieejamajiem lietotāja saskarnes šabloniem. Ideālā gadījumā pietiek ar šablona nosaukumu un aprakstu, lai varētu iegūt nepieciešamo lietotāja saskarnes šablonu. Papildus pamatelementiem, tiek piedāvāts arī definēt kāds šablonam ir vecākelements "<*parent*>". Tas atvieglo "atomiskā" projektējuma pieeju un palīdz strukturēt lietotāja saskarnes šablonus. Augstākā līmeņa šabloniem – skatījumiem jeb lapām – tiek piedāvāts definēt arī "<*inputs*>" un "<*outputs*>" parametrus, kas sevī satur norādes no kādiem šabloniem ir visprecīzāk nonākt pie esošā šablona un uz kādiem šabloniem ir visefektīvāk pāriet no esošā šablona. Tas veido tādu kā "šablonu karti", kuru var izmantot arī tādā gadījumā, ja tiek papildināts divpusložu modeļa veidošanas rīks *BrainTool* ar iespēju veidojot procesu diagrammu prognozēt nākošos iespējamos šablonus un tādejādi – procesu nosaukumus.

Šī bibliotēka ir nepieciešama, galvenokārt, transformācijas procesā, kurā izmantojot teksta klasificēšanas metodi tiek izgūti attiecīgie lietotāja saskarnes šabloni. Piemēram, izmantojamo *"Buttons*" šablonu (4.13. att.). Šablons tiek definēts XML datnē, kura sastāv no metadatiem – nosaukumu un aprakstu, kas ir vieni no galvenajiem parametriem izvēlē, kā arī no koda daļas. Koda daļa sastāv no 4.7. tabulā redzamajām daļām.

**4.7. tabula**

**Koncepta "Pasūtījums" apraksts**

| | |
|---|---|
| **Dependencies** | No kādām ārējām vai iekšējām bibliotēkām ir atkarīgs kods |
| **Inputs** | Kādus mainīgos vai funkcijas sagaida komponents |
| **Outputs** | Kādus mainīgos vai funkcijas tiek izvadītas |
| **Content** | Šablona koda daļa |

Neskatoties uz lietotāja saskarnes šablona vienkāršumu, tam ir jādefinē dažādi metadati, lai kvalitatīvi varētu savienot šablonus savā starpā.

```
<?xml version="1.0" encoding="UTF-8"?>
<pattern>
    <name>Buttons</name>
    <description>Buttons for shopping cart, checkout and item</description>
    <code>
        <dependencies>
            <dependecy><![CDATA[import Grid from '@mui/material/Grid';]]></dependecy>
            <dependecy><![CDATA[import Button from '@mui/material/Button';]]></dependecy>
        </dependencies>
        <inputs>
            <input>
                <name>backLabel</name>
                <type>variable</type>
            </input>
            <input>
                <name>nextLabel</name>
                <type>variable</type>
            </input>
        </inputs>
        <outputs>
            <output>
                <name>onBackClick</name>
                <type>function</type>
            </output>
            <output>
                <name>onNextClick</name>
                <type>function</type>
            </output>
        </outputs>
        <content>
            <![CDATA[
                <Grid item xs={12} md={12} style={{textAlign: "center", padding: "10px"}}>
                    <Button
                        size="small"
                        onClick={(e) => props.onBackClick(e)}
                    >
                        {backLabel}
                    </Button>
                    <Button
                        size="small"
                        onClick={(e) => props.onNextClick(e)}
                    >
                        {nextLabel}
                    </Button>
                </Grid>
            ]]>
        </content>
    </code>
</pattern>
```

**4.13. att. Lietotāja saskarnes šablons *Buttons***

Lai palaistu transformācijas rezultātu kā tīmekļa lietojumprogrammas priekšgala komponentu, tīmekļa lietojumprogrammas projektā ir jāintegrē ģenerētie lietotāja saskarnes prototipi, kas parasti sastāv no *React.js* komponentiem.

Sakarā ar to, ka izstrādes vide tiek manuāli izveidota un sagatavoti nepieciešamās veidnes *React.js* konteineru un komponentu ģenerēšanai, tad transformācijas procesā šīs datnes tiek izveidotas un papildinātas. Kad šis process ir noticis, izveidoto tīmekļa lietotni var palaist izmantojot tīmekļa pārlūku ar norādīto adresi, kura norāda uz nepieciešamo serveri.

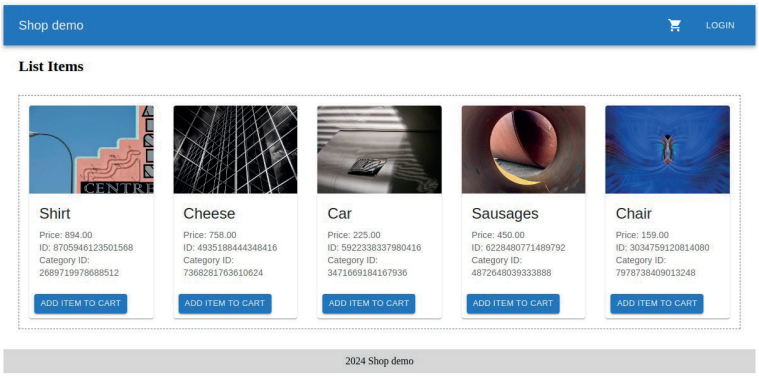
Ja visa augstākminētā procedūra norit sekmīgi, tad iespējams atvērt norādīto adresi un aplūkot iegūto rezultātu. Kā sākuma skatījumam ir jāparādās "*List Items*" (Preču katalogs).

Sadaļa "*List Items*" (4.14. att.). ir pamata daļa, kas izveidota, lai parādītu pieejamos vienumus un palīdzētu lietotājiem tos pārlūkot. Šī sadaļa, kas tiek automātiski izveidota saskaņā ar lietotāja saskarnes šabloniem un dizaina vadlīnijām, garantē vienveidību un lietotājam draudzīgumu visā iepirkšanās saskarnē. Šajā sadaļā tiek rādīti produktu sīktēli, kas sakārtoti režģa vai saraksta skatā, kur katrs sīktēls apzīmē vienu vienumu, kuru varat iegādāties. Lietotāji var pārvietoties pa katalogu un apskatīt preces vienumu vai arī uzreiz to pievienot grozam.

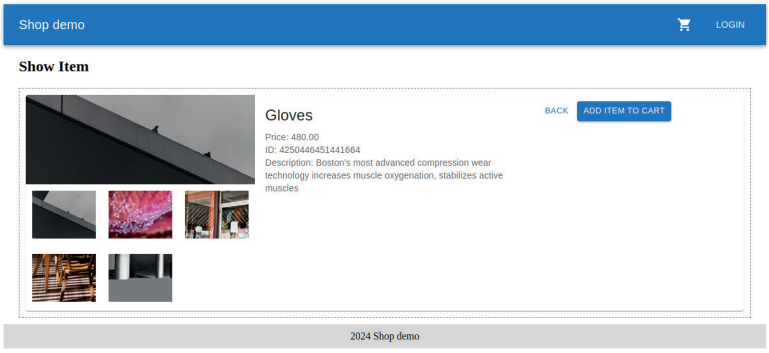
Skatījumā "*Show Item*" (4.15. att.) bieži dēvē par produkta informācijas lapu vai produkta lapu, ir būtiska interneta veikala sastāvdaļa. Tā kalpo kā īpaša vieta, kur lietotāji var izpētīt detalizētu informāciju par konkrētu produktu pirms pirkuma lēmuma pieņemšanas. Produkta informācijas lapā ir redzama produkta bilde kopā ar sīktēlu galeriju, preces apraksts, cena un iespēja to pievienot grozam.

Sadaļā "*Show Cart*" (4.16. att.) dod iespēju gala lietotājam pārskatīt visu grozam pievienoto preču kopsavilkumu. Lietotājs var redzēt šo prežu sarakstu, kurā ir redzamas grozā esošās preces, un tiek piedāvātas izvēles iespējas, piemēram, skatīt detalizētu informāciju par preci vai mainīt tajā esošo preču skaitu vai izņemt preci no groza.

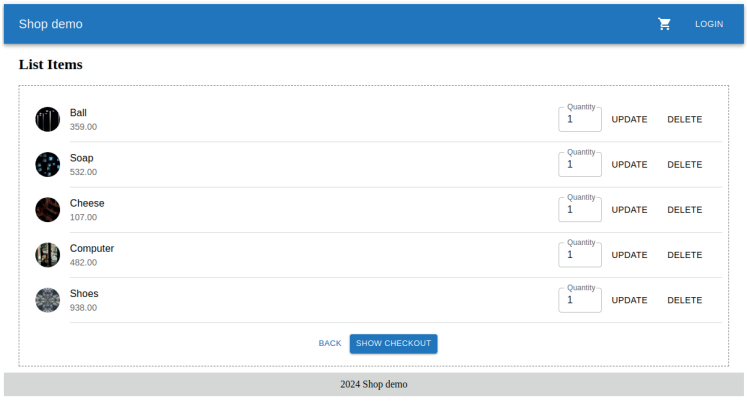
Lietotāja saskarnes prototipu automātiskais ģenerēšanas process nodrošina, ka sadaļa "*Show Cart*" (4.16. att.) atbilst iepriekš izstrādātiem lietotāja saskarnes šabloniem un labākajām metodēm, kādas izstrādātājs var ieviest veidojot šablonus, piemēram, sniedzot cilvēkiem viegli lietojamu pieredzi, orientējoties un rīkojoties ar iepirkumu groza precēm. Rādīt grozu process interneta veikala kontekstā ir paredzēts, lai sniegtu klientiem skaidru un detalizētu priekšstatu par precēm, kuras viņi ir izvēlējušies pirkšanai.




**4.14. att. Uzģenerēta lietotāja saskarnes prototips ar kataloga šablonu**



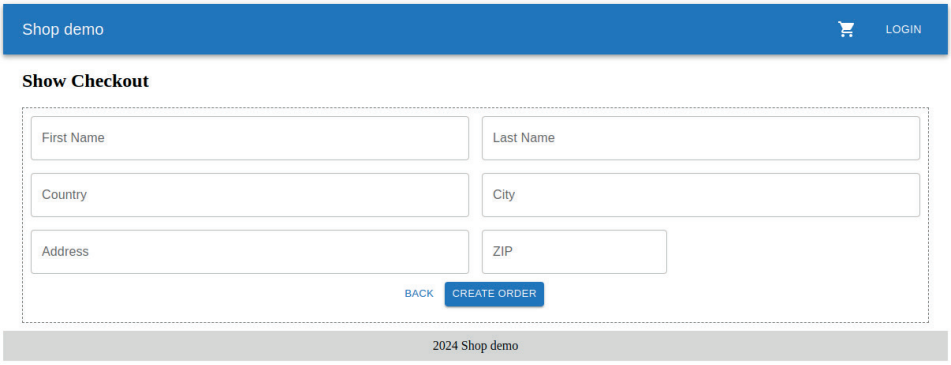

**4.15. attēls Uzģenerēta lietotajā saskarnes prototips ar preču vienuma šablonu**




**4.16. att. Uzģenerēta lietotajā saskarnes prototips ar preču groza šablonu**

Kad klienti piekļūst savam iepirkumu grozam, sistēma izgūst datus par atlasītajiem produktiem, tostarp to nosaukumus, attēlus, cenas, daudzumus un visas piemērojamās atlaides. Pēc tam šī informācija tiek parādīta sakārtotā veidā, parasti saraksta formātā. Katrai grozā esošajai precei ir pievienots tās sīktēla attēls, nosaukums, vienības cena, izvēlētais daudzums. Blakus katrai precei ir interaktīvie elementi, kas ļauj klientiem atjaunināt daudzumu vai pilnībā izņemt preci no groza. Kad klienti pielāgo daudzumus vai dzēš preces, grozs tiek dinamiski atjaunināts, lai atspoguļotu šīs izmaiņas, nodrošinot, ka kopējās izmaksas tiek precīzi aprēķinātas reāllaikā.

Interneta veikala norēķinu sadaļa (4.17. att.) ir pirkšanas procesa pēdējais posms, kurā lietotāji pirms maksājuma veikšanas izskata un apstiprina savus pasūtījumus, norādot piegādes adresi utt. Lietotāji tiek aicināti ievadīt piegādes informāciju, tostarp vārdu, adresi un kontaktinformāciju.




**4.17. att. Uzģenerēta lietotajā saskarnes prototips ar norēķina šablonu**

## 4.3. Transformācijas rezultāta koda kvalitātes pārbaude

Lai veiktu ģenerētā JavaScript pirmkoda kvalitātes pārbaudi var izmantot tādus rīkus kā *JSLint*, *JSHint*, *ESLint* u.c. Tie ir statiskās analīzes rīki, kas brīdina programmatūras izstrādātājus par iespējamām koda kļūdām vai kodēšanas standartu pārkāpumiem. Šādi rīki palīdz atrast kļūdas agrīnā izstrādes stadijā, pirms to labošana kļūst dārgāka (Tomasdottir, Aniche et al., 2020).

*JSLint* ir strikts JavaScript koda pārbaudes rīks, kas nepiedāvā daudz pielāgošanas iespēju un ir izveidots, lai garantētu stingru noteiktu likumu kopu ievērošanu. Tas var būt labi, lai saglabātu

vienveidību lielajās kodu bāzēs. Tomēr tā var nebūt labākā izvēle projektiem, kuriem nepieciešama lielāka pielāgošanās spēja ar kodēšanas standartiem. *JSLint* ar ierobežotajām pielāgošanas iespējām nav tik elastīgs kā pārējie divi rīki – *ESLint* un *JSHint* (Tomasdottir, Aniche et al., 2017; Zhang, Meng, 2020; Tomasdottir, Aniche et al., 2020).

*JSHint* ir mazāk strikts *JSLint* aizstājējs. Tas pieļauj dažas izmaiņas un dublē vairāk *ECMAScript* aspektu nekā *JSLint*. Turklāt, *JSHint* sniedz draudzīgus un skaidrus kļūdu ziņojumus, kas varētu palīdzēt izstrādātāju komandai ātrāk noteikt problēmas. *JSHint*, salīdzinot ar *ESLint*, sākotnējām konfigurācijām ir nedaudz vienkāršāka. Tomēr tas joprojām ir spēcīgs rīks, kas var palīdzēt programmētājiem saglabāt savus viendabīgus un kodus augstā kvalitātē (Tomasdottir, Aniche et al., 2017; Zhang, Meng, 2020; Tomasdottir, Aniche et al., 2020).

*ESLint* ir pazīstams ar savu lielisko pielāgojamības spēju. Programmētāji var izveidot savus likumus un izmantot spraudņus, lai varētu pielāgot pārbaudes procesu atbilstoši savām īpašajām prasībām. Šī funkcija padara ESLint piemērotu projektiem, kuriem ir īpaši kodēšanas standarti vai vajadzības. *ESLint* ir pilnīga savietojamība ar jaunākajām *ECMAScript* funkcijām un sniedz saprotamus kļūdu ziņojumus, padarot izstrādātājiem vieglāk atrodamas un izlabojamas koda kļūdas (Tomasdottir, Aniche et al., 2017; Zhang, Meng, 2020; Tomasdottir, Aniche et al., 2020).

Šajā promocijas darbā automātiski ģenerētais kods ir pārbaudīts uz visiem trijiem augstākminētajiem rīkiem.Automātiskajai lietotāja saskarnes prototipa ģenerēšanai, izmantojot lietotāja saskarnes šablonus, validācijas process pārbauda, cik labs ir ģenerētais kods tādās jomās kā kodēšanas standartu vai likumu ievērošana, modulāritāte un konsekventa darbība. Tas arī pārbauda, vai ģenerētais kods atbilst paraugpraksei, kas atvieglo tā uzturēšanu, vienlaikus saglabājot lasāmību un mērogojamību. Turklāt validācija nozīmē pārbaudi, vai izveidotais kods patiesi ievieš izvēlētos lietotāja sasakarnes šablonus un pareizi atspoguļo plānoto dizainu.

Vispārīgā nozīmē, ja tiek validēts automātisku lietotāja saskarnes prototipu ģenerēšanu, izmantojot šablonus un pēc tam aplūkojot koda kvalitāti, tas palīdz saprast, cik viegli ir uzturēt prototipus, cik tie ir uzticami un vai tos var viegli paplašināt – visi ieteikumi to uzlabošanai, lai izveidotās saskarnes atbilstu paredzamajiem standartiem un prasībām (Becucci, Fantechi et al., 2005).

