TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika eriala

Margus Ojanurme

Kohanemisvõimelise dialoogsüsteemi prototüüp

Bakalaureusetöö (6 EAP)

Juhendaja: Margus Treumuth

Autor: ..... "....." mai 2012
Juhendaja: ..... "....." mai 2012

Lubada kaitsmisele:
Professor: ..... "....." 2012

TARTU 2012

Sisukord

Sissejuhatus

Käesolev bakalaureusetöö käsitleb dialoogsüsteeme. Dialoogsüsteem on arvutiprogramm, mis saab sisendiks kasutaja lause (antud juhul) eesti keeles ning vastab kasutajale samuti eesti keeles. Selle tulemusel tekib sidus vestlus, mille käigus antakse inimesele üldjuhul infot teatud kitsa ainevaldkonna kohta.

Tartu Ülikooli tudengite poolt on seni realiseeritud mitmeid dialoogsüsteeme. Tutvudes varasemate töödega eestikeelsete dialoogsüsteemide loomisel (Raul Sirel [1], Margus Treumuth [2], Joel Edenberg [3]), võis märgata, et seniste dialoogsüsteemide üldine põhimõtteline realisatsioon (st võtmesõnale vastava vastuse väljastamine) on ennast õigustanud. Seetõttu näis otstarbekas mitte korrata eelmainitud realisatsioone, vaid võtta üks neist aluseks ning panustada selle süsteemi edasiarendusse. Senistest töödest osutus (ka lähtekoodina) kättesaadavaks Margus Treumuth'i poolt realiseeritud raamistik (ADS raamistik[1]), mis ongi võetud käesolevas töös tehtavate edasiarenduste aluseks.

Edasiarenduste eesmärk oli luua olemasolevale dialoogsüsteemide raamistikule sellised uued komponendid, mis kohanevad kasutaja stiili järgi.

Töö käigus lisati ADS raamistikule järgnev uus funktsionaalsus:

- kasutaja kasutab suuri/väikeseid algustähti lause alguses, süsteem hakkab samuti kasutama vastavalt kas suuri või väikeseid algustähti;
- kasutaja trükib kiiresti või aeglaselt, süsteem vastab sama kiiresti või aeglaselt;
- kasutaja kasutab slängi, süsteem saab aru ka sõna slängivormidest.

Töö koosneb viiest osast. Esimeses osas tutvustatakse ADS raamistikku. Teises osas uuritakse funktsionaalseid probleeme (sh arvestades kasutajate senist tagasisidet) ning langetatakse otsused ADS raamistiku täiendusvajaduste osas. Kolmas osa käsitleb tehnilisi võimalusi ADS raamistiku