Ņemot par piemēru "*Show Items*" konteineru, var redzēt uzģenerēto konteinera veidni 4.7. attēlā (Minimālā konteinera veidne), kuru transformējot tiek iegūts pirmkods, kas redzams 4.18.

attēlā (Ģenerētais lietotāja saskarnes "*Show Item*" konteiners). Iegūtā datne pilnībā atbilst uzstādītajai konfigurācijai, jo faktiski tiek apstrādāta ar divpusložu modeļa konceptu modeli, kas ir kartēts ar transformācijas veidnēm.

```
import React from 'react';
import { connect } from 'react-redux';
import { faker } from '@faker-js/faker';
import { goNext, getStateMeta } from "./shared"

import ShowItem from "./ShowItem"


function mapStateToProps(state, props) {
    const currentState = props.stateActor.getSnapshot();
    const meta = getStateMeta(currentState);

    const item = {
        id: faker.number.int(),
        name: faker.commerce.product(),
        price: faker.commerce.price(),
        description: faker.commerce.productDescription()
    }

    const images = new Array(5).fill().map(() => {
        return {
            id: faker.number.int(),
            item_id: item.id,
            name: faker.commerce.product(),
            description: faker.commerce.productDescription(),
            url: faker.image.urlLoremFlickr({ category: 'abstract' }),
            default: faker.datatype.boolean()
        }
    })

    return {
        title: meta.title,
        item: item,
        images: images
    };
}

function mapDispatchToProps(dispatch, props) {
    return {
        goNext: (nextStateName) => {
            return goNext(nextStateName, props.stateActor)
        },
    }
}

export default connect(
    mapStateToProps,
    mapDispatchToProps
)(ShowItem);
```

**4.18. att. Ģenerētais lietotāja saskarnes "*Show Item*" konteinera pirmkods**

Savukārt, iegūtais konteinera komponenta pirmkods ir ģenerēts (4.19. att.) izmantojot minimālo komponenta veidni un atbilst sagaidāmajam. Transformācijas rezultātā var mainīties rindiņu atkāpe, īpaši gadījumos, kad šablonam ir vairāki bērnelementi – citi šabloni.

```jsx
import * as React from "react";
import { useNavigate } from "react-router-dom";

import Box from '@mui/material/Box';
import Card from '@mui/material/Card';

import ItemDetails from "./ItemDetails"
import Buttons from "./Buttons"

export default function ShowItem(props) {
    const navigate = useNavigate();
    const { goNext, title, item, images } = props;

    return (
        <div>
            <h2>{title}</h2>

            <Box sx={{
                flexGrow: 1,
                border: '1px dashed grey',
                marginBottom: "10px",
                padding:"10px"
            }}>
                <Card sx={{ display:"flex", width:"100%" }}>

                    <ItemDetails
                        item={item}
                        images={images}
                    />

                    <Buttons
                        backLabel="Back"
                        nextLabel="Add Item To Cart"
                        onBackClick={() => navigate(goNext("ON_LIST_ITEMS"))}
                        onNextClick={() => navigate(goNext("ON_ADD_ITEM_TO_CART"))}
                    />

                </Card>
            </Box>
        </div>
    )
}
```

**4.19. att. Ģenerētais lietotāja saskarnes "*Show Item*" komponenta pirmkods**

Kā redzams 4.29. attēlā, tad iekļautie komponenti "<*ItemDetails*>" un "<*Buttons*>" atrodas iekļauti kā "*ShowItem*" komponenta bērnelementi, kuri nodrošina vēlamo funkcionalitāti. Piemēram, viens no bērnelementiem "<*Buttons*>" kartēšanas un transformācijas rezultātā iegūst pirmkodu (4.20. att. Ģenerētais lietotāja saskarnes "<*Buttons*>" komponenta pirmkods), kas ir kartēts savā starpā ar vecākelementu "<*ShowItem*>" izmantojot gan konceptu elementus "*backLabel*" un "*nextLabel*", bet arī metodes "*onBackClick*" un "*onNextClick*".

```jsx
import * as React from "react";

import Button from '@mui/material/Button';
import Grid from '@mui/material/Grid';

export default function Buttons(props) {

    const { backLabel, nextLabel } = props

    return (
        <Grid item xs={12} md={12} style={{textAlign: "center", padding: "10px"}}>
            <Button
                size="small"
                onClick={(e) => props.onBackClick(e)}
            >
                {backLabel}
            </Button>
            <Button
                size="small"
                variant="contained"
                onClick={(e) => props.onNextClick(e)}
            >
                {nextLabel}
            </Button>
        </Grid>
    )
}
```

**4.20. att. Ģenerētais lietotāja saskarnes "<*Buttons*>" komponenta pirmkods**

Transformācijas rezultātā iegūtais kods atbilsts veidņu saturam, kas tika izmantots priekš JavaScript pirmkoda ģenerēšanas. Pats kods nesatur biznesa loģiku, tas ir tikai priekš lietotāja saskarnes demonstrēšanas un pārvietošanos pa lapām. Ņemot vērā iegūto rezultātu var secināt, ka koda kvalitāte ir atkarīga, galvenokārt, no izvēlētiem šabloniem un to satura (4.21. att.un 4.22. att.).

```jsx
import * as React from "react";

import Box from '@mui/material/Box';
import CardMedia from '@mui/material/CardMedia';

import ItemImageThumbnails from "./ItemImageThumbnails"
import ItemDescription from "./ItemDescription"

export default function ItemDetails(props) {
    const { item, images } = props;

    return (
        <>
            <Box sx={{ display: 'flex', flexDirection: 'column' }}>
                <CardMedia
                    sx={{ height: 140, width: 360 }}
                    component="img"
                    image={images[0].url}
                />
                <ItemImageThumbnails images={images} />
            </Box>
            <Box sx={{ display: 'flex', flexDirection: 'column' }}>
                <ItemDescription item={item} />
            </Box>
        </>
    )
}
```

**4.21. att. Ģenerētais lietotāja saskarnes "<*ItemDetails*>" komponenta pirmkods**

```
import * as React from "react";

import Typography from '@mui/material/Typography';
import CardContent from '@mui/material/CardContent';

export default function ItemImageThumbnails(props) {
    const { images } = props;

    return (
        <div className="thumbnails" style={{width: "360px"}}>
            {images.map((image, index) => {
                return (
                    <div key={index} style={{display:"inline-block", padding:"10px"}}>
                        <img src={image.url} style={{width:"100px"}} />
                    </div>
                );
            })}
        </div>
    )
}
```

**4.22. att. Ģenerētais lietotāja saskarnes "*<ItemImageThumbnails>*" komponenta pirmkods**

Palaižot uzģenerēto kodu izmantojot *JSLint*, *JSHint* un *ESLint* var konstatēt, ka kodam nav nekādu nopietnu kļūdu un no tā var secināt, ka automātiski ģenerētā koda kvalitāte ir apmierinoša.

Tomēr, ir jāņem vērā, ka, piemēram, *JSLint* ir nepieciešama specifiski konfigurācijas uzstrādījumi, lai tas varētu korekti pārbaudīt *React JSX* datnes. No tā var secināt, ka no augstākminētajiem koda pārbaudes rīkiem, vispiemērotākais turpmākai lietošanai, būtu *ESLint*, kuram ir pieslēgts "*eslint-plugin-react*" spraudnis.

## 4.4. Transformācijas rezultāta pieņemšanas testēšana

Lai veiktu transformācijas rezultāta pieņemšanas testēšanu saskaņā ar programmatūras testēšanas standartu (ISO/IEC/IEEE 29119), pirmkārt ir jādefinē pieņemšanas kritēriji. Šie kritēriji ir jādefinē tā, lai būtu iespējams pārliecināties, ka automātiski ģenerētais kods darbojas un iegūtais tīmekļa lietojums atbilst sagaidāmajam rezultātam vizuālā izskata un satura ziņā un vai iegūtais prototips ļauj lietotājam izpildīt procesu, kas ir definēts procesu modelī. Kritērijiem jābūt tādiem, kurus var novērtēt, piemēram, izpildās process vai neizpildās, vai iegūtais eksrāna izskats (vai saturs) atbilst sagaidāmajam vai nē vai daļēji atbilst.

Pirmais solis pieņemšanas pārbaudē ir katra procesa pieņemšanas kritēriju uzskaitīšana. Šie kritēriji kalpo kā pārbaudes gadījumi, kas tiks izpildīti, lai pārbaudītu funkcionalitāti (Sibal, Kaur et al., 2018). Darbā izstrādātajā risinājumā tiek izveidota divpusložu modeļa procesu tabula (4.9. tabula). Pēc tam šie procesi tiek izpildīti uz automātiski ģenerētā prototipa, lai veiktu

pieņemšanas testēšanu. Ar izpildi tiek saprasta mijiedarbību ar programmatūras sistēmu un apstiprināšanu, ka faktiskie rezultāti atbilst sagaidāmajiem rezultātiem. Pieņemšanas pārbaude tiek veikta no galalietotāja viedokļa, lai nodrošinātu, ka programmatūra ir gatava lietošanai. (Raharjana, Arifin et al., 2023). Pārbaudes gadījumi ir izstrādāti, pamatojoties uz procesu sarakstu vai lietotāju stāstiem un tā tiek veikta vidē, kas ļoti līdzinās produkcijas videi. Kopējais pieņemšanas testēšanas process ir redzams tabulā 4.8.

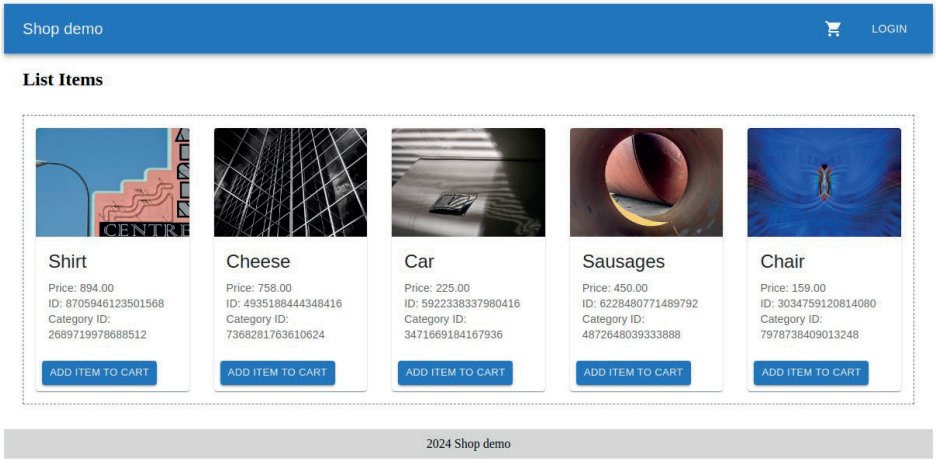
**4.8. tabula**

**Pieņemšanas testēšanas procesa soļi**

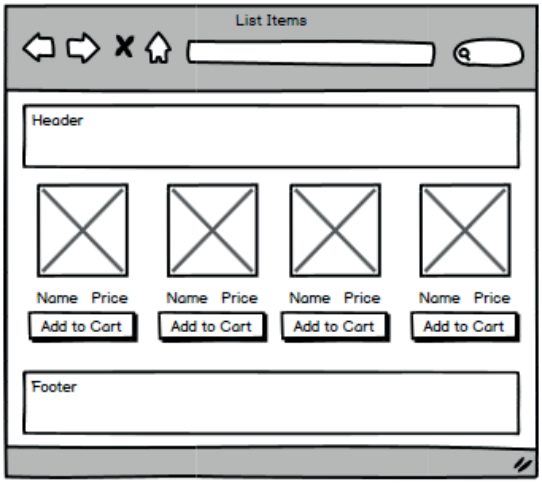
| **Solis** | **Apraksts** |
|---|---|
| Prasību analīze | Analizēt procesu sarakstu (tabula 4.9), lai noteiktu paredzamos rezultātus un pieņemšanas kritērijus |
| Pieņemšanas pārbaudes plāns | Izstrādāt plānu, kurā tiek definētas darbības un kritēriji, kuriem piepildoties var atzīmēt, ka process izpildās vai nē |
| Pārbaudes gadījumu izstrāde | Tiek noteikti ievades un izpildes soļi un sagaidāmie rezultāti, balstīti uz procesu sarakstu (tabula 4.9) |
| Pārbaudes vides uzstādīšana | Tiek sagatavota produkcijs videi līdzīga testēšanas vide, kurā tiek veiktas pārbaudes |
| Pārbaužu veikšana | Veicot pārbaudes tiek atzīmēts vai process izpildās vai neizpildās, tas atbilst sagaidāmajam rezultātam vai nē |
| Rezultātu apkopošana | Tiek pārskatīti rezultāti, lai pārliecinātos, ka ir izpildīti visi pieņemšanas kritēriji un prototips ir gatavs izvietošanai. |

Automātiski ģenerētam lietotāja saskarnes prototipam validācija koncentrējas uz apstiprināšanu, vai izstrādātie prototipi pareizi attēlo noteiktās funkcionālās prasības un lietotāju mijiedarbību. Tas ietver pārbaudi, cik labi darbojas lietotāja saskarnes prototipu komponenti, piemēram, pogas, formas, navigācijas izvēlnes un interaktīvie elementi, lai pārliecinātos, ka tie darbojas, kā sagaidāms. Validācija var nozīmēt arī pārbaudi, vai lietotāju plūsmas un prototipa informācijas struktūra ir loģiska un atbilstoša. Tas ir nepieciešams, lai nodrošinātu, ka cilvēki, kas to izmanto, var konsekventi mijiedarboties ar prototipu. Tas ir arī katra ieviestā procesa pārbaudi,

lai apstiprinātu tā pareizību un uzticamību. Šis process ietver lietotāja mijiedarbības pārbaudi, ievades pareizības apstiprināšanu un datu apstrādi. Mērķis ir pārliecināties, ka prototips dažādās situācijās darbojas tā, kā vajadzētu. Veicot pārbaudi, var redzēt, ka automātiski ģenerētas lietotāja saskarnes prototips (4.23. att.) un uzstādītais sagaidāmais rezultāts (4.24. att.) sakrīt. Galvenokārt, tādēļ, ka automātiski ģenerējamais prototipa kvalitāte ir tieši atkarīga no šablonu bibliotēkas.




**4.23. att. Automātiski ģenerēta lietotāja saskarnes prototips ar kataloga šablonu**




**4.24. att. Sagaidāmais rezultāts kataloga šablonam**

Promocijas darbā procesu prototipa sagaidāmais rezultāts it zīmēts *Balsamiq* rīkā (https://balsamiq.com), kas ir pietiekami vienkāršs lietošanā, bet pietiekami bagāts ar iespējām, lai demonstrētu nepieciešamo struktūru.

Attēlos (4.23. att un 4.24.att) var salīdzināt izkārtojumu un struktūru starp sagaidāmo rezultātu un automātiski ģenerēto prototipu.Var konstatēt, ka izkārojums atbilst sagaidāmajam un elementu (attēli, pogas u.c.) izvietojums sakrīt.

Savukārt, no vizuālās hierarhijas skatījuma var salīdzināt elementu izmēru un fokusu. Relatīvie izmēri ģenerētajā lietotāja saskarnes prototipā atbilst *Balsamiq* skices lielumam, saglabājot paredzēto vizuālo hierarhiju.

Var salīdzināt arī komponentu precizitāti un secināt, ka elementi (attēli, pogas u.c.) sakrīt ar sagaidāmo rezultātu. Redzams, ka automātiski ģenerētajā lietotaja saskarnes prototipā tiek aizstāti, piemēram, attēli, kuri tiek ģenerēti izmantojot demonstrācijas datu ģeneratoru. Jāņem vērā, ka *Balsamiq* veidotais sagaidāmais rezultāts ir zemākas precizitātes nekā automātiski ģenerētais prototips, tādēļ nav korekti salīdzināt funkcionalitāti un gala lietotāja mijiedarbošanos ar ģenerēto prototipu. Šī pieņemšanas daļa tiek deleģēta procesu (4.9. tabula) izpildes pārbaudei.

**4.9. tabula**

**Pieņemšanas kritēriju saraksts**

| **No** | **Skatījums** | **Procesa apraksts** | **Sagaidāmais rezultāts** | **Rezultāta novērtējums** |
|---|---|---|---|---|
| 1 | Sākums | Prototipu iespējams palaist | Prototipu iespējams palaist izstrādes vai demonstrācijas vidē un ar to var mijiedaroboties – pārslēgties starp ekrāniem un sadaļām. | Ir atbilstošs sagaidāmajam |
| 2 | Preču katalogs | Palaižot prototipu, atveras preču saraksts | Atverot preču saraksta skatījumu jeb katalogu, iespējams aplūkot preces, to nosaukumus, cenu. Ir iespēja pievienot izvēlēto preci grozam. | Ir atbilstošs sagaidāmajam |
| 3 | Preču katalogs | Lietotājs var atvērt | Lietotājam ir iespēja atvērt pieslēgšanās ekrānu, noklikšķinot uz attiecīgās | Ir atbilstošs sagaidāmajam |

| No | Skatījums | Procesa apraksts | Sagaidāmais rezultāts | Rezultāta novērtējums |
|---|---|---|---|---|
| | | pieslēgšanās ekrānu | ikonas, kas pārslēdz ekrānu no preču kataloga uz pieslēgšanās ekrānu, kur iespējams ievadīt savu epastu un paroli, lai pieslēgtos kontam. | |
| 4 | Pieslēgšanās skatījums | Lietotājs var pieteikties sistēmā | Lietotājam ievadod pieslēgšanās ekrānā attiecīgajā formā savu epastu un paroli, ekrāns pārslēdzas uz lietotāja profilu | Ir atbilstošs sagaidāmajam |
| 5 | Profila skatījums | Lietotājs var apskatīt savu profilu | Atverot lietotāja profilu ir iespējams apskatīt lietotāja kartiņu, kā arī veikto pasūtījumu sarakstu. | Ir atbilstošs sagaidāmajam |
| 6 | Profila skatījums | Lietotājs var apskatīt savus pasūtījumus | Atverot lietotāja profila sadaļā “Mani pasūtījumi”, lietotājs var detalizētā veidā aplūkot veiktos pasūtījumus. | Ir atbilstošs sagaidāmajam |
| 7 | Preču katalogs | Lietotājs var atvērt preci no preču saraksta | Lietotājam atrodoties preču kataloga sadaļā, var tikt aplūkoti visi preču vienumi un aktivizējot kādu no preču vienumiem, lietotājs tiek pāradresēts uz preču vienuma skatījumu. | Ir atbilstošs sagaidāmajam |
| 8 | Preces vienuma skatījums | Lietotājs var apskatīt informāciju par preci | Izvēloties kādu preci no preču kataloga un aktivizējot to, lietotājs tiek pāradresēts uz preču vienuma skatījumu, kurā ir redzams attēls, preces galerija, preces apraksts un iespēja preces vienumu pievienot preču grozam. | Ir atbilstošs sagaidāmajam |
| 9 | Preču katalogs, preces | Lietotājs var pievienot | Izvēloties noteikto preces vienumu un aktivizējot tā pievienošanu preču grozam, lietotājs tiek pāradresēts uz | Ir atbilstošs sagaidāmajam |

| No | Skatījums | Procesa apraksts | Sagaidāmais rezultāts | Rezultāta novērtējums |
|---|---|---|---|---|
| | vienuma skatījums | izvēlēto preci grozam | preču groza skatījumu, kurā ir redzamas pievienotās preces. Jāuzsver, ka preču pievienošana grozam faktiski nenotiek – lietotājs tiek pāradresēts uz preču groza skatījumu, kurā, savukārt, ir jau definētas preces. | |
| 10 | Groza skatījums | Lietotājs var apskatīt groza saturu | Groza skatījumā lietotājs var redzēt groza preču saturu, kas sastāv no vairākiem vienumiem, kuriem ir iespējams gan mainīt to daudzumu, gan dzēst to no groza. | Ir atbilstošs sagaidāmajam |
| 11 | Groza skatījums | Groza skatījumā lietotājs var mainīt izvēlētās preces daudzumu | Sakarā ar to, ka šis process notiek fonā, tad lietotājs veicot preces daudzuma maiņu, neredz tā faktisko skaita izmaiņu grozā, bet skatījums tiek atgriezts lietotājam tādā formā, kāds tas bija pirms preces daudzuma maiņas, t.i. šajā skatījumā nav ieviesta funkcionalitāte. | *Ir daļēji atbilstošs sagaidāmajam* |
| 12 | Groza skatījums | Groza skatījumā lietotājs var izdzēst izvēlēto preci | Sakarā ar to, ka šis process notiek fonā, tad lietotājs aktivizējot preces dzēšanu, neredz tā faktisko izdzēšanu no groza, bet skatījums tiek atgriezts lietotājam tādā formā, kāds tas bija pirms preces dzēšanas aktvizācijas, t.i. šajā skatījumā nav ieviesta funkcionalitāte. | *Ir daļēji atbilstošs sagaidāmajam* |
| 13 | Groza skatījums | Lietotājs var pāriet uz | Lietotājam atverot preču groza skatījumu ir iespējams pāriet uz norēķinu skatījumu | Ir atbilstošs sagaidāmajam |

| No | Skatījums | Procesa apraksts | Sagaidāmais rezultāts | Rezultāta novērtējums |
|---|---|---|---|---|
| | | norēķinu skatījumu | uzklikšķinot uz pogas “Checkout”. Šajā gadījumā lietotājs tiek pāradresēts uz norēķinu skatījumu. | |
| 14 | Norēķina skatījums | Lietotājs var apskatīt norēķina informāciju | Lietotājam atverot norēķina skatījumu ir redzams norēķina skatījuma forma, kurā var ievadīt nepieciešamos datus. Kad dati ir ievadīti, lietotājs var noklikšķināt uz pogas “Create Order”, lai izveidotu pasūtījumu. | Ir atbilstošs sagaidāmajam |
| 15 | Norēķina skatījums | Lietotājs var pāriet uz pasūtījuma izveides skatījumu | Lietotājam aktivizējot norēķina skatījumā pogu “Create Order”, tiek izveidots pasūtījums un lietotājs tiek pāradresēts uz pasūtījuma kopskata skatījumu. | Ir atbilstošs sagaidāmajam |
| 16 | Pasūtījuma izveides skatījums | Lietotājs var apskatīt informāciju par izveidoto pasūtījumu | Pasūtījuma kopskatā lietotājs var redzēt detalizētu informāciju par sava pasūtījuma izveidošanas datiem, kā arī tālāk pāriet uz citām sadaļām, piemēram, preču katalogu vai citām sadaļām. | Ir atbilstošs sagaidāmajam |

Runājot par reaģējošo dizainu, tad *Balsamiq* rīkā ir iespēja zīmēt skices vai struktūrskices dažādos ekrāna izmēros un ar dažādiem elementiem, kamēr promocijas darbā atbildība par reaģējošo dizainu tiek pārnesta uz lietotāja saskarnes šablonu bibliotēku, kurā tiek paredzēts, ka šabloni tiek veidoti uz kādu no lietotāja saskarnes satvariem – *Material Design*, *Ant Design*, utt., kuri jau paredz iespējamo reaģējošā dizaina funkcionalitāti. Pieņemšanas testēšanas rezultāts rāda, ka no 16 pieņemšanas kritērijiem 14 kritēriji ir apmierināti pilnībā un par dēļēji atbilstošajiem sagaidāmajam rezultātam ir atzīti divi kritēriji. 11.kritērijā groza skatījumā lietotājs, mainot

izvēlētās preces daudzumu, neredz tā faktisko skaita izmaiņu grozā, bet skatījums tiek atgriezts lietotājam tādā formā, kāds tas bija pirms preces daudzuma maiņas. Tas ir saistīts ar to, ka šajā skatījumā nav ieviesta šī procesa funkcionalitāte, kas pēc būtības ir tīmekļa lietojuma loģikas komponents programmatūras aizmugurējā (angl. *back-end*) pusē nevis promocijas darbā piedāvātajā risinājumā ģenerēta lietotāja saskarne, kas ir lietojuma priekšgala (angl. *front-end*) puse. Līdzīgi arī 12.kritērijā, kurā ir sagaidams, ka lietotājs varēs izdzēst izvēlēto preci, izpildes loģika ir ārpus ģenerēšanas risinājuma tvēruma.