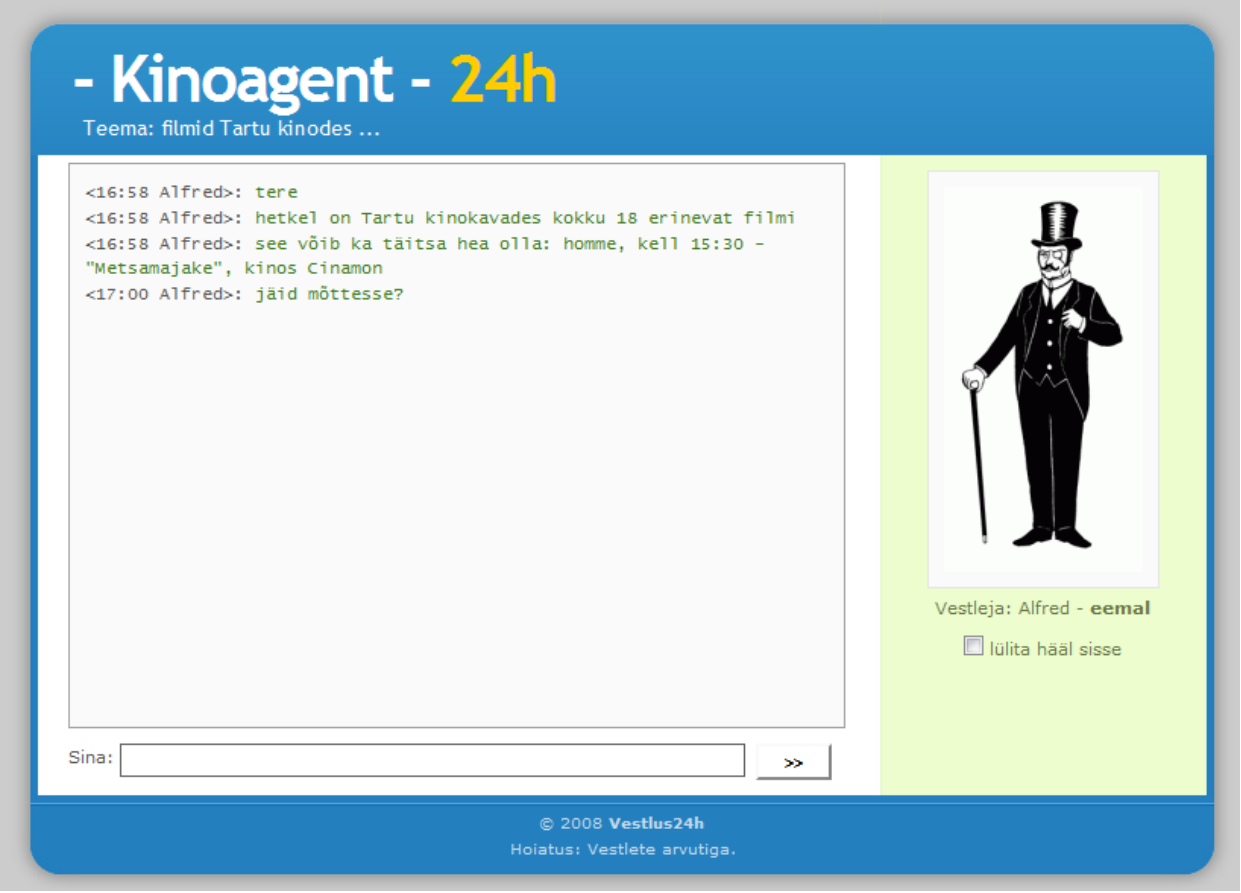
[1] ADS raamistik - asünkroonne dialoogsüsteemi raamistik

edasiseks muutmiseks ning kirjeldab tehtud muudatusi. Neljas osa räägib kasutajatelt saadud tagasisidest. Viiendas osas käsitletakse edasiarendusvõimalusi.

Tööle on lisatud juhend ADS raamistikule lisamoodulite koostamiseks.

1. ADS raamistiku ülevaade

ADS raamistik on töövahendite kogum dialoogsüsteemide loomiseks. Raamistik sisaldab taaskasutatavaid komponente, mida saab kasutada tekstipõhiste, kasutajaga loomulikus keeles suhtlevate dialoogsüsteemide loomisel veebis. Raamistiku abil loodavaid dialoogsüsteeme iseloomustab suhtluse asünkroonsus – kasutaja ei pea ootama vastust dialoogsüsteemilt ja võib esitada küsimuse suvalisel hetkel. Ka arvuti ei pea passiivselt ootama kasutaja sisendit, vaid saab vastata suvalisel hetkel ja anda vajadusel infot ka monoloogina. Sellest tuleneb ka lühend ADS raamistik (asünkroonsete dialoogsüsteemide raamistik).



Joonis 1. ADS raamistikuga realiseeritud kinoinfo dialoogsüsteemi kasutajaliides.

Alljärgnevalt loetletakse olulisemad komponendid, mis on realiseeritud raamistiku funktsioonidena:

- keelest sõltumatu lahendus õigekirja kontrollimiseks ja kasutaja sisendis leiduvate võimalike kirjavigade automaatseks parandamiseks;
- keelest sõltumatu lahendus sõnade järjekorra ignoreerimiseks, mis ühelt poolt annab võimaluse loobuda kasutaja sisendi süntaktilisest analüüsist ning teisalt lihtsustab teadmusbaasi lisatavate reeglite struktuuri;
- inimabi võimalus: süsteemi administraator saab vajadusel sekkuda info andmisse, abistades dialoogsüsteemi raskematel juhtudel.

Eelkirjeldatud raamistiku vahendeid kasutades on loodud kaks dialoogsüsteemi: kinoinfo [Joonis 1] ja hambaravi valdkonnas.

2. Täiendusvajaduste analüüs

ADS raamistiku testimisel olid kasutajad kirjeldanud teatud probleeme, kus võis märgata kasutajate arvamuste lahknevust. Teisisõnu, näis, et kasutajad eeldavad dialoogsüsteemilt teatavat kohanemisvõimet kasutaja stiili osas. Kui kasutajailt küsiti dialoogsüsteemi vastamise kiiruse kohta, siis jagunesid arvamused kaheks. Hea trükkimiskiirusega kasutajate arvates vastas süsteem liiga aeglaselt, kuid aeglasema trükkimiskiirusega kasutajate arvates kulges vestlus liiga kiiresti.

Antud probleemi lahenduseks näis olevat kohanemisvõime lisamine raamistikule ehk võime kohaneda kasutaja trükkimise kiirusega. Antud mooduli ülesanne on hinnata trükkimiskiirust ning selle alusel teha järeldus kasutajale sobiva kiiruse osas ning rakendada see kiirus süsteemipoolsete vastuste kuvamisel. Vajalik oli ka kaaluda, kas kiirus peaks olema ühekordselt muutuv või võiks vestluskiirust muuta ka mitu korda vestluse jooksul.

Vestluste logide analüüsist selgus, et osad kasutajad kasutavad vastamisel slängi, mille peale süsteem nendest aru ei saanud. Teatud lihtsamate slängivormide tuvastusel oli küll abiks ADS raamistiku sõnede sarnasusel baseeruv automaatne õigekirjakorrektor, kuid see ei suutnud

lahendada raskemaid vorme. Esmalt lisati morfoloogilise analüsaatori kasutajasõnastikku mõned sagedasemad slängivormid, mida kasutajad kasutama kippusid ja mida morfoloogiline analüsaator seni ei tundnud. Seejärel leiti Muischneki, Kaalepi ja Sireli [4] koostatud morfoloogilise analüsaatori kasutajasõnastik, mis oli orienteeritud just internetisuhtluse slängivormidele. Raamistikus otsustati võtta kasutusele ka see sõnastik.

Seepeale suureneks tõenäosus, et kasutaja sisendis leiduv võimalik slängisõna teisendatakse teadmusbaasi jaoks arusaadavasse vormi ja arvuti suudab anda vastuse. Probleemiks jääb siiski slängi tuvastamine juhtudel, mil slängisõnastik ei sisalda sobivat vormi.

Kokkuvõtvalt, analüüsi tulemusel otsustati realiseerida järgnev uus funktsionaalsus:

- kasutaja kasutab suuri/väikeseid algustähti lause alguses, süsteem hakkab samuti kasutama vastavalt kas suuri või väikeseid algustähti;
- kasutaja trükib kiiresti või aeglaselt, süsteem vastab sama kiiresti või aeglaselt;
- kasutaja kasutab slängi, süsteem saab aru ka sõna slängivormidest.

3. Täienduste realisatsioon

Olemasolev süsteem kasutas *AJAX*[1] [5] päringute tegemiseks üsna lihtsat lähenemist, mis ei arvestanud kõikide võimalike internetilehitsejate eripärasustega. Selle vea parandamiseks tuli võtta kasutusele hetkel kõige viimase *jQuery*[2](versioon 1.7.2) tugiraamistik, mis selle töö ära teeb. Seega tuli asendada praeguste *AJAX* päringute loomise kood *jQuery AJAX* päringute koodiga. *jQuery* kasutuselevõtt lihtsustas ka koodi loetavust.

Praeguses süsteemis vahetas klient serveriga andmeid, mis ei olnud kliendi jaoks lihtsalt kasutatavad ja vajasid enne andmete kasutamist nende töötlemist. Nimelt olid andmed XML-kujul, kuna klient kasutab *JavaScript*, siis õigeks kujuks oleks *JSON*-formaat [6]. Serveri pool oli kirjutatud *PHP* keeles ning seal on lihtne väljastatavad andmed muuta *JSON* tüüpi andmeteks

[1] AJAX - asünkroonne *JavaScript* ja *XML*

[2] jQuery - JavaScript tugiraamistik, rohkem informatsioon http://www.jquery.com

kasutades funktsiooni *json_encode* [7], kuid kuna see funktsioon on saadaval alles *PHP* versioonist 5.2.0 ning praegune programm töötab veel vanema versiooni peal, siis tuli kirjutada andmete *JSON*-formaati väljastamise jaoks tagasiühilduv funktsioon.