## 4.5. Pārējie lietotāja saskarnes pārbaudes aspekti

Papildus var definēt vismaz trīs būtiskus faktorus programmatūras sistēmu projektēšanā un izstrādē. Tie ir lietojamība, veiktspēja un drošība. Lietojamības aspekts nosaka to, cik gala lietotājiem ir vienkārši mijiedarboties ar programmatūru, lai efektīvi veiktu savus uzdevumus. Veiktspēja ir saistīta ar programmas vai lietojumprogrammas darbības ātrumu, kā arī tās reaģētspēju. Savukārt, drošības aspekta mērķis ir aizsargāt sistēmu un tās datus no neapstiprinātas iekļūšanas, ļaunprātīgas izmantošanas un citiem kiberdraudiem. Lai gan visas šīs jomas pašas par sevi ir nozīmīgas, tās ir arī atkarīgas viena no otras un tādēļ tās ir apkopotas vienā apakšnodaļā.

### 4.5.1. Lietojamības aspekti

Lietotāja saskarnes projektēšana ir laikietilpīgs process, kas prasa no dizaineriem daudz pūļu un resursu, lai izveidotu augstas kvalitātes lietotāja saskarni, un lietotāja saskarnes bieži vien projektē vairāki dizaineri. Automātiskā lietotāja saskarnes ģenerēšana novērš nekonsekvences problēmu un izveido lietotāja saskarnes, izmantojot iepriekš definētus šablonus, vienlaikus nodrošinot efektīvu lietotāja saskarni.

Situācijā, kad lietotāja saskarnes prototips tiek automātiski ģenerēts no divupusložu modeļa ar lietotāja saskarnes šablonu bibliotēkas palīdzību, ir ļoti svarīgi ņemt vērā lietojamīgas aspektus. Tie nodrošina, ka ģenerētās saskarnes prototips ne tikai labi darbojas, bet arī ir saprotamas un patīkamas lietotājam. Galvenā šablonu bibliotēku izmantošanas priekšrocība ir tā, ka tās izmantošana nodrošina konsekvenci starp dažādām lietotāja saskarnes sastāvdaļām. Konsekvence

ir svarīgs labas lietojamības elements, jo tā palīdz lietotājiem paredzēt, kā saskarne reaģēs, samazinot viņu mentālo slodzi un padarot lietojumprogrammu vieglāk saprotamu un lietojamu. Izmantojot iebūvētu lietotāja saskarnes šablonu bibliotēku, izstrādātāji var nodrošināt, ka viņu izveidotajā lietotāja saskarnē tiek ievēroti noteikti dizaina noteikumi un principi, kas lietotājiem rada vienotu un saprotamu pieredzi.

Tomēr, lai gan automātiskā lietotāja saskarņu ģenerēšana ir daudzsološa lietojamības uzlabošanai, tā rada arī vairākas problēmas. Viens no lielākajiem izaicinājumiem ir nodrošināt, lai radītās saskarnes varētu pielāgoties konkrētām lietotāju prasībām un situācijām. Tas var prasīt detalizētu izpratni par mērķauditoriju, kas automātiskajām sistēmām var nebūt pieejama. Šablonu bibliotēkas nodrošina labu pamatu, bet var neaptvert visus lietošanas gadījumus un lietotāju scenārijus. Turklāt automatizētajam ģenerēšanasa procesam jāspēj pielāgoties reālām lietotāju reakcijām. Citiem vārdiem sakot, tam jānodrošina iespēja veikt atkārtotus uzlabojumus, pamatojoties uz lietojamības testiem. Šim nolūkam labi jāintegrē atgriezeniskās saites cilpas. Informāciju no lietotāju mijiedarbības var izmantot, lai uzlabotu lietotāja saskarnes modeli un ģenerēšanas procesu. Nobeigumā jāsecina, ka automatizēta lietotāja saskarnes ģenerēšana, izmantojot šablonu bibliotēkas, sniedz būtiskas priekšrocības attiecībā uz lietojamību un konsekvenci, taču, lai nodrošinātu vislabāko iespējamo lietotāja pieredzi, nepieciešama nepārtraukta lietotāju izpēte un iteratīva projektēšanas prakse. Tas jo īpaši attiecas uz ergonomiku.

Ergonomika ir vērsta uz cilvēka labklājības un vispārējās sistēmas veiktspējas optimizēšanu, izstrādājot saskarnes, kas reaģē uz cilvēka spējām un ievēro to ierobežojumus. Šajā kontekstā ergonomika nodrošina, ka izveidotie lietotāja saskarnes prototipi ir ne tikai funkcionāli, bet arī ērti un efektīvi lietotājam. Izmantojot lietotāja saskarņu modeļu bibliotēku, ergonomikas principus var sistemātiski iekļaut automatizētā ģenerēšanas procesā, tādējādi radot saskarnes, kas samazina lietotāja darba slodzi un palielina produktivitāti.

Viens no galvenajiem ergonomiskajiem aspektiem lietotāja saskarnes projektēšanā ir nodrošināt, lai saskarnes elementi būtu intuitīvi organizēti un viegli pārvietojami. Piemēram, bieži izmantoto elementu izvietošana viegli sasniedzamā attālumā samazina vajadzību pēc pārmērīgām kustībām un līdz minimumam samazina atkārtota noguruma traumu risku. Modeļu bibliotēku izmantošana nodrošina konsekventu elementu izvietojumu, pogu izmēru un attālumu starp tām, kā arī veicina lietošanas ērtumu, izmantojot ergonomiskus kritērijus. Šī sistemātiskā pieeja palīdz

izveidot saskarnes, kas ir ne tikai vizuāli pievilcīgas, bet arī fiziski ērtas lietotājiem, lai tās varētu lietot ilgstoši.

Turklāt automatizētās lietotāja saskarnes izveides ergonomika ir pielāgota lietotāju vajadzību un vēlmju daudzveidībai. Dažādiem lietotājiem ir atšķirīgas ergonomiskās prasības atkarībā no tādiem faktoriem kā vecums, fiziskās spējas un ierīces lietošanas paradumi. Piemēram, var integrēt pielāgojamus fontu izmērus, alternatīvas navigācijas iespējas un balss vadības funkcijas, lai pielāgotos lietotājiem ar dažādu veiklības un redzes līmeni. Iekļaujot ergonomiskos apsvērumus automatizētajā ģenerēšanas procesā, dizaineri var nodrošināt, ka izveidotās saskarnes ir visaptverošas un pieejamas, tādējādi palielinot lietotāju apmierinātību un kopējo sistēmas efektivitāti.

### 4.5.2. Veiktspējas aspekti

Automātiskā lietotāja saskarnes prototipa ģenerēšana no divpusložu modeļa, izmantojot lietotāja saskarnes šablonu bibliotēku, nodrošina daudzas uz veiktspēju balstītas priekšrocības. Pirmkārt, tas uzlabo izstrādes efektivitāti, automatizējot ģenerēšanas procesu. Tas samazina laiku un enerģiju, kas nepieciešama, lai manuāli izveidotu dizainu. Standarta modeļu izmantošana garantē vienveidību un optimizāciju, kā rezultātā tiek nodrošināta labāka lietotāja pieredze un veiktspēja, izmantojot pareizi plānotus komponentus, kas ir viegli resursi. Turklāt šai metodei ir iespēja paplašināt un uzturēt lietotāja saskarnes šablonu bibliotēku, kas nodrošina ka var veikt sistemātiskus lietotāja saskarnes prototipu atjauninājumus, veicot mazāk atkārtotu darbu, nodrošinot vienmērīgu veiktspēju, kamēr lietojumprogramma attīstās. Tomēr, šis sistēmas pirmreizēja uzstādīšana var būt sarežģīta un aizņemt daudz laika, jo ir nepieciešams definēt un integrēt visus nepieciešamos šablonus, kas prasa ievērojamas pūles. Pastāv iespēja, ka šāds veids varētu samazināt elastību, apgrūtinot ļoti pielāgotu komponentu ieviešanu. Arī ātrums, kādā darbojas izveidotās lietotāja saskarnes, ir atkarīgs no tā, cik laba ir šablonu bibliotēka. Tas nozīmē, ka vienmēr ir jārūpējas un jāatjaunina lietotāja sakskarnes šablonu bilbiotēka.

#### *4.5.2.1. Efektivitāte ģenerēšanas procesā*

Divpusložu modelis ievērojami uzlabo lietotāja saskarnes prototipu automātisko ģenerēšanu. Tas ir iespējams, ja divpusložu modelis ir apvienots ar labi sakārtotu lietotāja saskarnes

šablonu bibliotēku. Divpusložu modeļa kartēšanas procesa automatizācija ļauj izstrādātājiem izlaist parasti laikietilpīgo manuālās izstrādes darbību, kas paātrina visu izstrādes ciklu. Lietotāja saskarnes šablonu bibliotēka ir kā lietotāja saskarnes elementu noliktava, ko var izmantot atkal un atkal. Tas nodrošina, ka šie elementi ir ērti pieejami, ļaujot tos viegli apvienot jaunos prototipos. Tas samazina replikāciju un veicina efektīvu jaunu dizainu izveidi.

Lai to paveiktu, nepieciešams izprast un definēt prasības – var sākt ar tradicionāli ieinteresēto personu iesaistīšanu, piemēram, dizainerus un izstrādātājus, lai apkopotu lietotāju vajadzības, funkcionālās prasības un tehniskos ierobežojumus. Tālāk nepieciešams atlasīt lietotāja saskarnes šablonu grupas, kas lietojumprogrammā bieži parādās. Šajā kopā ir jāietver tipiski saskarnes komponenti, piemēram, pogas, veidlapas, modālie logi un navigācijas joslas. Izveidojot maksmāli plašu šablonu bibliotēku, kas kartē šos šablonus kopā ar lietošanas gadījumiem un instrukcijām to lietošanai. Padarot visus lietotāja saskarnes šablonus konsekventus, tiek samazinātas atšķirības un sarežģījumi, kas padara ģenerēšanas procesu vieglāk paredzamu un efektīvu.

Tālāk ir nepieciešams realizēt uz komponentiem balstītu arhitektūru, lai lietotāja saskarnes komponenti tiek iesaiņoti atsevišķos un atkārtoti lietojamos moduļos. Katram modulim ir jābūt autonomam, īpaši pārvaldot vienu lietotāja saskarnes apgabalu. Šī metode veicina vairākkārtēju elementu izmantošanu un padara izstrādi un uzturēšanu vienkāršāku. Tomēr, ir svarīgi noteikt vai šablona detaļas atbilst līdzīgiem dizaina noteikumiem un vai tās var bez piepūles savienot savā starpā, lai izveidotu sarežģītas saskarnes.

Lai varētu automētiski ģenerēt lietotāja saskarnes prototipus nepieciešams izveidot vai integrēt lietotāja saskarnes šablonu dzinēju, kas var automatizēt lietotāja saskarnes daļu izkārtojumu atbilstoši šabloniem un lietošanas gadījumiem. Šim dzinējam jāspēj izprast lietotāja konceptuālo modeli (lietotāja prasības) un pārveidot to par tehnisko lietotāja saskarni (ieviešanas modeli). Viens no optimizācijas variantiem būtu mākslīgā intelekta un mašīnmācīšanās algoritmu ieviešana, lai uzlabotu atpazīšanu un dinamisku piemērotu lietotāja saskarnes šablonu izveidi, ko veic dzinējs.

Pie turpmākās optimizācijas – padarīt ģenerēšanas programmas loģiku efektīvāku – var izmant metadatus un veidnes, lai vienkāršotu lietotāja saskarnes šablonu kartēšanu ar koda

komponentiem. Izmantojiet heiristiku vai likumus, kas vada ģenerēšanas procesu, pārliecinoties, ka katram kontekstam ir izvēlēti pareizi šabloni.

#### *4.5.2.2. Konsekvence un optimizācija*

Lietotāja saskarnes prototipu automātiska ģenerēšanā ir svarīga veiktspējas optimizācijas sastāvdaļa. Tas nodrošina, ka izveidotās lietotāja saskarnes prototipi ne tikai darbojas, bet arī darbojas efektīvi. Divpusložu modelis, kad tas ir apvienots ar lietotāja saskarnes šablonu bibliotēku, nodrošina, ka izveidotās lietotāja saskarnes ievēro vienotus dizaina un mijiedarbības noteikumus. Šī vienveidība nodrošina labu veiktspēju, jo standarta komponenti parasti tiek pārbaudīti un izgatavoti ātri, vienlaikus izmantojot resursus. Turklāt iepriekš definētu lietotāja saskarnes šablonu izmantošana ģenerēšanas fāzē palīdz uzlabot veiktspēju, piemēram, ielādes laiku un reaģēšanu. Šī optimizācija pirms gala rezultāta izveides var palīdzēt izvairīties no iespējamiem veiktspējas ierobežojumiem, kas var rasties slikti izstrādātu un nepielāgotu komponentu dēļ.

Konsekvences uzturēšanai ir nepieciešams izveidot pilnīgu testēšanas struktūru, kas ietver vienību testus, integrācijas testus un pilnīgus testus lietotāja saskarnes komponentiem un transformācijas dzinējam. Papildinot vai mainoties lietotāja saskarnes šablonu bibliotēkai ir nepieciešams regulāri veikt automātiskos testus, lai agrīnā stadijā identificētu kļūdas un garantētu ģenerētā lietotāja saskarnes kvalitāti. Kā arī var iekļaut validācijas mehānismus, lai apstiprinātu elementu specifikācijas un funkcionālās prasības ģenerētajās saskarnēs.

Optimizācijai var apkopot un izpētīt informāciju par ģenerētās lietotāja saskarnes prototipa ekspluatāciju, lai noskaidrotu šablonus un elementus, kuras ir jāuzlabo. Šo informāciju var izmantot, lai uzlabotu gan ģenerēšanas loģiku, gan arī lietotāja saskarnes šablonus. Nav izslēgta arī A/B testēšana, kurā tiektu salīdzinātas izgatavotās lietotāja saskarnes variācijas, lai noskaidrotu, kura no tām ir labāk lietojama un patīkama lietotājiem.

Efektīvi var būt arī ieviest veidus, kā apkopot lietotāju atsauksmes, piemēram, aptaujas vai sesijas testēšanai ar reāliem lietotājiem. Izmantojiet rīkus, kas sniedz datus par to, kā cilvēki mijiedarbojas ar ģenerēto prototipu (analītika). Pievēršot uzmanību šīm atsauksmēm un izmantojot tās ģenerētā prototipa uzlabošanai var turpināt uzlabot lietotāja saskarnes prototipa ģenerēšanas procesu, pārliecinoties, ka tas pielāgojas lietotāju prasībām.

#### *4.5.2.3. Mērogojamība un uzturēšana*

Divpusložu modelis palīdz nodrošināt arī tādus veiktspējas aspektus kā mērogojamība un uzturēšana. Kad lietojumprogrammas kļūst lielākas vai mainās, ir ļoti svarīgi saglabāt augstas veiktspējas standartu, atkārtoti neveicot daudz darba. Modeļvadāmā pieeja nodrošina, ka lietotāja saskarnes šablonu izmaiņas var apstrādāt sistemātiski, veicot korekcijas bāzes šablonos, kas pēc tam ir sastopami visā prototipā. Šī pieeja ne tikai atvieglo lietotāja saskarnes prototipa mērogošanu, bet arī saglabā veiktspēju visās tā versijās. Turklāt, izmantojot lietotāja saskarnes šablonu bibliotēku no paša sākuma, var nodrošināt, ka mērogojami dizaina šabloni tiek iebūvēti prototipos. Tas palīdz pārvaldīt veiktspēju, lietojumprogrammai kļūstot lielākai. Gadījumā ja ģenerētais prototips ir liela apjoma, tad to var risināt ar kešatmiņas stratēģijām – samazināt prototipa ielādes laiku un uzlabot veiktspēju.

Lietotāja saskarnes sadalīšana modulāros elemento ir ļoti svarīga mērogojamībai un vienkāršai apkopei. Lietotāja saskarne ir jāsadala atkārtoti lietojamos komponentos atbilstoši izvēlētajiem lietotāja saskarnes šabloniem. Katram komponentam ir jābūt noteiktai funkcionalitātei un dizaina elementiem, kas veicina atkārtotu izmantošanu, vienlaikus atvieglojot atjaunināšanu.

Lietotāja saskarnes ģenerēšanas loģika prasa, lai katrs lietotāja saskarnes šablons atbilstu tā koda veidnēm vai komponentiem. Tas ietver arī metadatu izmantošanu, lai izskaidrotu, kā jāievieš katrs šablons, norādot tādus elementus kā ievades lauki, pogas un izkārtojuma režģi.

Lai saglabātu un paplašinātu risinājumu, ir ļoti svarīgi, lai būtu nepārtrauktas integrācijas un nepārtrauktas izvietošanas (angl. *Continuous Integration and Continuous Delivery/Deployment*, *CI/CD*) prakse. Nepieciešams pārliecināties, ka lietotāja saskarnes šabloniem, kā arī šablonu dzinējam ir pilnīgas automātiskās pārbaudes (testi). Šajos testos jāiekļauj vienības testi, integrācijas testi un pilnīga pārbaude, lai varētu pārliecienātos par to uzticamību. Kad tiek sakārtota *CI/CD* plūsma, tad ir vieglāk ilgtermiņā automatizēt izveides, testēšanas un izvietošanas posmus. Tas nodrošina ātru un uzticamu atjauninājumu veikšanu bez manuālas darbības no šajos procesos iesaistītajām personām. Sasvukārt, ar versiju kontroles sistēmām, piemēram, *GIT*, var monitorēt un pārvaldīt izmaiņas kodā.

#### *4.5.2.4. Ierobežojumi*

Divpusložu modeļa izstrāde un pilnīgas lietotāja saskarnes šablonu bibliotēkas izveidošana var būt sarežģīta, un sākotnēji tas var prasīt daudz laika. Mēģinājums definēt un apvienot visus nepieciešamos šablonus, kā arī nodrošināt to pilnīgu optimizāciju var būt kritisks risinājuma pilna apjoma realizēšanai. Šī sākotnējā sarežģītība varētu būt šķērslis noteiktām komandām, lai to ieviestu savā darba rutīnā un procesos. Lai gan standartizētu lietotāja saskarnes šablonu izmantošana ir izdevīga konsekvences un optimizācijas veicināšanai, tā var arī kavēt elastību. Izstrādātājiem var rasties grūtības iekļaut atšķirīgus vai īpaši pielāgotus komponentus, kas atšķiras no iepriekš izstrādātiem šabloniem. Šis ierobežojums var negatīvi ietekmēt projektus, kuriem nepieciešama liela pielāgošana vai kreatīvs lietotāja saskarnes dizains.

Ģenerēto lietotāja saskarnes prototipu veiktspēja ir lielā mērā saistīta ar to, cik kvalitatīva un pilnīga ir lietotāja saskarnes šablonu bibliotēka. Ja bibliotēkā ir šabloni, kas nav pietiekami optimizēti, lietotāja saskarnei var būt problēmas ar veiktspēju vai tā var neatbilst noteiktām dizaina prasībām. Projektam attīstoties ir nepieciešams sekot līdzi lietotāja saskarnes šablonu bibliotēkai un papildināt to, kaut gan tas var sagādāt papildus piepūli un paaugstināt kļūdu varbūtību. Šādā gadījumā būtu nepieciešams ieviest lietotāja saskarnes šablonu bibliotēkas testēšanu, kas pārbaudītu vai jauniviestās izmaiņas un papildinājumi nenojauc esošos lietotāja saskarnes šablonus.

### 4.5.3. Drošības aspekti

Pirmais solis drošas un privātas lietotāja saskarnes prototipa ģenerēšanai ir definēt projekta drošības un privātuma prasības. Tas nozīmē, ka ir jāsaprot iegūstamo, glabājamo un apstrādājamo datu veids, apjoms un mērķis, kā arī šādas rīcības juridiskās un ētiskās sekas. Nepieciešams arī identificēt iespējamos draudus un riskus, ar kuriem var saskarties lietotāja saskarnes prototups, piemēram, uzlaušana, pikšķerēšana vai datu noplūde. Pamatojoties uz šīm prasībām, ir jānosaka drošības un privātuma mērķi – principi, kas būs izstrādes lēmumu pamatā.

Viens no veidiem, kā samazināt datu noplūdi vai ļaunprātīgas izmantošanas risku, ir ierobežot lietotāja saskarnes prototipa darbības jomu, mērogu vai izmantošanas termiņu. Tas nozīmē, ka prototipā jāiekļauj tikai svarīgākās funkcijas un dati, kas nepieciešami lietotāja

saskarnes prototipa pārbaudei. Nepieciešams izvairīties no reālu vai sensitīvu datu izmantošanu un tā vietā izmantot fiktīvus vai anonimizētus datus (demonstrācijas datus). Turklāt, ja nepieciešams kopīgot ģenerēto prototipu ar komandu vai citām iesaistītajām personām, tad nepieciešams prototipu kopīgot tikai ar cilvēkiem, kuriem tas ir jāredz, un tikai tik ilgi, kamēr ir nepieciešamas šo personu atgriezeniskā saite (princips "*Need-to-know*" (Zwingelberg, 2011)). Prototipiem, pēc savu mērķu sasniegšanas ir jāliedz piekļuve vai arī prototips un visi saistītie dati, ir jāizdzēš. No otras puses lai nodrošinātu lietotāja saskarnes prototipu privātumu un drošību, ir nepieciešams informēt lietotājus un dot tiem iespēju piekrist drošības prasībām. Tas nozīmē, ka iesasitītājām personām ir skaidri jānodod informācija par ģenerētā prototipa mērķi un darbības jomu, kā tiks apkopoti un izmantoti viņu dati un kā aizsargāsit viņu privātumu un drošību. Pirms lietotāja saskarnes prototipu ģenerēšanas uzsākšanas ir nepieciešams lūgt iesaistīto personu atļauja un jādod viņiem iespēja jebkurā laikā atteikties no savu personisko datu izmantošanas, ja tas ir nepieciešams.

Lietotāja saskarnes prototipu izstrādē jāievēro paraugprakse un standarti attiecībā uz privātumu un drošību. Tas ietver drošu protokolu un metožu izmantošanu datu pārraidei un uzglabāšanai, piemēram, *HTTPS*, *SSL* un šifrēšanu. Tas ietver arī integrētās privātuma principu ievērošanu, kas nozīmē, ka privātums un drošība ir jāiekļauj katrā prototipēšanas procesa posmā, sākot no plānošanas līdz novērtēšanai. Turklāt, ir nepieciešams ievērot visus attiecīgos tiesību aktus un noteikumus, piemēram, *GDPR*, *CCPA* vai *HIPAA*, atkarībā no satura un auditorijas. Ievērojot šos principus, var nodrošināt, ka lietotāja saskarnes prototipu ģenerēšana ir ne tikai efektīva, bet arī ētiska un atbildīga.

Nākamais solis droša un privāta lietotāja saskarnes prototipa izveidē ir drošības un privātuma veidņu izmantošana. Šīs veidnes palīdz piemērot drošības un privātuma aspektus, kā arī iepriekš izceltos principus. Drošības un privātuma veidnes ir standarta risinājumi, kas atrisina tipiskas problēmas, kas saistītas ar drošību un privātumu lietotāja saskarnes plānošanā, piemēram, autentifikāciju, autorizāciju, šifrēšanu vai atšifrēšanu, kā arī samazina datu apjomu, vienlaikus saglabājot tos arī anonimizētus. Paroles, biometrijas datu, pilnvaras vai daudzfaktoru autentifikācija ir piemēri, kā padarīt lietotāja saskarnes prototipu drošāku un privātību nodrošinošu. Šifrēšana un atšifrēšana nozīmē informācijas pārveidošanu formāta, kas ir neizlasāms, kad tas tiek sūtīts, piemēram, caur datortīklu. Dažādi algoritmi, piemēram, *AES*, *RSA*, *SSL* tiek izmantoti šifrēšanai un atšifrēšanai, lai novērstu nelegālas piekļuves vai mēģinājumiem veikt modifikāciju.

Manuālas lietotāja saskarnes izveides process ir neaizsargāts pret cilvēku kļūdām, kas var izraisīt aizmirstas drošības problēmas. Automatizētā ģenerēšana garantē, ka katrs lietotāja saskarnes prototips ir izveidots saskaņā ar identiskiem drošības standartiem un rutīnām, samazinot nejaušu drošības kļūdu iespējamību. Automātiskais process var iekļaut drošības pārbaudes un verifikācijas katrā solī. Tādejādi var pārliecināties, ka galvenais produkts atbilst visiem drošības standartiem. Kaut arī šis process ir automatizēts, metodiska pieeja palīdz ātri noteikt un novērst drošības problēmas agrīnās attīstības posmos. Piemēram, iepriekš izveidoto lietotāja saskarņu šabloniem iespējams pārbaudīt drošības nepilnības, lai pārliecinātos, ka tiek samazināti tādi izplatīti riski kā starpvietņu skriptēšana (*XSS*) vai *SQL* injekcija, gadījumā, ja ģenerētajam prototipam tiek pievienots arī datu slānis. Ja viens šablons tiek labots, tas automātiski tiks labots arī visur citur. Ja ir nepilnības atsevišķos šablonos, tad tos var ātri mainīt un tie mainīsies arī visā lietotaja saskarnes prototipā.

Pēdējais solis lietotāja saskarnes prototipā ar drošām un privātām funkcijām ir paša prototipa pārbaude. To var izdarīt, izmantojot reālus vai simulētus lietotājus, kā arī situācijas, lai pārbaudītu prototipa funkcionalitāti, tā lietojamību utt. Šis process palīdz arī noteikt visas drošības vai privātuma problēmas, kas varētu rasties lietotāja mijiedarbības laikā ar lietotāja saskarni. Šajā posmā var tikt izlabotas kļūdas un nepilnības. Testēšana un atsauksmes no lietotājiem ir iespējamas veikt attālināti vai klātienē, ar daudziem rīkiem kuru palīdzību var izmantot, piemēram, *UserTesting*, *UserZoom*, *Lookback* u.c. Droša un privāta lietotāja saskarnes prototipa izveides process var būt sarežģīts. Tam nepieciešama plānošana, pareiza izpilde un pēc tam novērtējums, vai rezultāts atbilst noteiktajiem likumiem. Taču, ievērojot šos principus un darbības, ir iespējams izveidot gan drošu, gan privātu saskarnes prototipu, kas atbilst lietotāju prasībām, vienlaikus ievērojot drošības standartus un noteikumus.

Tomēr, kā jau minēts, ir arī dažas negatīvās puses. Piemēram, liels risks tiek radīts ar atkarību no šablona bibliotēkas drošību. Ja bibliotēkai ir ievainojamības vai to neregulāri atjaunina priekš jauniem drošības riskiem, tad tas tiks pielietots pret visiem ģenerātajiem lietotāja saskarnes prototipiem. Kad saskarnei tiek pievienoti jauni elementi un funkcionalitāte paplašinās, var rasties vairāki uzdevumi – katram jaunam šablonam iespējai būs jāpieraksta automātiski testi un jāveic drošības pārbaudes.

## 4.6. Nodaļas secinājumi un rezultāti

Noslēgumā jāsaka, ka divpusložu modelis piedāvā vērtīgu sistēmu lietotāja saskarņu strukturēšanai. Transformācijas process, ko vada lietotāja saskarnes šabloni un metamodeļi, pārvērš šo modeli konkrētos lietotāja saskarnes komponentos un interaktīvos prototipos. Izmantojot šo metodi, lietotāja saskarnes prototipus var izveidot tā, lai tas labi atbilstu sistēmas pamata arhitektūrai. Rezultātā automātiski ģenerētais lietotāja saskarnes prototips ir funkcionāli spēcīgs, vienlaikus draudzīgs un interesantas lietotājiem, ar kurām mijiedarboties. Kad transformācijas procesus apvieno ar jaunākajām priekšgala tehnoloģijām, piemēram, *React.js*, ir viegli un ātri ieviest lietotāja saskarnes prototipus. Tas paātrina izstrādes laiku un uzlabo vispārējo programmatūras kvalitāti. Divpusložu modelis sniedz izstrādātājiem noderīgu procesu struktūru un konceptu kartēšanai ar priekšgala saskarnēm. Galu galā tas var palīdzēt programmatūras projektiem izstrādāties ātrāk, izpildot lietotāju prasības un biznesa mērķus – mazinātu atšķirības starp tehniskajām funkcijām un galalietotāja pieredzi.

Risinājuma veiktspēja ir lielā mērā atkarīga no tā, cik labi ir izstrādāta un uzturēta šablonu bibliotēka. Lai uzturētu augstu veiktspējas līmeni, ir svarīgi regulāri atjaunināt un optimizēt šablonus, kā arī nākotnē būtu lietderīgi veikt pētījumus par to, kā automatizēt lietotāja saskarnes šablonu bibliotēkas izveidi.

Lai gan darbā tiek runāts par lietotāja saskarnes prototipiem, kas vairāk ir paredzēti iekšējai lietotšanai, tomēr, pieejas realizēšana var notikt dažādos līmeņos. Lai nodrošinātu ilgtermiņa drošību, sistēmai būtu jāiekļauj uzraudzības mehānismi (ievērot attiecīgās drošības regulas un standartus, piemēram, *GDPR*), kas ļauj noteikt un reaģēt uz drošības incidentiem reālajā, piemēram, ietverot brīdinājumu sistēmu, kas informē par aizdomīgām darbībām. Kā arī jāņem vērā, ka automātiskajā lietotāja saskarnes prototipu ģenerācijas procesā var tikt ieviestas drošības ievainojamības – ir svarīgi veikt regulāru drošības testēšanu un validāciju.

Modeļvadāmas pieejas ļauj programmatūras izstrādātājiem koncentrēties uz augsta līmeņa dizaina jautājumiem. Tomēr joprojām tiek diskutēts, piemēram, cik efektīvas būs automatizētās koda ģenerēšanas metodes sarežģītos projektos vai kā šīs metodes darbosies dažādās programmēšanas valodās (Uyanık & Sayar, 2024).

# REZULTĀTI UN SECINĀJUMI

Programmatūras inženierijā lietotāja saskarne kļuva par tā būtisku daļu pēc programmatūras inženierijas krīzes 1969. gadā. Šī situācija atklāja nopietnas programmatūras izstrādes grūtības, piemēram, projektu budžeta pārtēriņu un grūtībām nodot projektus laikā vai vispār tos pabeigt. Problēma ir tajā, ka, programmām kļūstot sarežģītākām un grūtāk apstrādājamām, izstrādātājiem kļūst vitāli svarīgi ne tikai tās izveidot, bet arī nodrošināt lietotājiem veidus, kā efektīvi mijiedarboties ar izstrādātajiem projektiem. Tādējādi labi izveidota lietotāja saskarne ir kļuvusi par vienu no galvenajām atbildēm uz šīm problēmām, uzlabojot lietojamību – atvieglojot sarežģītas programmatūras sistēmas, izmantojot intuitīvus projektēšanas principus. Lietotāja saskarnes dizainam ir nozīmīga loma lietotāju apmierinātības uzlabošanā, apmācību un atbalsta izmaksu samazināšanā, padarot lietotāja saskarnes vieglāk saprotamas un lietojamas. Efektīvs lietotāja saskarnes dizains ar vienkāršiem izstrādes procesiem, samazina lietotāju kļūdas un nodrošina, ka uzdevumus var veikt ātri un precīzi. Veicinot labāku saziņu starp lietotājiem un izstrādātājiem, lietotāja saskarnes projektējums padara vienkāršāku tādas programmatūras izveidi, kas precīzāk atbilst lietotāju prasībām. Turklāt modernās lietotāja saskarnes izstrādes metodes ļauj pielāgoties dažādām vajadzībām, izmantojot atkārtotu uz lietotājiem vērstu izstrādi un dizainu, piemēram, lietotāja saskarnes šablonus. Būtībā, koncentrējoties uz lietotāja saskarnes dizainu, tiek risinātas daudzas problēmas, kas saistītās ar programmatūras inženierijas krīzi.

Vēl viens svarīgs aspekts programmatūras inženierijā ir programmatūras komponenšu ātrā izstrāde, nezaudējot to kvalitāti. Līdz ar to promocijas darbs ir veltīts modeļvadāmās inženierijas pamatkoncepcijās sakņota risinājuma izstrādei, kas dod iespēju automātiski ģenerēt tīmekļa lietotnes priekšgala komponentu pirmkodu, kas no vienas puses paātrina lietotāja saskarnes komponenta izstrādi, un no otrās puses, nodrošina transformācijas rezultāta kvalitāti.

Promocijas darba izstrādes laikā definētie **uzdevumi tiek izpildīti pilnībā**:

1. Tiek izpētītas lietotāja saskarnes izstrādes metodes un paņēmienus ar nolūku identificēt potenciālus pamata elementus un formālismus, ko var izmantot lietotāja saskarnes automātiskajā ģenerēšanā.

2. Tiek apkopota informācija par lietotāja saskarnes elementu daudzveidību ar nolūku definēt lietotāja saskarnes elementu taksonomiju un šablonus, kas kalpos par pamatu mērķa metamodeļa izstrādē.
3. Divpusložu modeļa notācija tiek bagātināta ar transformācijas likumiem nepieciešamajiem elementiem – procesu metadatiem.
4. Tiek izveidots avota un mērķa modeļa saturs, ar nolūku definēt transformācijas likumus, kur par avota modeli tiek izmantots problēmvides divpusložu modelis un par mērķa modeli tiek sagaidīts lietotāja saskarnes prototips šīs problēmvides atbalsta tīmekļa lietotnei.
5. Tiek izstrādāts risinājums lietotāja saskarnes prototipu ģenerēšanai, kas ir demonstrēts uz reālas problēmvides atbalsta tīmekļa lietotnes piemēra, un ir pārbaudīta tā atbilstība sagaidāmajam rezultātam.
6. Tiek veikta risinājuma novērtēšana un izdarīti secinājumu par pētījuma rezultātiem.

Promocijas darba **galvenais rezultāts** ir piedāvātais risinājums, kas nodrošina tīmekļa lietotnes lietotāja saskarnes ģenerēšanu no divpusložu modeļa, izmantojot modeļvadāmas izstrādes pamatkoncepcijas, kas ir metamodelēšana, modeļi un to transformācijas.

Šī promocijas darba ietvaros ir apskatīts lietotāja saskarnes izstrādes svarīgums mūsdienu lietojumprogrammās. Tas aptver dažādus lietotāja saskarnes izstrādes likumus un metodes, uzsverot prasību pēc lietotājiem viegli saprotamām un lietojamām saskarnēm. Tiek pētīti arī pašreizējie stili un tehnoloģijas lietotāja saskarnes izstrādē, uzsverot to ietekmi uz lietotāja pieredzi un programmatūras lietderību.

Tālāk tiek runāts par modeļavadāmās izstrādes potenciālu lietotāja saskarnes izstrādes uzlabošanā. Modeļvadāmas pieejas pamatideja ir izveidot abstraktus modeļus, kurus var automātiski pārveidot izpildāmā kodā. Mērķis ir palielināt efektivitāti, samazināt kļūdas un garantēt vienveidību dažādās programmatūras platformās. Šajā sadaļā ir arī sniegts pamatīgs skaidrojums par rīkiem un struktūrām, kas palīdz modeļvadāmai izstrādei.