Üheks eesmärgiks lisaks dialoogsüsteemi kaasajastamisele oli selle täiendamine paindlikuma moodulite süsteemiga. Moodulite toimimise jaoks on vaja aga sündmusi, millele moodulid saavad reageerida. Sündmusteks on näiteks teksti saabumine serverist või kasutaja nupuvajutus ja paljud teised. Selleks, et sündmuse toimumisel midagi juhtuks, tuleb enne tekitada kuularid, mis kinnituvad teatud sündmuste külge *jQuery* raamistiku *bind* funktsiooniga. Sündmuste esilekutsumiseks on *jQuery* raamistikus funktsioon *trigger*, mis käivitab kõik kuularid, kes teatud sündmust ootavad [8]. Moodulite süsteemi osas tehtud täiendustest saab lugeda peatükist "Moodulite süsteem".

Lisaks oli vaja tekitada juurde muutujad, mis hoiaksid sisse tulnud teksti ja vajadusel laseksid seda muuta enne, kui see kasutajale kuvatakse. Põhjuseks võib tuua moodulid, mis muudavad teksti, näiteks suure ja väikese algustähe moodul, mis kohandab teksti algustähte vastavalt kasutaja enda stiiliga. *JavaScript* on ühelõimeline keel, kui jätta arvestamata uus *HTML5 Web Workers API*, seega ei saa tekkida olukorda, kus rohkem kui üks funktsioon üritaks muutuja väärtust samal ajal seadistada.

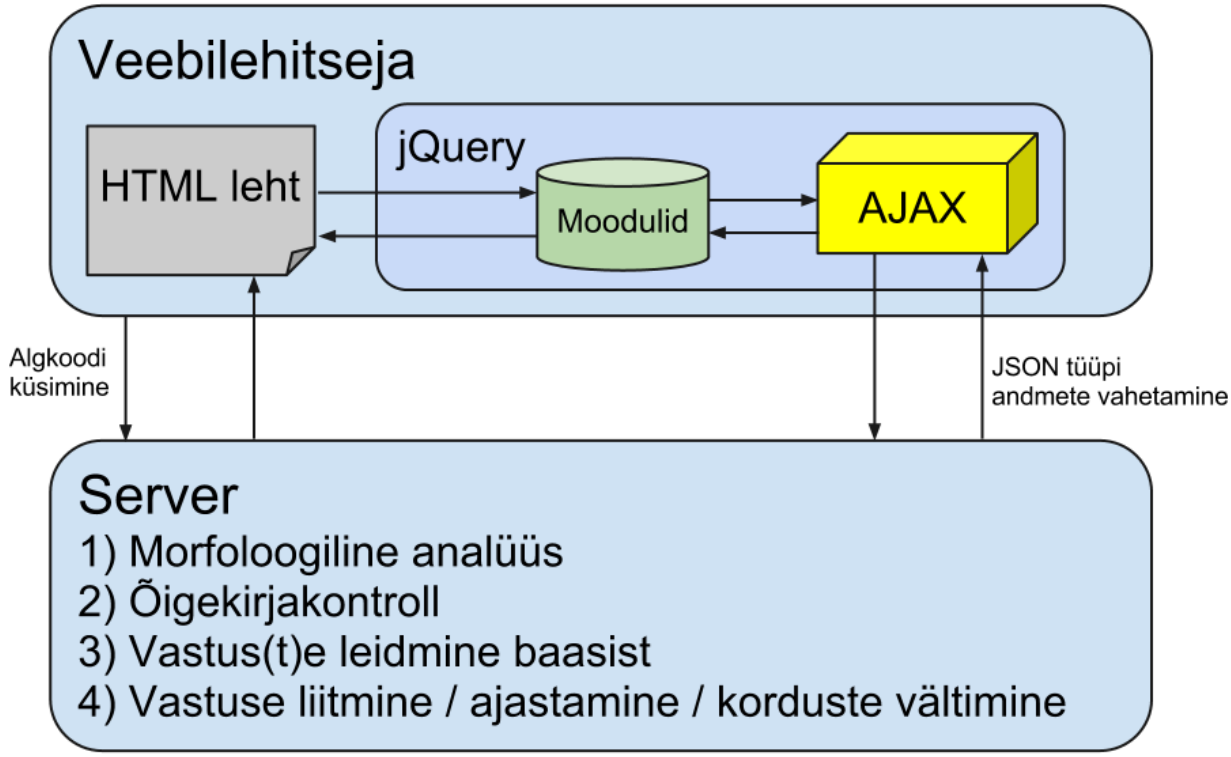
Suur osa kliendi koodist paiknes paljudes erinevates *JavaScript* failides, sellepärast tuli kasutusele võtta pakkimisprogramm Minify [9]. Minify võimaldab pakkida kõik projektis kasutatavad *JavaScript* failid üheks. Antud abivahendi korral on tagatud ka kooskõlalisus – st kui mõnda algfaili muudetakse, siis automaatselt uueneb ka kokkupakitud fail. Lisaks võimaldab Minify ka stiilifaile (CSS) pakkida, kuid hetkel selleks vajadust ei ole, kuna kasutusel on ainult üks stiilifail.