Promocijas darbā ir detalizēti aplūkota modeļvadāmās pieejas būtība. Tiek definēti transformācijas likumi, kas transformē augsta līmeņa modeļus reālās implementācijās. Tiek arī sniegts izklāsts par to, kā izveidot šos likumus un pārliecināties, ka tie ir pareizi, elastīgi un var tikt

pielāgoti dažādām lietotāja saskarnes vajadzībām. Tiek parādīti daži transformācijas likumi, to paraugi, kas parāda, kā tie tiek izmantoti praktiskās situācijās. Pie tam tiek arī runāts par grūtībām un galvenajām metodēm šo likumu realizēšanā un apstrādāšanai, lai iegūtu labākos rezultātus.

Pēdējā nodaļa ir veltīta ieteiktā modeļvadāmās lietotāja saskarnes izstrādes risinājuma aprobācijai un validācijai. Tajā ir izskaidrots darbā piedāvātā risinājuma pielietošanas process un veikta tā novērtēšana. Tiek sniegti testu rezultāti, kas parāda, cik precīzs ir piedāvātā risinājuma rezultāts attiecībā uz sagaidāmo. Darba noslēgumā tiek apkopotas visas atziņas, akcentējot risinājuma priekšrocības un iespējamos ierobežojumus. Tas noslēdzas ar ieteikumiem par gaidāmo izpēti un progresu modeļvadāmas lietotāja saskarnes projektēšanas jomā.

Šajā promocijas darbā ir veikts detalizēts pētījums par lietotāja saskarnes izstrādes pilnveidošanu, izmantojot modeļvadāmu metodi. Pētījumā tiek piedāvāts risinājums, kas apvieno abstraktu modelēšanu ar reālu ieviešanu, izmantojot precīzus transformācijas likumus, ar mērķi uzlabot programmatūras lietotāja saskarņu izstrādi, viendabīgumu un lietošanas vienkāršību.

Kopumā par promocijas darba **rezultātiem** var uzskatīt šādus:

1. Tiek apkopota informācija par tīmekļa lietojumu priekšgala programmēšanas satvaros esošajiem elementu klāstiem, kas kalpoja par pamatojumu izveidot lietotāja saskarnes elementu taksonomiju transformācijas likumu mērķa elementiem.
2. Tiek noteikti lietotāja saskarnes metamodeļa parametri un izveidots metamodelis uz ko veikt transformācijas no divpusložu modeļa papildinot procesu ar lietotāja saskarnes šabloniem.
3. Tiek veikta divpusložu modeļa notācijas papildināšana procesu modeļa metadatos, lai marķētu procesus, kuri ir jāpakļauj lietotāja saskarnes izveides procesam un kuri neattiecas uz prototipa izveidi, kā arī tiek piedāvāts ieviest papildus koncepta modeļa elementu atribūtu datu tipus.
4. Tiek definēti transformācijas likumi, kurus pielietojot kopā ar lietotāja saskarnes šablonu bibliotēku un šablonu atlases metodi var tikt palaisti, lai automātiski ģenerētu lietotāja saskarnes prototipus.
5. Tiek piedāvāts risinājums automātiskajai lietotāja saskarnes ģenerēšanai, kas sastāv no komponentiem saskaņā ar modeļvadāmās inženierijas pamatprincipiem.

6. Tiek demonstrētas piedāvāta risinājuma praktiskās pielietošanas iespējas, izmantojot to programmatūras sistēmas priekšgala komponentu izveidošanai.
7. Tiek ieskicētas divpusložu modeļa atbalsta rīka *BrainTool* attīstīšanas un bagātināšanas iespējas ar tādām funkcijām, kā lietotāja saskarnes šabloniem, procesu izvēles prognozēšanu, saistot to ar lietotāja saskarnes šablonu bibliotēkas elementu saiknēm, utt.

Paveiktais darbs un iegūtie rezultāti dod pamatojumu apgalvot, ka darba ievadā definētas tēzes ir apstiprinātas:

1. No RTU izstrādātā divpusložu modeļa ir iespējams ģenerēt tīmekļa lietotnes lietotāja saskarnes prototipus, izmantojot modeļvadāmas izstrādes pamatkoncepcijas, kas ir modeļi un modeļu transformācijas – par to liecina izstrādātais risinājums un tā demonstrācija uz aprobācijas piemēra.
2. Tīmekļa lietotnes lietotāja saskarnes prototipa automātiskā iegūšana no biznesa līmeņa modeļiem paātrina tīmekļa lietotnes izstrādes procesu, nezaudējot rezultāta kvalitāti. – par to liecina transformācijas rezultāta analīze gan iegūta koda kvalitātes ziņā, gan iegūta tīmekļa lietojuma atbilstība problēmvides pieņemšanas kritērijiem.

Balstoties uz promocijas darba ietvaros veiktiem pētījumiem un iegūtajiem rezultātiem, ir izdarīti šādi **secinājumi**:

1. Šobrīd modeļvadāmajā inženierijā ir ierasts izmantot UML valodu artefaktu definēšanā, tomēr tā ir vairāk gatavu risinājumu aprakstam un var būt sarežģīti izprotama problēmvides ekspertiem. Tomēr, veicot pētījumu, var secināt, ka divpusložu modelis ir vairāk piemērots lietotāja saskarnes prototipu ģenerēšanai.
2. Divpusložu modeļiem ir jābūt kvalitatīviem un strikti definētiem kā arī lietotāja saskarnes šabloniem un to bibliotēkai ir jābūt kvalitatīvām. Savukārt pats šablonu kods ir pareizi jāuzraksta, lai būtu iespējams ģenerēt lietotāja saskarnes prototipus.
3. Divpusložu modeļa notācija gandrīz pilnībā nodrošina lietotāja saskarnes ģenerēšanu, tomēr ir nepieciešams papildināt notāciju – koncepta modelī ar papildus lauku tipa vērtībām un procesu modelī – ar to vai konkrēts process ir lietotājam redzams vai nē.

4. Var papildināt BrainTool ar lietotāja saskarnes šabloniem un procesu prognozēšanu. Piemēram, izveidojot jaunu procesu tiktu piedāvāts automātiski izveidot tam sekojošus procesus, ja šāda informācija glabātos lietotāja saskarnes šablonu bibliotēkā.

**Praktiskā nozīme.** Izstrādāto risinājumu ir ieteikts pielietot tīmekļa lietotņu lietotāja saskarņu izstrādei, automātiski ģenerējot priekšgala komponentu pirmkodu prototipiem. Šīs pieejas mērķis ir risināt programmatūras izstrādes izplatītas problēmas, piemēram, projektu budžeta pārtēriņu un projektu realizēšanas termiņu problēmas, izmantojot modeļvadāmās inženierijas principus, lai uzlabotu izstrādes efektivitāti un nodrošinātu tās kvalitāti.

Definējot mērķa metamodeli un transformācijas likumus, risinājums ļauj automātiski ģenerēt lietotāja saskarnes prototipus no augsta līmeņa abstraktiem modeļiem. Tas samazina manuālās programmēšanas piepūli, paātrina izstrādes procesu un samazina cilvēku kļūdas.

Standartizētu šablonu un transformēšanas likumu izmantošana nodrošina, ka ģenerētās lietotāja saskarnes ir konsekventas dizainā un atbilst paraugpraksei, tādējādi nodrošinot augstāku kvalitāti un vieglāk uzturamu kodu.

Darbā piedāvātā risinājuma mērķis ir racionalizēt lietotāja saskarnes prototipu izstrādes procesu, padarot to ātrāku, efektīvāku un spējīgu radīt augstas kvalitātes, uz lietotāju orientētas saskarnes. Šī pieeja ne tikai risina dažus ieilgušos programmatūras inženierijas izaicinājumus, bet arī veido pamatu turpmākiem sasniegumiem uz modeļiem balstītā lietotāja saskarnes izstrādē.

Modificējot esošo divpusložu modeļa notāciju un adaptējot to transformācijām lietotāja saskarnes prototipu pirmkodā, promocijas darbā izstrādātais prototipu ģenerēšanas algoritms pēc tā implementācijas attiecīgajā atbalsta rīkā var būt ieviests industrijā, jo algoritms dot iespēju no problēmvides ekspertiem saprotamā modeļa iegūt lietotāja saskarnes prototipu pirmkodu.

Kā arī pētījuma gaitā definētie spriedumi un iegūtie rezultāti var būt interesanti plaša loka speciālistiem gan industrijā, gan turpinot zinātniskos pētījumus modeļvadāmās programmatūras jomā.

**Turpmākais virziens pētījuma jomas attīstīšanai**

Promocijas darba ietvaros izstrādātais risinājums nodrošina programkoda inženierijas uzdevuma risināšanu vienā virzienā no problēmvides modeļa uz kodu. Piedāvāta risinājuma

arhitektūra ļauj pētīt arī koda reinženierijas problēmas risināšanu, kas ir nodrošināt iespēju uzturēt kodā veiktās izmaiņas attiecībā uz modeļa saturu. Šī promocijas darba ietvaros koda reinženierijas problēmas risināšana ir ārpus pētījuma tvēruma.

Vēl viens nākotnes pētījuma virziens šim darbam būtu esošā rīka, *BrainTool*, papildināšana ar darbā apskatīto pieeju vai arī jauna redaktora izveide promocijas darba pieejas realizēšanai lietotājam draudzīgā vidē. *BrainTool* rīku varētu papildināt ar iespēju lietotājam, veidojot divpusložu modeļa procesus, automātiski piedāvāt kādi būtu nākošie procesi un attiecīgi – lietotāja saskarnes šabloni. Papildus, varētu papildināt lietotāja saskarnes šablonu sarakstu ar definīcijām un dažādiem lietotāju saskarnes satvariem, lai varētu ģenerēt prototipus plašākā tehnoloģiju spektrā.

Nozīmīgāks pētījums būtu izmantot mākslīgā intelekta piedāvātās iespējas, lai automātiski izveidotu un savienotu savā starpā nepieciešamos šablonus no lietotāja saskarnes bibliotēkas aprakstu, tādejādi samazinot manuālu darbu lietotāja saskarnes šablonu izstrādei.

Ņemot vērā, ka šobrīd tiek plaši attīstīti dažādi lietotāja saskarnes satvari, tādi kā *Material Design*, *Ant Design*, utt., tad būtu vērtīgi veikt pētījumus, kā var izmantot šo satvaru semantiku un citas pieejas, lai automātiski izveidotu vēlamos lietotāja saskarnes elementus tikai norādot kādu lietotāja saskarnes satvaru izmantot.

# IZMANTOTIE INFORMĀCIJAS AVOTI


Adefris B. B., Habtie A. B., Taye Y. G. (2022) Automatic Code Generation From Low Fidelity Graphical User Interface Sketches Using Deep Learning, 2022 International Conference on Information and Communication Technology for Development for Africa (ICT4DA), Bahir Dar, Ethiopia, 2022, pp. 1-6, DOI: https://doi.org/10.1109/ICT4DA56482.2022.9971204

Aizpurua A., Harper S., Vigo M. (2016) Exploring the relationship between web accessibility and user experience. International Journal of Human-Computer Studies. 91. DOI: https://doi.org/10.1016/j.ijhcs.2016.03.008

Akiki P., Bandara A., Yu Y. (2014) Adaptive Model-Driven User Interface Development Systems. ACM Comput. Surv. 47, 1, Article 9 (July 2014), 33 pages. DOI: https://doi.org/10.1145/2597999

Alamin M. A., Malakar S., Uddin G., Afroz S., Haider T. B. Iqbal A. (2021) An empirical study of developer discussions on low-code software development challenges. In 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR) (pp. 46–57). https://doi.org/10.1109/MSR52588.2021.00018

Alarcon J., Balcazar I., Collazos CA., Luna H., Moreira F. (2022) User Interface Design Patterns for Infotainment Systems Based on Driver Distraction: A Colombian Case Study. Sustainability. 2022; 14(13):8186. DOI: https://doi.org/10.3390/su14138186

Alexander C. (1979) The timeless way of building, vol 1. Oxford University Press, New York, ISBN: 978-0195024029

Alexander C., Ishikawa S., Silverstein M. (1977) A Pattern Language: Towns, Buildings, Construction. Berkeley, CA, USA: Oxford University Press, 1977 ISBN: 978-0195019193

Alfaridzi M. D., Yulianti L. P. (2020) UI-UX Design and Analysis of Local Medicine and Medication Mobile-based Apps using Task-Centered Design Process, 2020 International Conference on Information Technology Systems and Innovation (ICITSI), 2020, pp. 443-450, DOI: https://doi.org/10.1109/ICITSI50517.2020.9264947

Almendros-Jiménez J. M., Iribarne L. (2005) Designing GUI components from UML use cases, 12th IEEE International Conference and Workshops on the Engineering of Computer-

Based Systems (ECBS'05), Greenbelt, MD, USA, 2005, pp. 210-217, DOI: https://doi.org/10.1109/ECBS.2005.31

Almendros-Jiménez J. M., Iribarne L. (2008) An extension of UML for the modeling of WIMP user interfaces, Journal of Visual Languages & Computing, Volume 19, Issue 6, 2008, Pages 695-720, ISSN 1045-926X, DOI: https://doi.org/10.1016/j.jvlc.2007.12.004

Al-Saqqa S., Sawalha S., Abdel-Nabi H. (2020) Agile Software Development: Methodologies and Trends. International Journal of Interactive Mobile Technologies (iJIM). 14. 246. DOI: https://doi.org/10.3991/ijim.v14i11.13269

Ammar L. B. (2021) An Automated Model-Based Approach for Developing Mobile User Interfaces, in IEEE Access, vol. 9, pp. 51573-51581, 2021, DOI: https://doi.org/10.1109/ACCESS.2021.3066007

Andrade A., Vilar R., Lima A., Almeida H., Perkusich A. (2017) Analyzing duplication on code generated by Scaffolding frameworks for Graphical user interfaces. 511-516. https://doi.org/10.18293/SEKE2017-164

Antović I., Vlajić S., Milić M., Savic D. (2012) Model and software tool for automatic generation of user interface based on use case and data model. IET Software. 6. DOI: https://doi.org/10.1049/iet-sen.2011.0060

Arrhioui K., Mbarki S., Betari O., Roubi S., Erramdani M. (2017) A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications. Transactions on Machine Learning and Artificial Intelligence. 5. DOI: https://doi.org/10.14738/tmlai.54.3189

Aspray W., Keil R., Parnas D. (1999) History of Software Engineering

Babris K., Nikiforova O. (2024a) Towards Automated UI Mockup Generation from Two-Hemisphere Problem Domain Models: A Conceptual Framework and Approach, Proceedings of 19th Iberian Conference on Information Systems and Technologies (CISTI 2024, June 25-28), 2024 (accepted for publication) (SCOPUS)

Babris K., Nikiforova O. (2024b) From Models to Interfaces: Leveraging the Two-Hemisphere Model for Automated UI Generation, 2024 IEEE 65th International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS), Riga, Latvia, 2024, pp. 1-6, submitted for publication (SCOPUS)

Babris K., Nikiforova O., Sukovskis U. (2019) Brief Overview of Modelling Methods, Life-Cycle and Application Domains of Cyber-Physical Systems. Applied Computer Systems, 2019, Vol. 24, Issue 1, pp. 1.-8., DOI: https://doi.org/10.2478/acss-2019-0001 (Web of Science)

Bajammal M., Davood M., Ali M. (2018) Generating reusable web components from mockups. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18). Association for Computing Machinery, New York, NY, USA, 601–611. DOI: https://doi.org/10.1145/3238147.3238194

Bajovs A., Nikiforova O., Sējāns J. (2013) Code Generation from UML Model: State of the Art and Practical Implications. Scientific Journal of Applied Computer Systems, 14, 7-18, DOI https://doi.org/10.2478/acss-2013-0002

Batdalov R., Nikiforova O. (2021) Patterns for Assignment and Passing Objects Between Contexts in Programming Languages. In: Proceedings of the European Conference on Pattern Languages of Programs (EuroPLoP'21), Austria, Graz, 7-11 July, 2021. New York: ACM, 2021, pp.14:1-14:9. SBN 978-1-4503-8997-6.
DOI: https://doi.org/10.1145/3489449.3489975

Bäumer D. et al. (1996) User interface prototyping - concepts, tools, and experience. Proceedings of the 18th international conference on Software engineering, (March), pp.532–541. DOI: https://doi.org/10.1109/ICSE.1996.493447

Becucci M., Fantechi A., Giromini M., Spinicci E. (2005) A comparison between handwritten and automatic generation of C code from SDL using static analysis. Softw: Pract. Exper., 35: 1317-1347. DOI: https://doi-org.resursi.rtu.lv/10.1002/spe.673

Beek (ter) M.H., McIver A. (2021) Formal methods: practical applications and foundations. Form Methods Syst Des 58, 1–4 (2021). DOI: https://doi.org/10.1007/s10703-021-00380-6

Bell R. (1998) Code Generation from Object Models, Embedded Programming Systems Journal, March, 1998

Beltramelli T. (2018) Pix2code: Generating Code from a Graphical User Interface Screenshot. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '18). Association for Computing Machinery, New York, NY, USA, Article 3, 1–6. DOI: https://doi.org/10.1145/3220134.3220135

Beranic T., Rek P., Hericko M. (2020) Adoption and Usability of Low-Code/ No-Code Development Tools. Proceedings of the Central European Conference on Information and Intelligent Systems, Varaždin, 97-103.

Bingbing X. (2012) A Mapping Method of Using the Compound Use Cases to Generate User Interface. Open Journal of Applied Sciences. 2. DOI: http://doi.org/10.4236/ojapps.2012.23026

Blomqvist E. (2008) Pattern ranking for semi-automatic ontology construction. In Proceedings of the 2008 ACM symposium on Applied computing (SAC '08). Association for Computing Machinery, New York, NY, USA, 2248–2255.
DOI: https://doi.org/10.1145/1363686.1364224

Boduch A. (2019) React Material-UI Cookbook: Build captivating user experiences using React and Material-UI. Packt Publishing Ltd, 2019. ISBN: 978-1789615227

Borchers J. O. (2000) A pattern approach to interaction design. In Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '00). Association for Computing Machinery, New York, NY, USA, 369–378. DOI: https://doi.org/10.1145/347642.347795

Brambilla M., Umuhoza E., Acerbis R. (2017) Model-driven development of user interfaces for IoT systems via domain-specific components and patterns. J Internet Serv Appl 8, 14 (2017). DOI: https://doi.org/10.1186/s13174-017-0064-1

Briand L. C., Labiche Y., Lin Q. (2005) Improving statechart testing criteria using data flow information, 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), Chicago, IL, USA, 2005, pp. 10 pp.-104,
DOI: https://doi.org/10.1109/ISSRE.2005.24

Cabot J. (2020) Positioning of the low-code movement within the field of model-driven engineering. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '20). Association for Computing Machinery, New York, NY, USA, Article 76, 1–3. https://doi.org/10.1145/3417990.3420210

Cai Y., Mao S., Wu W., Wang Z., Liang Y., ge T., Wu C., You W., Song T., Xia Y., Tien J., Duan N. (2023) Low-code LLM: Visual Programming over LLMs. DOI: https://arxiv.org/abs/2304.08103

Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J. (2003) A Unifying Reference Framework for multi-target user interfaces, Interacting with Computers, Volume 15, Issue 3, June 2003, Pages 289–308, DOI: https://doi.org/10.1016/S0953-5438(03)00010-9

Cao R. K., Liu X. (2020) IFML-Based Web Application Modeling, Procedia Computer Science, Volume 166, 2020, Pages 129-133, ISSN 1877-0509, DOI: https://doi.org/10.1016/j.procs.2020.02.034

Chen W., Lin Y., Yen T., Peng S., Lin Y. (2022) DeviceTalk: A nocode low-code IoT device code generation. Sensors, 22(13), 4942. DOI: https://doi.org/10.3390/s22134942

Cimino M. G. C. A., Marcelloni F. (2012) An efficient model-based methodology for developing device-independent mobile applications. J. Syst. Archit. 58, 8 (September, 2012), 286–304. DOI: https://doi.org/10.1016/j.sysarc.2012.06.001

Conchúir E. O., Ågerfalk P. J., Olsson H. H., Fitzgerald B. (2009) Global software development: Where are the benefits? Communications of the ACM, 52(8), 127–131. https://doi.org/10.1145/1536616.1536648

Coyette A., Vanderdonckt J., Limbourg Q. (2007) SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping. In: Guelfi, N., Buchs, D. (eds) Rapid Integration of Software Engineering Techniques. RISE 2006. Lecture Notes in Computer Science, vol 4401. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-540-71876-5_11

Cruz (da) R., Miguel A., João F. (2010a) Automatic Generation of User Interface Models and Prototypes from Domain and Use Case Models. DOI: https://doi.org/10.5772/9498

Cruz (da) R., Miguel A., João F. (2010b) A Metamodel-Based Approach for Automatic User Interface Generation. 6394. DOI: https://doi.org/10.1007/978-3-642-16145-2_18

Diehl C., Martins A., Almeida A., Silva T., Ribeiro Ó., Santinha G., Rocha N., Silva AG. (2022) Defining Recommendations to Guide User Interface Design: Multimethod Approach. JMIR Hum Factors. 2022 Sep 30;9(3):e37894. DOI: http://doi.org/10.2196/37894

Domingo A., Echeverría J., Pastor O., Cetina C. (2020) Evaluating the Benefits of Model-Driven Development: Empirical Evaluation Paper. DOI: https://doi.org/10.1007/978-3-030-49435-3_22

Duan P., Hartmann B., Nguyen K., Li Y., Hearst M., Morris M.R. (2023) Towards Semantically-Aware UI Design Tools: Design, Implementation, and Evaluation of Semantic Grouping Guidelines. ICML 2023 Workshop on Artificial Intelligence and Human-Computer Interaction.

Dubey G. (2011) A Survey on Guiding Logic for Automatic User Interface Generation. In: Stephanidis, C. (eds) Universal Access in Human-Computer Interaction. Design for All and eInclusion. UAHCI 2011. Lecture Notes in Computer Science, vol 6765. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-21672-5_40

Elec T., Uyanik B., Şahin V. (2020) A template-based code generator for web applications. Turkish Journal of Electrical Engineering and Computer Sciences. 28. 1747 – 1762. DOI: https://doi.org/10.3906/elk-1910-44

Elkoutbi M., Keller R.K. (2000) User Interface Prototyping Based on UML Scenarios and High-Level Petri Nets. In: Nielsen, M., Simpson, D. (eds) Application and Theory of Petri Nets 2000. ICATPN 2000. Lecture Notes in Computer Science, vol 1825. Springer, Berlin, Heidelberg. DOI: https://doi-org.resursi.rtu.lv/10.1007/3-540-44988-4_11

Elkoutbi M., Khriss I., Keller R. K. (1999) Generating user interface prototypes from scenarios, Proceedings IEEE International Symposium on Requirements Engineering (Cat. No.PR00188), Limerick, Ireland, 1999, pp. 150-158, DOI: https://doi.org/10.1109/ISRE.1999.777995

Elkoutbi M., Khriss I., Keller R.K. (2006) Automated Prototyping of User Interfaces Based on UML Scenarios. Autom Software Eng 13, 5–40 (2006). DOI: https://doi.org/10.1007/s10515-006-5465-5

Elkoutbi M., Khriss I., Keller R. (2009) User Interface Prototyping using UML Specifications, 2009, available at:https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6c8d8240205c8fcbc6500f5159e12788d5d01481

Elrom E. (2021) React Router and Material-UI. In: React and Libraries. Apress, Berkeley, CA. DOI: https://doi.org/10.1007/978-1-4842-6696-0_4

Engel J., Herdin C., Märtin C. (2012) Exploiting HCI pattern collections for user interface generation. In Proc. of 4th Int. Conf. on Pervasive Patterns and Applications (Nice, July 22-27, 2012). PATTERNS'2012. Int. Acad., Research, and Industry Assoc., Wilmington, 36–44

Eriksson A., Lindström B., Offutt J. (2013) Transformation Rules for Platform Independent Testing: An Empirical Study, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg, 2013, pp. 202-211, DOI: https://doi.org/10.1109/ICST.2013.28

Escalona M.J., García-Borgoñón, L., Koch, N. (2021) Don't Throw your Software Prototypes Away. Reuse them!

Falb J., Popp R., Rock T., Jelinek H., Arnautovic E., Kaindl H. (2007) Fully-automatic generation of user interfaces for multiple devices from a high-level model based on communicative acts, 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07), Waikoloa, HI, USA, 2007, pp. 26-26, DOI: https://doi.org/10.1109/HICSS.2007.236

Fernández E., Liu Y., Pan R. (2001) Patterns for Internet shops. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8ad2c6f226600828dab4e5317ff713603e003383

Ferreira J., Restivo A., Ferreira H. (2021) Automatically Generating Websites from Hand-drawn Mockups. In Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP, ISBN 978-989-758-488-6 ISSN 2184-4321, pages 48-58. DOI: http://doi.org/10.5220/0010193600480058

Fincher S., Finlay J., Greene S., Jones L., Matchen P., Thomas J., Molina P. (2003) Perspectives on HCI patterns: concepts and tools. 1044-1045. DOI: https://doi.org/10.1145/765891.766140

Forbrig P., Dittmar A., Reichart D., Sinnig D. (2004) From Models to Interactive Systems Tool Support and XIML. Available:

https://www.researchgate.net/publication/220726900_From_Models_to_Interactive_Systems_Tool_Support_and_XIML

Frost B. (2016) Atomic design, ISBN: 978-0998296609, 2016

Fung H. (2019) The Successful Software Manager, ISBN: 9781789615531, Packt Publishing, 2019

Gao D. (2022) Measuring developers' episodic experience of low-code development platforms [Maģistra darbs, Aalto universitāte] https://aaltodoc.aalto.fi/handle/1234567 89/117387.

Gharaat M., Sharbaf M., Zamani B. et al. (2021) ALBA: a model-driven framework for the automatic generation of android location-based apps. Autom Softw Eng 28, 2 (2021). DOI: https://doi.org/10.1007/s10515-020-00278-3

Gilbert D., Fischer H., Röder D. (2021) UX at the Right Level: Appropriately Plan the UX Expertise Using the PUXMM – A UX Maturity Model for Projects. i-com, Vol. 20 (Issue 1), pp. 105-113. DOI: https://doi.org/10.1515/icom-2020-0029

Gómez A., Mendialdua X., Barmpis K. et al. (2020) Scalable modeling technologies in the wild: an experience report on wind turbines control applications development. Softw Syst Model 19, 1229–1261 (2020). DOI: https://doi.org/10.1007/s10270-020-00776-8

Grønmo R., Møller-Pedersen B. (2011) From UML 2 Sequence Diagrams to State Machines by Graph Transformation, Journal of Object Technology, Volume 10, (2011), pp. 8:1-22, DOI: http://doi.org/10.5381/jot.2011.10.1.a8

Grundspenkis, J., Jekabsons, J. (2001) Model Transformations for Knowledge Base Integration within the Framework of Structural Modelling. In: Barzdins, J., Caplinskas, A. (eds) Databases and Information Systems. Springer, Dordrecht. DOI: https://doi-org.resursi.rtu.lv/10.1007/978-94-015-9636-7_21

Grundspenkis J., Zeltmate I. (2008) Formal method of functional model building based on graph transformations (2008) 7th International Workshop on Modeling and Applied Simulation, MAS 2008, Held at the International Mediterranean Modeling Multiconference, I3M 2008, pp. 140-147. Scopus

Grundspenkis J. (1997) Structural modelling of complex technical systems in conditions of incomplete information: A review (1997) Modern Aspects of Management Science, (1), pp. 111-135

Guliyeva A. (2023) Definition of Patterns for UML Use Cases, Master Thesis, Riga Technical University, 2023

Gundrota N. (2018) https://towardsdatascience.com/code2pix-deep-learning-compiler-for-graphical-user-interfaces-1256c346950b

Gusarovs K. (2020) Metodes izstrāde koda ģenerēšanai no divpusložu modeļa, promocijas darbs, Rīgas Tehniskā universitāte, 2020, DOI: https://doi.org/10.7250/9789934225772

Hailpern B., Tarr P. (2006) Model-driven development: The good, the bad, and the ugly, in IBM Systems Journal, vol. 45, no. 3, pp. 451-461, 2006, DOI: https://doi.org/10.1147/sj.453.0451

Hamdani M., Haider W., Anwar M., Azam F. (2018) A Systematic Literature Review on Interaction Flow Modeling Language (IFML). 134-138. DOI: https://doi.org/10.1145/3180374.3181333

Hanelt A., Bohnsack R., Marz D., Antunes Marante C. (2021) A systematic review of the literature on digital transformation: Insights and implications for strategy and organizational change, Journal of Management Studies, vol. 58, no. 5, pp. 1159–1197, 2021. DOI: http://doi.org/10.1111/joms.12639

Harel D. (1987) Statecharts: a visual formalism for complex systems, Science of Computer Programming, Volume 8, Issue 3, 1987, Pages 231-274, ISSN 0167-6423, DOI: https://doi.org/10.1016/0167-6423(87)90035-9

Hasan H. M., Sanyal F., Chaki D., Ali Md. (2017) An empirical study of important keyword extraction techniques from documents. DOI: https://doi.org/10.1109/ICISIM.2017.8122154

Hasheminejad S. M. H., Jalili S. (2012) Design patterns selection: An automatic two-phase method. Journal of Systems and Software. 85. 408-424. DOI: https://doi.org/10.1016/j.jss.2011.08.031

Hasso S., Carlson C. (2005) A Theoretically-based Process for Organizing Design Patterns.

Haz A. L., Nobuo F., Fajrianti E. D., Sukaridhoto S. (2024) A Study of Summarization and Keyword Extraction Function in Meeting Note Generation System from Voice Records. In Proceedings of the 2023 12th International Conference on Networks, Communication and

Computing (ICNCC '23). Association for Computing Machinery, New York, NY, USA, 106–112. DOI: https://doi.org/10.1145/3638837.3638853

Hebig R., Khelladi D. E., Bendraou R. (2017) Approaches to Co-Evolution of Metamodels and Models: A Survey, in IEEE Transactions on Software Engineering, vol. 43, no. 5, pp. 396-414, 1 May 2017, doi: 10.1109/TSE.2016.2610424.

Hennicker R., Koch N. (2001) A UML-based Methodology for Hypermedia Design. Lecture Notes in Computer Science. DOI: https://doi.org/10.1007/3-540-40011-7_30

Hinderks A., Mayo F. J. D., Thomaschewski J., Escalona M. J. (2022) Approaches to manage the user experience process in Agile software development: A systematic literature review, Information and Software Technology, Volume 150, 2022, 106957, ISSN 0950-5849, DOI: https://doi.org/10.1016/j.infsof.2022.106957

Hitz M., Werthner H. (1995) A Graph Oriented Approach to Enhance Reusability in *-bases.

Horrocks I. (1999) Constructing the User Interface with Statecharts (1st. ed.). A, ddison-Wesley Longman Publishing Co., Inc., USA., ISBN:978-0-201-34278-9

Hsueh N. L., & Kuo J. Y., Lin C. (2009) Object-oriented design: A goal-driven and pattern-based approach. Software and System Modeling. 8. 67-84. DOI: https://doi.org/10.1007/s10270-007-0063-y

Huang J. et al. (2017) Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 3296-3297, DOI: https://doi.org/10.1109/CVPR.2017.351

Huddleston R. (2017) Beginning Adobe Experience Design: Quickly Design and Prototype Websites and Mobile Apps. 10.1007/978-1-4842-2964-4.

Hussmann H., Meixner G., Zühlke D. (2011) Model-Driven Development of Advanced User Interfaces. DOI: https://doi.org/10.1007/978-3-642-14562-9

Huzarr Z., Loniewskil G. (2006) Deriving Prototypes from UML 2.0 Sequence Diagrams, 2006, available at: https://ceur-ws.org/Vol-252/paper09.pdf, last visited 18.04.2024.

ISO/IEC/IEEE 29119 Software Testing Standards, International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE), 2013

Javeed A. (2019) Performance Optimization Techniques for ReactJS, 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 2019, pp. 1-5, DOI: https://doi.org/10.1109/ICECCT.2019.8869134

Jia X., Steele A., Liu H., Qin L., Jones C. (2004) Using ZOOM Approach to Support MDD.. 144-150. Available: https://www.researchgate.net/publication/221610212_Using_ZOOM_Approach_to_Support_MDD

Johnson G., Gross M., Hong J., Do E. (2009) Computational Support for Sketching in Design: A Review. Foundations and Trends in Human-Computer Interaction. 2. 1-93. DOI: http://doi.org/10.1561/1100000013

Jongmans E., Jeannot F., Liang L., Damperat M. (2022) Impact of website visual design on user experience and website evaluation: the sequential mediating roles of usability and pleasure. Journal of Marketing Management. 38. 1-36. 10.1080/0267257X.2022.2085315.

Joo H. (2017) A Study on Understanding of UI and UX, and Understanding of Design According to User Interface Change, in International Journal of Applied Engineering Research ISSN 0973-4562, vol. 12, no. 20, pp. 9931-9935, 2017

Kaluarachchi T., Wickramasinghe M. (2023) A systematic literature review on automatic website generation, Journal of Computer Languages, Volume 75, 2023, DOI: https://doi.org/10.1016/j.cola.2023.101202.

Khoury P. E., Mokhtari A., Coquery E., Hacid M.S. (2008) An Ontological Interface for Software Developers to Select Security Patterns, 2008 19th International Workshop on Database and Expert Systems Applications, Turin, Italy, 2008, pp. 297-301, DOI: https://doi.org/10.1109/DEXA.2008.110

Kim D. K., Khawand C. E. (2007). An approach to precisely specifying the problem domain of design patterns. Journal of Visual Languages & Computing. 18. DOI: https://doi.org/10.1016/j.jvlc.2007.02.009

Kim D. K., Wuwei S. (2008) Evaluating pattern conformance of UML models: A divide-and-conquer approach and case studies. Software Quality Journal. 16. 329-359. DOI: https://doi.org/10.1007/s11219-008-9048-5

Kimseng N., Kurnia D. A., Vuthy I., Arifin R. W., Setiyadi D. (2024) UI/UX Development Using Figma based on Inclusive Design. JINAV: Journal of Information and Visualization, 4(2), 227-234. https://doi.org/10.35877/454RI.jinav2257

Kiruthika J., Khaddaj S., Greenhill D., Francik J. (2016) User Experience Design in Web Applications, 2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES), Paris, France, 2016, pp. 642-646, DOI: https://doi.org/10.1109/CSE-EUC-DCABES.2016.253

Kleppe A. G., Warmer J., Bast W. (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., USA. ISBN: 978-0-321-19442-8

Ko H. K, Park G., Jeon H., Jo J., Kim J., Seo J. (2023) Large-scale Text-to-Image Generation Models for Visual Artists' Creative Works. In Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI '23). Association for Computing Machinery, New York, NY, USA, 919–933. DOI: https://doi.org/10.1145/3581641.3584078

Koch N., Mandel L. (1999) Using UML to design hypermedia applications. Technical Report 9901, Institut für Informatik, Ludwig-Maximilians-Universität, München, March 1999.

Koch N., Baumeister H., Hennicker R., Mandel L. (2000) Extending uml to model navigation and presentation in web applications.

Kolthoff K. (2019) Automatic Generation of Graphical User Interface Prototypes from Unrestricted Natural Language Requirements, 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1234-1237, DOI: https://doi.org/10.1109/ASE.2019.00148

Kompaniets V., Lyz A., Kazanskaya A. (2020) An Empirical Study of Goal Setting in UX/UI-design, 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT), 2020, pp. 1-5, DOI: https://doi.org/10.1109/AICT50176.2020.9368570

Kozačenko L. (2014) Divpusložu modeļa lietošanas analīze UML diagrammu ģenerēšanā, Maģistra darbs, 2014.

Kruchten P. (2003) The Rational Unified Process: An Introduction (3rd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA. ISBN: 978-0-321-19770-2

Kruschitz C., Hitz M. (2010) Human-Computer Interaction Design Patterns: Structure, Methods, and Tools. International Journal on Advances in Software. 3. 225-237.

Lassaad B. A., Fadwa Y. (2021) An Automated Model-Based Approach for Developing Mobile User Interfaces. IEEE Access. PP. 1-1. DOI: https://doi.org/10.1109/ACCESS.2021.3066007

Leuthold S. (2010) User Interface, Navigation Design and Content Representation: Three Perspectives on World Wide Web Navigation. Dr. Lothar Müller (Second Reviewer) Prof. Dr. Andreas Monsch.

Li J., Cao H., Lin L., Hou Y., Zhu R., El Ali A. (2023) User Experience Design Professionals' Perceptions of Generative Artificial Intelligence, DOI: https://doi.org/10.48550/arXiv.2309.15237

Li T. J. J., Popowski L., Mitchell T., Myers B. (2021) Screen2Vec: Semantic Embedding of GUI Screens and GUI Components. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 578, 1–15. DOI: https://doi.org/10.1145/3411764.3445049

Lin J., Landay J. (2002) Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. Proceedings of the 8th International Conference on Distributed Multimedia Systems.