Viimane suurem muudatus puudutas dialoogsüsteemis kasutusel olevat morfoloogilist analüsaatorit, mille kasutajasõnastikku (.udr laiendiga fail) täiendati slängivormidega, mis saadi peamiselt Muischneki, Kaalepi ja Sireli [4] koostatud sõnastikust. See muudatus lisas süsteemile

võimaluse aru saada slängis kirjutatud sõnadest ja teisendada neid endale arusaadavateks sõnadeks.

3.1. Andmevoog

Andmevoo kirjeldamiseks on kasutatud lisaks tekstile joonise [Joonis 2] abi.



Joonis 2. Dialoogsüsteemi andmevoog

Esimeses etapis küsib veebilehitseja serverilt HTML lehe genereerimiseks algkoodi. Seejärel saadetakse vastu HTML lehe kood, mis sisaldab lisaks viiteid mitmetele failidele, mida süsteem oma töös kasutab nagu *JavaScript*, *CSS*[1] ja mõned pildifailid. Seejärel laetakse ka need. Pärast laadimist kuvatakse kasutajale kasutajaliides, mille abil saab kasutaja agendiga suhtlust hakata pidama. Moodulite ja *AJAX* koodi toetab *jQuery* raamistik. Kõik andmed suunatakse kasutaja

[1] CSS - *Cascading Style Sheets*, veebilehe erinevate komponentide stiili kirjeldused

sisestusest moodulitesse ja seejärel *AJAX* abil serverisse. Serverist tulevad vastu andmed *JSON* formaadis, mis on kliendile kõige optimaalsem.

3.2. Moodulite süsteem

Moodulite süsteem võimaldab luua dialoogsüsteemile juurde lisafunktsioone nagu näiteks helide mängimine sõnumite saabumisel, teha reaalajalist statistikat kasutaja harjumuste kohta ja palju muud. Seega moodulite süsteem võimaldab dialoogsüsteemi rikastada lisafunktsioonidega. Moodulid tuleb kirjutada enamasti programmeerimiskeeles *JavaScript*, kuid neid saab kergesti ka ühendada muude programmeerimiskeeltega, näiteks *PHP* skripte on võimalik käima panna, kui kirjutada *AJAX* päringuid, mis suhtleksid *PHP* skriptiga.

3.2.1. Struktuur

Moodulid asetsevad kaustas nimega *modules*, kus on iga mooduli jaoks veel omakorda endanimeline alamkaust. Moodul koosneb vähemalt kolmest failist: *index.php*, *settings.js* ja *source.js*.
Index.php sisaldab administraatorile suunatud lehte. *Settings.js* sisaldab *JavaScript* seadeid ning *source.js* sisaldab mooduli funktsionaalsust. Loodud moodulid pakitakse automaatselt kokku üheks *JavaScript* failiks *Minify* programmi abil.

3.2.2. Dialoogsüsteemi sündmused

Süsteemi kasutajaliides teatab sündmustest, mis on seotud enamasti teksti sisestamise ja saabumisega serverist. Neid sündmuseid on võimalik siduda enda loodud funktsioonidega.
Hetkel on olemas järgmised sündmused:

- *incoming* - sündmus, mis tähistab teksti saabumist serverilt, pärast seda hakkab agent "kirjutama"
- *agent_write* - sündmus, mis tähistab agendi teksti "kirjutamise" lõppemist ja kuvamist ekraanile
- *outgoing* - sündmus, mis tähistab kliendi teksti saatmist serverile

Rohkem sündmusi praegu realiseeritud ei ole, kuid neid saab vajadusel lisada. Igal sündmusel on kaasas hulk põhiparameetreid[1], lisaks on kaasas ka lisaparameetrid, mis on rohkem seotud antud sündmusega.