Liu H., Ma F. (2010) Research on visual elements of Web UI design, 2010 IEEE 11th International Conference on Computer-Aided Industrial Design & Conceptual Design 1, Yiwu, China, 2010, pp. 428-430, DOI: https://doi.org/10.1109/CAIDCD.2010.5681317

Liu T. F., Craft M., Situ J., Yumer E., Mech R., Kumar R. (2018) Learning Design Semantics for Mobile Apps. In Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18). Association for Computing Machinery, New York, NY, USA, 569–579. DOI: https://doi.org/10.1145/3242587.3242650

Lopez-Jaquero V., Montero F., González P. (2011) T:XML: A Tool Supporting User Interface Model Transformation. In: Hussmann, H., Meixner, G., Zuehlke, D. (eds) Model-Driven

Development of Advanced User Interfaces. Studies in Computational Intelligence, vol 340. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-14562-9_12

Lumertz P.R., Ribeiro L., Duarte L. M. (2016) User interfaces metamodel based on graphs, Journal of Visual Languages & Computing, Volume 32, 2016, Pages 1-34, ISSN 1045-926X, DOI: https://doi.org/10.1016/j.jvlc.2015.10.026

Lycett M., Marcos E., Storey V. (2007) Model-driven systems development: an introduction. European Journal of Information Systems, 16(4), 346–348. DOI: https://doi.org/10.1057/palgrave.ejis.3000684

Macedo (de) G. T., Fontão A., Gadelha B. (2023) UIProtoCheck: A Checklist for Semantic Inspection of User Interface Prototypes. In Proceedings of the XXXVII Brazilian Symposium on Software Engineering (SBES '23). Association for Computing Machinery, New York, NY, USA, 485–490. DOI: https://doi.org/10.1145/3613372.3613376

Machado M., Couto R., Campos J.C. (2017) MODUS: model-based user interfaces prototyping. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '17). Association for Computing Machinery, New York, NY, USA, 111–116. DOI: https://doi.org/10.1145/3102113.3102146

Mahatody T., Ilie M., Rapatsalahy M. A., Dimbisoa W. G., Ilie S. (2021) Metamodel based approach to generate user interface mockup from UML class diagram, Procedia Computer Science, Volume 184, 2021, Pages 779-784, ISSN 1877-0509, DOI: https://doi.org/10.1016/j.procs.2021.03.096.

Marín B., Giachetti G., Pastor O., Abran A. (2010) A Quality Model for Conceptual Models of MDD Environments. Advances in Software Engineering. 2010. DOI: https://doi.org/10.1155/2010/307391

Raluca M., Seceleanu C., Le Guen H., Pettersson P. (2015) Chapter Three - A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs, Editor(s): Ali R. Hurson, Advances in Computers, Elsevier, Volume 98, 2015, Pages 89-140, ISSN 0065-2458, ISBN 9780128021323, DOI: https://doi.org/10.1016/bs.adcom.2015.03.003

Martinez R. A., Estrada E. H., Pastor O. (2002) From Early Requirements to User Interface Prototyping: A Methodological Approach. 257-260. DOI: https://doi.org/10.1109/ASE.2002.1115025

Martinez E., Pfister L. (2023) Benefits and limitations of using low-code development to support digitalization in the construction industry, Automation in Construction, Volume 152, 2023, 104909, ISSN 0926-5805, DOI: https://doi.org/10.1016/j.autcon.2023.104909

Martins J., Branco F., Mamede H. (2023) Combining low-code development with ChatGPT to novel no-code approaches: A focus-group study, Intelligent Systems with Applications, Volume 20, 2023, 200289, ISSN 2667-3053, DOI: https://doi.org/10.1016/j.iswa.2023.200289

Marzuoki (el) N. (2021) Model Composition in Multi-Modeling Approaches Based on Model Driven Architecture, PhD Thesis, la Faculte des Sciences Dhar El Maharaz Fes, Marocco, 2021

Matra M., Kusumasari T., Al-Anshary F. (2023) Redesign User Interface Website Portal Menggunakan Metode Atomic Design (Studi Kasus: Badan Pengawas Obat Dan Makanan). JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika). 8. 500-513. DOI: http://doi.org/10.29100/jipi.v8i2.3502

Mbaki E. (2013) A Model-Driven Approach for Designing Multi-platform User Interface Dialogues. Ubiquitous Computing. Université catholique de Louvain, 2013. English. https://hal.science/tel-01185463

Mbugua S. T., Korongo J., Samuel M. (2022) On Software Modular Architecture: Concepts, Metrics and Trends, International Journal of Computer and Organization Trends, vol. 12, no. 1, Jan-Apr. 2022, pp. 3-10. DOI: 10.14445/22492593/IJCOT-V12I1P302

McGinnity C. (2021) Atomic design systems, 2021, Available: https://www.hatchd.com.au/blog/atomic-design-systems

Medina J. L. P. (2016) The UsiSketch Software Architecture. Romanian Journal of Human - Computer Interaction 9, 4 (2016), 305–333. http://hdl.handle.net/2078.1/187342

Meixner G., Paternò F., Vanderdonckt J. (2011) Past, Present, and Future of Model-Based User Interface Development. i-com. 10. 2-11. DOI: https://doi.org/10.1524/icom.2011.0026

Mens T. (2013) Model Transformation: A Survey of the State of the Art. Model-Driven Engineering for Distributed Real-Time Systems: MARTE Modeling, Model Transformations and their Usages. 1-19. DOI: https://doi.org/10.1002/9781118558096.ch1

Molina P. J. (2004) User interface generation with OlivaNova model execution system. In Proceedings of the 9th international conference on Intelligent user interfaces (IUI '04). Association for Computing Machinery, New York, NY, USA, 358–359. DOI: https://doi.org/10.1145/964442.964533

Molina P., Meliá S., Pastor O. (2002) User Interface Conceptual Patterns. 159-172. DOI: http://doi.org/10.1007/3-540-36235-5_12

Moran K., Bernal-Cárdenas C., Curcio M., Bonett R., Poshyvanyk D. (2020) Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps, in IEEE Transactions on Software Engineering, vol. 46, no. 2, pp. 196-221, 1 Feb. 2020, DOI: http://doi.org/10.1109/TSE.2018.2844788

Moreira R. M. L. M., Paiva A. C. R., Memon A. (2013) A pattern-based approach for GUI modeling and testing, 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), Pasadena, CA, USA, 2013, pp. 288-297, DOI: https://doi.org/10.1109/ISSRE.2013.6698881

Morgado I. C., Paiva A. C. R. (2015) The iMPAcT Tool: Testing UI Patterns on Mobile Applications, 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 2015, pp. 876-881, DOI: https://doi.org/10.1109/ASE.2015.96

Mukthapuram R. (2021) Analysis of Component Libraries for React JS. IARJSET. 8. 43-46. DOI: http://doi.org/10.17148/IARJSET.2021.8607

Nacheva R. (2017) Prototyping Approach In User Interface Development, 2nd Conference on Innovative Teaching Methods, Bulgaria, 28-29 June, 2017, 80-87

Nagel W. (2022) Multiscreen UX Design. Elsevier Science, 2015. Web. 15 Oct. 2022.

Narang P., Mittal P. (2022) Performance Assessment of Traditional Software Development Methodologies and DevOps Automation Culture. Engineering, Technology & Applied Science Research. 12. 9726-9731. DOI: https://doi.org/10.48084/etasr.5315

Nasiri S., Rhazali Y., Adadi A., Lahmer M. (2023) Generation of User Interfaces and Code from User Stories. In: Joshi, A., Mahmud, M., Ragel, R.G. (eds) Information and Communication Technology for Competitive Strategies (ICTCS 2021). Lecture Notes in

Networks and Systems, vol 400. Springer, Singapore. DOI: https://doi.org/10.1007/978-981-19-0095-2_38

Newman M. W., Landay J.A. (2000) Sitemaps, storyboards, and specifications: a sketch of Web site design practice. In Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '00). Association for Computing Machinery, New York, NY, USA, 263–274. DOI: https://doi.org/10.1145/347642.347758

Nguyen T., Csallner C. (2015) Reverse Engineering Mobile Application User Interfaces with REMAUI (T). 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2015): 248-259. DOI: https://doi.org/10.1109/ASE.2015.32

Nguyen T. D., Vanderdonckt J., Seffah A. (2016) Generative patterns for designing multiple user interfaces. 151-159. DOI: https://doi.org/10.1145/2897073.2897084

Nguyen T. D., Vu P., Pham H., Nguyen T. (2018) Deep Learning UI Design Patterns of Mobile Apps, 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER), Gothenburg, Sweden, 2018, pp. 65-68. DOI: https://doi.org/10.1145/3183399.3183422

Nichols J. (2004) Automatically Generating User Interfaces for Appliances. Pieejams: https://www.researchgate.net/publication/228810732_Automatically_Generating_User_Interfaces_for_Appliances

Nichols J., Chau D. H., Myers B. A. (2007) Demonstrating the viability of automatically generated user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07). Association for Computing Machinery, New York, NY, USA, 1283–1292. DOI: https://doi.org/10.1145/1240624.1240819

Nikiforova O., Gusarovs K. (2016) Comparison of BrainTool to Other UML Modeling and Model Transformation Tools, AIP Conference Proceedings, International Conference on Numerical Analysis and Applied Mathematics 2016, ICNAAM 2016; 6th Symposium on Computer Languages - SCLIT 2016, Rhodes, Greece, 19-25 September 2016, published in 2017 (Web of Science, Scopus) DOI: https://doi.org/10.1063/1.4992503

Nikiforova O., Gusarovs K. (2020) Anemic Domain Model vs Rich Domain Model to Improve the Two-Hemisphere Model-Driven Approach, Applied Computer Systems, 25 (1), 51-56, DOI: https://doi.org/10.2478/acss-2020-0006

Nikiforova O., Kirikova, M. (2004) Two-Hemisphere Model Driven Approach: Engineering Based Software Development. In: Persson, A., Stirna, J. (eds) Advanced Information Systems Engineering. CAiSE 2004. Lecture Notes in Computer Science, vol 3084. Springer, Berlin, Heidelberg, 2004. DOI: https://doi.org/10.1007/978-3-540-25975-6_17

Nikiforova O., Pavlova, N. (2011) Open Work of Two-Hemisphere Model Transformation Definition into UML Class Diagram in the Context of MDA. In: Software Engineering Techniques: Lecture Notes in Computer Science. Vol.4980. Berlin: Springer Berlin Heidelberg, 2011. pp.118-130. ISBN 9783642223853

Nikiforova O. (2002) General Framework for Object-Oriented Software Development Process. Applied computer systems. Vol.13, 2002, pp.132-144. ISSN 1407-7493

Nikiforova O. (2009). Two Hemisphere Model Driven Approach for Generation of UML Class Diagram in the Context of MDA. e-Informatica. 3. 59-72. https://www.e-informatyka.pl/attach/e-Informatica_-_Volume_3/eInformatica2009Art4.pdf

Nikiforova O., Babris K., Kristapsons J. (2020) Survey on Risk Classification in Agile Software Development Projects in Latvia. Applied Computer Systems, 2020, Vol. 25, No. 2, pp. 105-116. ISSN 2255-8683. e-ISSN 2255-8691. DOI: https://doi.org/10.2478/acss-2020-0012 (Web of Science)

Nikiforova O., Babris K., Madelāne L. (2021) Expert Survey on Current Trends in Agile, Disciplined and Hybrid Practices for Software Development, Applied Computer Systems, vol.26, no.1, 2021, pp.38-43. DOI: https://doi.org/10.2478/acss-2021-0005 (Web of Science)

Nikiforova O., Babris K., Mahmoudifar F. (2024a) Automated Generation of Web Application Front-End Components from User Interface Mockups, Proceedings of International Conference on Software Technologies (ICSOFT 2024, July 8-10), SCITEPRESS Digital Library, 2024 (accepted for publication) (SCOPUS)

Nikiforova O., Babris K., Guliyeva A. (2024b) Definition of a Set of Use Case Patterns for Application Systems: A Prototype-Supported Development Approach, Applied Computer Systems, vol.29, no.1, 2024 (submitted for publication) (Web of Science)

Nikiforova O., El Marzouki N., Gusarovs K., Vangheluwe H., Bures T., Al-Ali R., Iacono M., Orue Esquivel P., and Leon F. (2017) The Two-Hemisphere Modelling Approach to the

Composition of Cyber-Physical Systems" - Proceedings of International Conference on Software Technologies (ICSOFT 2017), 24-26 July, 2017, Madrid, Spain. SCITEPRESS Digital Library, pp. 286-293, DOI: https://doi.org/10.5220/0006424902860293

Nikiforova O., K. Gusarovs, O. Gorbiks, N. Pavlova (2012) BrainTool. A Tool for Generation of the UML Class Diagrams" Proceedings of the Seventh International Conference on Software Engineering Advances, Mannaert H. et al. (Eds), IARIA ©, Lisbon, Portugal, November 18-23, 2012, pp. 60-69 (Scopus)

Nikiforova O., Gusarovs, K., Gorbiks, O., Pavlova, N. (2013) Improvement of the Two-Hemisphere Model-Driven Approach for Generation of the UML Class Diagram. Applied Computer Systems. Vol.14, 2013, pp.19-30. ISSN 22558683. DOI: https://doi.org/10.2478/acss-2013-0003

Nikiforova O., Iacono M., El Marzouki N., Romanovs A. and Vangheluwe H. (2020) Enabling Composition of Cyber-Physical Systems with the Two-Hemisphere Model-Driven Approach, In: Multi-Paradigm Modelling Approaches for Cyber-Physical Systems. B.Tekinerdogan, D.Blouin, H.Vangheluwe, M.Goulão, P.Carreira, V.Amaral ed. 125 London Wall, London EC2Y 5AS, United Kingdom: Academic Press is an imprint of Elsevier, 2020. pp. 149-168, ISBN 978-0-12-819105-7 (Scopus)

Nikiforova O., Kozacenko L. and Ahilcenoka D. (2013) UML Sequence Diagram: Transformation from the Two-Hemisphere Model and Layout, Applied Computer Systems, vol.14, no.1, 2013, pp.31-41. DOI: https://doi.org/10.2478/acss-2013-0004

Nikiforova O., Kozacenko, L., Ahilcenoka, D., Gusarovs, K., Ungurs, D., Jukss, M. (2015) Comparison of the Two-Hemisphere Model-Driven Approach to Other Methods for Model-Driven Software Development, Scientific Journal of Riga Technical University: Applied Computer Systems, 18,. 5-14, DOI: http://doi.org/10.1515/acss-2015-0013

Nikiforova O., Sukovskis U., Gusarovs K. (2015). Application of the two-hemisphere model supported by BrainTool: Football game simulation. AIP Conference Proceedings. 1648. DOI: https://doi.org/10.1063/1.4912557

Nikiforova O., Zabiniako V., Kornienko J., Rizhko R., Babris K., Gasparoviča-Asīte M. (2021a) Efficiency Monitoring of Engineering System Designer Work Based on Multi-System User Behavior Analysis with AI/ML Algorithms, 2021 IEEE 62nd International Scientific

Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), 2021, pp. 1-6, DOI: https://doi.org/10.1109/RTUCON53541.2021.9711720 (SCOPUS)

Nikiforova O., Zabiniako V., Kornienko J., Rizhko R., Babris K., Nikulsins V., Garkalns P., Gasparoviča-Asīte M. (2021b) Solution to On-line vs On-site Work Efficiency Analysis on the Example of Engineering System Designer Work, Applied Computer Systems, vol.26, no.2, 2021, pp. 87-95, DOI: https://doi.org/10.2478/acss-2021-0011 (SCOPUS)

Novac O. C., Madar D. E., Novac C. M., Bujdosó G., Oproescu M., Gal T. (2021) Comparative study of some applications made in the Angular and Vue.js frameworks, 2021 16th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 2021, pp. 1-4, DOI: https://doi.org/10.1109/EMES52337.2021.9484150

Obrist M., Tscheligi M., de Ruyter B., Schmidt A. (2010) Contextual user experience: how to reflect it in interaction designs? In CHI '10 Extended Abstracts on Human Factors in Computing Systems (CHI EA '10). Association for Computing Machinery, New York, NY, USA, 3197–3200. DOI: https://doi.org/10.1145/1753846.1753956

Omari M., Erramdani M., Rhouati A. (2020) A Model Driven Approach for Generating Angular 7 Applications. International Journal of Recent Contributions from Engineering, Science & IT (iJES). 8. 36. DOI: https://doi.org/10.3991/ijes.v8i2.14131

Osis J., Asnina E. (2011) Model-Driven Domain Analysis and Software Development: Architectures and Functions. 1 edition. Hershey - New York, USA: IGI Global, 2011. 514 p. ISBN13: 9781616928742. Available from: DOI: http://doi.org/10.4018/978-1-61692-874-2

Pandian V. P. S., Suleri S., Beecks C., Jarke M., (2021) MetaMorph: AI Assistance to Transform Lo-Fi Sketches to Higher Fidelities. In Proceedings of the 32nd Australian Conference on Human-Computer Interaction (OzCHI '20). Association for Computing Machinery, New York, NY, USA, 403–412. DOI: https://doi.org/10.1145/3441000.3441030

Pang, B., Nijkamp E., Wu Y. N. (2020) Deep Learning With TensorFlow: A Review. Journal of Educational and Behavioral Statistics, 45(2), 227-248. DOI: https://doi.org/10.3102/1076998619872761

Pantelelis M., Kalloniatis C. (2023) Mapping CRUD to Events - Towards an object to event-sourcing framework. 285-289. DOI: https://doi.org/10.1145/3575879.3576006

Pastor O., Abrahao S., Molina J.C., Torres I. (2001) A FPA-like Measure for Object Oriented Systems from Conceptual Models, Current Trends in Software Measument, Ed. Shaker Verlag, pages 51–69, Montreal, Canada, 2001.

Pavlova N. (2008) Platformneatkarīga modeļa izstrādes pieeja modeļvadāmas arhitektūras ietvarā, promocijas darbs, Rīgas Tehniskā universitāte, 2008

Pelechano V., Pastor O., Insfrán E. (2002) Automated code generation of dynamic specializations: an approach based on design patterns and formal techniques, Data & Knowledge Engineering, Volume 40, Issue 3, 2002, Pages 315-353, ISSN 0169-023X, DOI: https://doi.org/10.1016/S0169-023X(02)00020-4

Peres A. L., Meira S. L. (2015) Towards a framework that promotes integration between the UX design and SCRUM, Aligned to CMMI, 2015 10th Iberian Conference on Information Systems and Technologies (CISTI), Aveiro, Portugal, 2015, pp. 1-4, DOI: https://doi.org/10.1109/CISTI.2015.7170443

Pimentel J., Castro J., Mylopoulos J., Angelopoulos K., Souza V. S. (2014) From requirements to statecharts via design refinement. Proceedings of the ACM Symposium on Applied Computing. DOI: https://doi.org/10.1145/2554850.2555056

Planas E., Daniel G., Brambilla M. et al. (2021) Towards a model-driven approach for multiexperience AI-based user interfaces. Softw Syst Model 20, 997–1009 (2021). DOI: https://doi.org/10.1007/s10270-021-00904-y

Popp R., Raneburger D., Kaindl H. (2013) Tool support for automated multi-device GUI generation from discourse-based communication models. In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '13). Association for Computing Machinery, New York, NY, USA, 145–150. DOI: https://doi.org/10.1145/2494603.2480334

Qing H., Ibrahim R., Nies H.W. (2024) Analysis of web design visual element attention based on user educational background. Sci Rep 14, 4657 (2024), DOI: https://doi.org/10.1038/s41598-024-54444-8

Queiroz R., Marques A. B., Lopes A., Oliveira E., Conte T. (2018) Evaluating Usability of IFML Models: How Usability is Perceived and Propagated. In Proceedings of the 17th Brazilian Symposium on Human Factors in Computing Systems (IHC '18). Association for

Computing Machinery, New York, NY, USA, Article 21, 1–10. DOI: https://doi.org/10.1145/3274192.3274213

Raharjana I., Arifin M., Nur A., Mubarak N. (2023) Conversion of User Story Scenarios to Python-Based Selenium Source Code for Automated Testing. TEM Journal. 12. 309-315. 10.18421/TEM121-39.

Raneburger D., Kaindl H., Popp R. (2015) Model transformation rules for customization of multi-device graphical user interfaces. In Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '15). Association for Computing Machinery, New York, NY, USA, 100–109. https://doi.org/10.1145/2774225.2774839

Rech J., Bunse C. (2008) Model-Driven Software Development - Integrating Quality Assurance, (2008)., DOI: http://doi.org/10.4018/978-1-60566-006-6

Riaz S., Arshad A., Band S. S., & Mosavi A. (2022) Transforming Hand Drawn Wireframes into Front-End Code with Deep Learning, Computers, Materials & Continua. 72. 4303-4321. DOI: https://doi.org/10.32604/cmc.2022.024819

Riccio V., Jahangirova G., Stocco A. et al. (2020) Testing machine learning based systems: a systematic mapping. Empir Software Eng 25, 5193–5254 (2020). DOI: https://doi.org/10.1007/s10664-020-09881-0

Rivero J., Rossi G., Grigera J., Luna R. E., Navarro A. (2011) From Interface Mockups to Web Application Models, 6997. 257-264. DOI: http://doi.org/10.1007/978-3-642-24434-6_20

Robinson A. (2019) Sketch2code: Generating a website from a paper mockup, 2019, DOI: https://doi.org/10.48550/arXiv.1905.13750

Rokis K., Kirikova M. (2022) Challenges of Low-Code/No-Code Software Development: A Literature Review. In: Nazaruka, Ē., Sandkuhl, K., Seigerroth, U. (eds) Perspectives in Business Informatics Research. BIR 2022. Lecture Notes in Business Information Processing, vol 462. Springer, Cham. https://doi-org.resursi.rtu.lv/10.1007/978-3-031-16947-2_1

Roubi S., Erramdani M., Mbarki S. (2016) Modeling and generating graphical user interface for MVC Rich Internet Application using a model driven approach, 2016 International Conference on Information Technology for Organizations Development (IT4OD), Fez, Morocco, 2016, pp. 1-6, doi: https://doi.org/10.1109/IT4OD.2016.7479249

Rudd J., Stern K., Isensee S. (1996) Low vs. high-fidelity prototyping debate. interactions 3, 1 (Jan. 1996), 76–85. https://doi.org/10.1145/223500.223514

Ruscio D. D., Kolovos D., de Lara J. et al. (2022) Low-code development and model-driven engineering: Two sides of the same coin?. Softw Syst Model 21, 437–446 (2022). DOI: https://doi.org/10.1007/s10270-021-00970-2

Sajji A., Rhazali Y., Hadi Y. (2022) An Approach to Automate Generation of Graphical User Interfaces Through IFML. Procedia Comput. Sci. 201, C (2022), 621–626. DOI: https://doi.org/10.1016/j.procs.2022.03.081

Samir M., Elsayed A., Marie M. I. (2024) A Model for Automatic Code Generation from High Fidelity Graphical User Interface Mockups using Deep Learning Techniques, International Journal of Advanced Computer Science and Applications(IJACSA), 15(3), 2024. DOI: http://doi.org/10.14569/IJACSA.2024.0150369

Santoso M. (2024) Implementation Of UI/UX Concepts And Techniques In Web Layout Design With Figma. Jurnal Teknologi Dan Sistem Informasi Bisnis, 6(2), 279-285. https://doi.org/10.47233/jteksis.v6i2.1223

Sharp H., Rogers Y., Preece J. (2019) Interaction Design: Beyond Human Computer Interaction (5th Edition), ISBN: 9781119547259, Wiley, 2019

Shatnawi A., Shatnawi R (2016) Generating a Language-Independent Graphical User Interfaces from UML Models. IAJIT. 13., 2016

Shobha K. (2024) Design and Implementation of the Music Website Design using Figma. International Journal of Innovative Science and Research Technology (IJISRT). 3335-3340. 10.38124/ijisrt/IJISRT24MAY2501.

Sibal R., Kaur P., Sharma C. (2018) Prioritization of User Story Acceptance Tests in Agile Software Development Using Meta-Heuristic Techniques and Comparative Analysis, DOI: 10.1007/978-981-13-2348-5_4.

Siering M. (2017) Teaching a machine to convert wireframes into code. Website. Retrieved August 1, 2021 from https://tech.xing.com/teaching-a-machineto-convert-wireframes-into-code-f9333e125e61

Silva (da) A. R. (2003) The XIS approach and principles, 2003 Proceedings 29th Euromicro Conference, Belek-Antalya, Turkey, 2003, pp. 33-40, DOI: https://doi.org/10.1109/EURMIC.2003.1231564

Silva (da) T. S., Martin A., Maurer F., Silveira M. (2011) User-Centered Design and Agile Methods: A Systematic Review, 2011 Agile Conference, 2011, pp. 77-86, DOI: https://doi.org/10.1109/AGILE.2011.24

Silva (da) A. R., Filipe L., Afonso P., Junior P., Freire P. (2022) Mobile User Interaction Design Patterns: A Systematic Mapping Study, Information 13, no. 5: 236. DOI: https://doi.org/10.3390/info13050236

Singh P., Srivastava M., Kansal M., Singh A. P., Chauhan A., Gaur A. (2023) A Comparative Analysis of Modern Frontend Frameworks for Building Large-Scale Web Applications, 2023 International Conference on Disruptive Technologies (ICDT), Greater Noida, India, 2023, pp. 531-535, DOI: https://doi.org/10.1109/ICDT57929.2023.10150911

Soares F. L., Falcao T. A. et al. (2023) A React Style Guide Library for MUI Web Apps, 2023 9th International HCI and UX Conference in Indonesia (CHIuXiD), Bali, Indonesia, 2023, pp. 77-82, DOI: https://doi.org/10.1109/CHIuXiD59550.2023.10452729

Sottet JS., Calvary G., Coutaz J. et al. (2007) A language perspective on the development of plastic multimodal user interfaces. J Multimodal User Interfaces 1, 1–12 (2007). DOI: https://doi.org/10.1007/BF02910054

Spero E., Biddle R. (2020) Home and Away: UI Design Patterns for Supporting End-User Security. In Proceedings of the European Conference on Pattern Languages of Programs 2020 (EuroPLoP '20). Association for Computing Machinery, New York, NY, USA, Article 13, 1–9. DOI: https://doi.org/10.1145/3424771.3424780

Stahl T., Voelter M., Czarnecki K. (2006) Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons, Inc., Hoboken, NJ, USA. ISBN:978-0-470-02570-3

Stahl T., Völter M., Bettin J., Haase A., Helsen S. (2006) Model-driven software development - technology, engineering, management. Pitman pp. I-XVI, 1-428, ISBN: 978-0-470-02570-3

Stige Å., Zamani E.D., Mikalef P., Zhu Y. (2023) Artificial intelligence (AI) for user experience (UX) design: a systematic literature review and future research agenda, Information Technology & People. DOI: https://doi.org/10.1108/ITP-07-2022-0519

Stompff G., Smulders F. (2015) The Right Fidelity: Representations That Speed Up Innovation Processes. Design Manag J, 10: 14-26. DOI: https://doi.org/10.1111/dmj.12019

Subramanian, V. (2017) React-Bootstrap. In: Pro MERN Stack. Apress, Berkeley, CA. DOI: https://doi.org/10.1007/978-1-4842-2653-7_10

Suleri S., Kipi N., Jarke M. (2019) UI Design Pattern-driven Rapid Prototyping for Agile Development of Mobile Applications. DOI: https://doi.org/10.1145/3338286.3344399

Suleri S., Pandian V. P. S., Shishkovets S., Jarke M. (2019) Eve: A Sketch-based Software Prototyping Workbench. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19). Association for Computing Machinery, New York, NY, USA, Paper LBW1410, 1–6. DOI: https://doi.org/10.1145/3290607.3312994

Sunitha E. V., Samuel P. (2019) Automatic Code Generation From UML State Chart Diagrams, in IEEE Access, vol. 7, pp. 8591-8608, 2019, DOI: https://doi.org/10.1109/ACCESS.2018.2890791

Sutipitakwong S., Jamsri P., (2020) Pros and Cons of Tangible and Digital Wireframes. DOI: https://doi.org/10.1109/FIE44824.2020.9274234

Tena S., Diez D., Díaz P., Aedo I. (2013) Standardizing the Narrative of Use Cases: A Controlled Vocabulary of Web User Tasks. Information and Software Technology. 55. 1580-1589. https://doi.org/10.1016/j.infsof.2013.02.012

Thomas D. (2004) MDA: Revenge of the Modelers or UML Utopia? IEEE Softw. 21, 3 (May 2004), 15–17. DOI: https://doi.org/10.1109/MS.2004.1293067

Tidwell J. (2020) Designing Interfaces: Patterns for Effective Interaction Design; O'Reilly Media, Inc.: Newton, MA, USA, 2020. ISBN: 978-1492051961

Tomasdottir K. F., Aniche M., Deursen A. (2017) Why and how JavaScript developers use linters. 578-589. 10.1109/ASE.2017.8115668.

Tomasdottir K. F., Aniche M., Deursen A. (2020) The Adoption of JavaScript Linters in Practice: A Case Study on ESLint, in IEEE Transactions on Software Engineering, vol. 46, no. 8, pp. 863-891, 1 Aug. 2020, DOI: 10.1109/TSE.2018.2871058

Toxboe A. (2024) UI-Patterns.com — User Interface Design Pattern Library. Pieejams tiešsaistē: http://ui-patterns.com/ (apskatīts – 10.05.2024).

Trehan V., Chapman C., Raju P. (2015) Informal and formal modelling of engineering processes for design automation using knowledge based engineering. J. Zhejiang Univ. Sci. A 16, 706–723 DOI: https://doi.org/10.1631/jzus.A1500140

Uyanık B., Sayar A. (2024) Analysis and Comparison of Automatic Code Generation and Transformation Techniques on Low-Code Platforms. In Proceedings of the 2023 5th International Conference on Software Engineering and Development (ICSED '23). Association for Computing Machinery, New York, NY, USA, 17–27. https://doi-org.resursi.rtu.lv/10.1145/3637792.3637795

Uzayr S. (2022) Mastering UI Mockups and Frameworks: A Beginner's Guide (1st ed.). CRC Press. DOI: https://doi.org/10.1201/b22860

Vadiyala V. R., Baddam P. R. (2017) Mastering JavaScript's Full Potential to Become a Web Development Giant. 2. 13-24.

Vanderdonckt J. (2001) A Small Knowledge-Based System for Selecting Interaction Styles. In: Vanderdonckt, J., Farenc, C. (eds) Tools for Working with Guidelines. Springer, London. DOI: https://doi.org/10.1007/978-1-4471-0279-3_24

Vanderdonckt J. (2005) A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Pastor O., Falcão e Cunha J. (eds) Advanced Information Systems Engineering. CAiSE 2005. Lecture Notes in Computer Science, vol 3520. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/11431855_2

Jain V., Agrawal P., Banga S., Kapoor R., Gulyani S. (2019) Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network. DOI: https://doi.org/10.48550/arXiv.1910.08930

Virzi R. A., Sokolov J. L., Karis D. (1996) Usability problem identification using both low- and high-fidelity prototypes. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96). Association for Computing Machinery, New York, NY, USA, 236–243. DOI: https://doi.org/10.1145/238386.238516

Völter M., Bettin J. (2004) Patterns for Model-Driven Software-Development. 525-560.

Wang L., Song T., Song H. N., Zhang S. (2022) Research on Design Pattern Detection Method Based on UML Model with Extended Image Information and Deep Learning. Appl. Sci. 2022, 12, 8718. https://doi.org/10.3390/app12178718

Warfel T.Z. (2009) Prototyping: A Practitioner's Guide. , p.195. ISBN: 978-1933820217

Welie M., Veer G. (2003) Pattern Languages in Interaction Design. Available: https://www.researchgate.net/publication/221055002_Pattern_Languages_in_Interaction_Design

Welie M., Veer G., Elins A. (2000) Patterns as Tools for User Interface Design. DOI: https://doi.org/

Wellner P. D. (1989) Statemaster: A UIMS based on statecharts for prototyping and target implementation. SIGCHI Bull. 20, SI (March 1989), 177–182. DOI: https://doi.org/10.1145/67450.67486

Wimmer C., Untertrifaller A., Grechenig T. (2020) SketchingInterfaces: A Tool for Automatically Generating High-Fidelity User Interface Mockups from Hand-Drawn Sketches. In 32nd Australian Conference on Human-Computer Interaction (OzCHI '20). Association for Computing Machinery, New York, NY, USA, 538–545. DOI: https://doi-org.resursi.rtu.lv/10.1145/3441000.3441015

Wolff A., Forbrig P. (2010) Pattern Catalogs using the Pattern Language Meta Language. ECEASST. 25. DOI: https://doi.org/10.14279/tuj.eceasst.25.371

Wolff A., Forbrig P., Dittmar A., Reichart D. (2005) Linking GUI elements to tasks: supporting an evolutionary design process. In Proceedings of the 4th international workshop on Task models and diagrams (TAMODIA '05). Association for Computing Machinery, New York, NY, USA, 27–34. DOI: https://doi.org/10.1145/1122935.1122941

Yigitbas E., Jovanovikj I., Biermeier K. et al. (2020) Integrated model-driven development of self-adaptive user interfaces. Softw Syst Model 19, 1057–1081 (2020). DOI: https://doi.org/10.1007/s10270-020-00777-7

Zhang M., Meng W. (2020) Detecting and understanding JavaScript global identifier conflicts on the web. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE

2020). Association for Computing Machinery, New York, NY, USA, 38–49, DOI: https://doi.org/10.1145/3368089.3409747

Zhao M., Gao Y., Liu C. (2012) Research and Achievement of UI Patterns and Presentation Layer Framework, 2012 Fourth International Conference on Computational Intelligence and Communication Networks, Mathura, India, 2012, pp. 870-874, DOI: https://doi.org/10.1109/CICN.2012.175

Zwingelberg H. (2011) Necessary Processing of Personal Data: The Need-to-Know Principle and Processing Data from the New German Identity Card. In: Fischer-Hübner, S., Duquenoy, P., Hansen, M., Leenes, R., Zhang, G. (eds) Privacy and Identity Management for Life. Privacy and Identity 2010. IFIP Advances in Information and Communication Technology, vol 352. Springer, Berlin, Heidelberg, DOI: https://doi.org/10.1007/978-3-642-20769-3_13

# PIELIKUMI

**1. pielikums.**

**Divpusložu modeļa konceptu definēšana**

**1. tabula**

**Koncepta "Prece" apraksts**

| **Nosaukums** | **Datu tips** | **Apraksts** |
|---|---|---|
| ID | Integer | Preces unikālais ID |
| Name | Char | Preces nosaukums |
| Price | Float | Preces cena |
| Description | Char | Preces apraksts |

**2. tabula**

**Koncepta "Lietotājs" apraksts**

| **Nosaukums** | **Datu tips** | **Apraksts** |
|---|---|---|
| ID | Integer | Lietotāja unikālais ID |
| Email | Char | Lietotāja e-pasts |
| Password | Char | Parole |
| First_name | Char | Lietotāja vārds |
| Last_name | Char | Lietotāja uzvārds |
| Company_name | Char | Uzņēmuma nosaukums |
| Company_reg | Char | Uzņēmuma reģistrācijas numurs |
| Company_vat | Char | Uzņēmuma PVN reģistrācijas numurs |
| Company_address | Address | Uzņēmuma juridiskā adrese |

**3. tabula**

**Koncepta "Attēls" apraksts**

| **Nosaukums** | **Datu tips** | **Apraksts** |
|---|---|---|
| ID | Integer | Attēla unikālais ID |
| Item_ID | Item | Norāde uz piesaistītās preces ID |
| Name | Char | Attēla nosaukums |

| Nosaukums | Datu tips | Apraksts |
|---|---|---|
| Description | Char | Attēla apraksts |
| URL | Char | Norāde uz attēla atrašanās vietu |
| Default | Boolean | Norāde vai attēls ir galvenais |

**4. tabula**

**Koncepta "Adrese" apraksts**

| Nosaukums | Datu tips | Apraksts |
|---|---|---|
| ID | Integer | Adreses unikālais ID |
| Country | Item | Valsts |
| City | Char | Pilsēta |
| Address | Char | Adrese |
| ZIP | Char | Pasta indekss |

**5. tabula**

**Koncepta "Grozs" apraksts**

| Nosaukums | Datu tips | Apraksts |
|---|---|---|
| ID | Integer | Lietotāja unikālais ID |
| Item_ID | Item | Preces unikālais ID |
| User_ID | User | Lietotāja unikālais ID |
| Price | Float | Izvēlētās preces cena |
| Quantity | Int | Cik būs produkta lietotāju abonēšanas periodā |
| Frequency | Char | Norāda abonēšanas periodu – mēnesi vai gadu |
| Auto_renew | Boolean | Atzīme vai automātiski pagarināt abonementu |
| Total | Float | Gala cena (preces cenas reizinājums ar lietotāju skaitu) |

**6. tabula**

**Koncepta "Pasūtījums" apraksts**

| **Nosaukums** | **Datu tips** | **Apraksts** |
|---|---|---|
| ID | Integer | Pasūtījuma unikālais ID |
| Item_ID | Item | Preces unikālais ID |
| User_ID | User | Lietotāja unikālais ID |
| Address_ID | Address | Adreses unikālais ID |
| Price | Float | Izvēlētās preces cena |
| Quantity | Int | Cik būs produkta lietotāju abonēšanas periodā |
| Frequency | Char | Norāda abonēšanas periodu – mēnesi vai gadu |
| Auto_renew | Boolean | Atzīme vai automātiski pagarināt abonementu |
| Status | Char | Pasūtījuma status |
| Total | Float | Gala cena (preces cenas reizinājums ar lietotāju skaitu) |

**Kristaps Babris** dzimis 1986. gadā Rīgā. Rīgas Tehniskajā universitātē ieguvis bakalaura (2015) un maģistra (2018) grādu datorsistēmās ar programmēšanas inženiera kvalifikāciju. Kopš 2012. gada strādā SIA "*IT Sapiens*", ieņemot IT struktūrvienības vadītāja amatu, un kopš 2016. gada strādā Rīgas Tehniskajā universitātē zinātniskā asistenta amatā. Zinātniskās intereses ir saistītas ar modeļvadāmas inženierijas principu lietošanu tīmekļa sistēmu izstrādē.