Incoming sündmuse lisaparameetriteks on *text* ja *actor*, millest esimene tähistab sissetulnud teksti algkujul ja teine tähistab agendi nime, näiteks Rudolf.
Agent_write sündmus lisaparameetreid ei sisalda.
Outgoing sündmuse lisaparameetriteks on *text*, mis sisaldab kasutaja poolt saadetavat teksti.

Kuulari loomiseks on vaja kasutada järgnevat koodi [10]:

```
$("#dialoogid").bind({
  'sündmus.use_mooduli_nimi': function(event){
  // loogika siia
  }
});
```

Kus tuleb asendada sündmus vastava sündmuse nimega eelnevast nimekirjast ja mooduli nime kasutamine on kohustuslik, et ei tekiks konflikte erinevate moodulite vahel.

3.2.3. Mooduli valmistamine

Mooduli valmistamise õpetuse leiab tööle lisatud LISA 1 alt.

3.2.4. Lisatud moodulid

Töö käigus lisati raamistikule järgnevad kolm moodulit.
Algustäht - moodul mõõdab kasutaja suurte ja väikeste tähtede suhet lause alguses ja vastavalt sellele muudab ka agendi lausete algused suurteks või väikesteks.
Helid - moodul lisab heli mängimise uue sõnumi kuvamisel, mis annab rohkem vestlusprogrammi tunde.

[1] võib lugeda täpsemalt http://www.w3.org/TR/2003/WD-DOM-Level-3-Events-20030331/ecma-script-binding.html

Vastamise kiirus - moodul arvutab kasutaja keskmise sisestuskiiruse ning selle põhjal mõjutab agendi vastamisaega.

4. Kasutajate tagasiside

Uuringus osales 15 kasutajat. Nad kasutasid enne küsitluse täitmist dialoogsüsteemi, millele olid lisatud kõik antud töös loodud moodulid. Uuringu eesmärgiks oli välja selgitada millised moodulid kasutajatele meeldivad. 87% vastanutest pidas end kiireks kirjutajaks ning 13% mitte. 53% kasutajatest jäi rahule dialoogsüsteemi vastamiskiirusega ja 47% pidas vastamist liiga aeglaseks ning neid, kelle jaoks vastused liiga kiiresti tulid ei olnudki. Sellest võib järeldada, et kasutajad eeldavad, et vastused peaksid neile tulema kiiremini, kui nad ise vastavad.

Helisid kuulsid 80% kasutajatest, 20% neist andsid teada, et helide esitamise võimalus nende arvutis puudus. 75% kasutajatest, kes kuulsid helisid, vastasid, et helisignaal oli abiks ning juhtis tähelepanu agendi vastamisele. 25% aga arvasid, et helid on häirivad. Seega helide moodulit võiks täiendada kasutajapoolsete seadistustega, mis võimaldaksid helid vajadusel kinni panna.

Algustähe kasutamise küsimusele vastati järgmiselt - 67% vastas, et kasutas lause alguses suurt algustähte ning 33% väikest algustähte. Süsteem kohanes nende algustähe stiiliga ning 87% kasutajatest arvas, et see meeldis neile ning 13% seda arvamust ei jaganud. Samas kasutajate vastused ei läinud kokku logidega. Nimelt 33% kasutajatest väitsid küsitluses, et kasutasid väikest algustähte. Tegelikkuses on see protsent aga kõvasti kõrgem. Lausa 60% kasutab järjepidevalt väikest algustähte. Mida siit järeldada? Siit võib järeldada, et kasutajad isegi ei märka sellist pisiasja ja arvavad, et nad on korrektselt kasutanud õigekirja. Nende jaoks on väikese algustähega alustamine nii loomulik.

Küsitluses otsustati mitte uurida kasutajate suhtumist slängi tuvastamisse, sest üldjuhul inimene ei erista ega mäleta, kas ta kasutas vestluses slängi või mitte. Vestluste logisid uurides selgus, et taolisi vorme antud korral ei kasutatud, mis olnuks kuidagi algselt tuvastamata jäänud ning

slängisõnastiku abil tuvastatud saanud. Seega hetkel jäi testimata slängisõnastiku mõju, kuid loodame, et tulevik näitab selle kasu peagi.

5. Edasiarendamise võimalused

Praeguses töös tehtud veebikihi täiendused on ettevalmistuseks tulevaste uute suhtlusfunktsioonide lisamiseks.

Näiteks nüüd on lihtne lisada dialoogsüsteemile:

- Vestlustegelase pildi muutumine (näoilme muutumine).
- Vestlusakna tühjendamine, kasutaja sooviavalduse korral („alusta uuesti“, „tee aken tühjaks“).
- Vestluse kiirendamine/aeglustamine kasutaja korralduse peale („räägi kiiremini“, "aeglasemalt") - *käesoleva seisuga juba realiseeritud*

Veel võiks olemasolevate moodulitele teha kasutajate arvamust arvestades täiendusi. Näiteks helide moodulile võiks lisada heli kasutajapõhise väljalülitamise võimaluse ning vastamise kiiruse mooduli kiirust tõsta natukene kõrgemaks kasutaja enda kirjutamise kiirusest.

Moodulite süsteemi arendustööde hõlbustamiseks võiks aga moodulite süsteemi täiendada järgnevalt:

- Loodud moodulite jaoks kataloog, kust saaks tõmmata alla moodulite lähtekoodi.
- Edasi arendada moodulite süsteemi võimeid, näiteks lisada administraatori paneelile parooliga kaitse, täiendada salvestatavate seadete tüüpide hulka.

Kokkuvõte

Lõputöö käsitleb dialoogsüsteeme. Dialoogsüsteem on arvutiprogramm, mis saab sisendiks kasutaja lause (antud juhul) eesti keeles ning vastab kasutajale samuti eesti keeles. Selle tulemusel moodustub sidus vestlus, mille käigus antakse inimesele üldjuhul infot teatud kitsa ainevaldkonna kohta.

Lõputöös on võetud aluseks ühe olemasoleva dialoogsüsteemi raamistik, mille autor on Margus Treumuth. Seeläbi tekkis võimalus keskenduda seniste tulemuste edasiarendustele ja kasutada praeguseks juba põhjalikult läbitöötatud komponente (sh morfoloogiline analüüs, õigekirjakontroll, sõnajärjekorra lahendamine, korduste vältimine, inim-abi liides), mille realisatsioon oleks suures osas juba olemasoleva kordus.

Muudeti olemasoleva raamistiku pealiskihti, et ta oleks paremini ühilduv lisamoodulitega. Senine raamistik on veebipõhine, mis kasutab vastuse leidmiseks serveripoolset andmebaasikihti. Pealiskihi all ongi mõeldud veebikihti ning veebikihi täienduste tulemusel muutus aluskihi (ehk peamiste keeletöötlusmoodulite) kasutamine arendajaile lihtsamini kättesaadavaks.

Olemasolevale raamistikule loodi moodulite süsteem, mis võimaldab lihtsalt ning kiirelt lisada ja eemaldada veebiliidese mooduleid. Lisaks loodi mõned moodulid, mida saab antud süsteemiga kasutada. Realiseeriti moodul, mis kohaneb kasutaja vastamise kiirusega ja muudab vastavalt sellele agendi vastamise kiirust. Peale selle loodi komponent, mis jälgib kasutaja algustähe kasutamist ning paneb agendi sarnast stiili kasutama. Viimasena koostati moodul, mis mängib agendi teksti lisamisel heli signaali ning selle mooduli valmistamise protsess on lisatud moodulite valmistamise õppematerjalina töö lõppu lisana.

Lisaks moodulitele, täiendati ka dialoogsüsteemi sõnastikku slängisõnastikuga. Släng tuvastatakse kasutaja sisendis, võrreldes seda slängisõnastikuga. Esmalt on täienduseks asjaolu, et kasutaja sisendis leiduv võimalik slängisõna teisendatakse teadmusbaasi jaoks arusaadavasse

vormi ja arvuti suudab anda vastuse. Peamiseks probleemiks on slängi tuvastamine sisendis, sest slängisõnastik ei pruugi sisalda kõiki vorme.

Veel said 15 inimest dialoogsüsteemi kasutada ja vastata küsimustikule ning tulemuseks oli see, et üldiselt kasutajatele meeldisid moodulid, kuid nad andsid väärtuslikku tagasisidet, mille põhjal saab ideid moodulite edasiarendamiseks. Küsitlusest ja vestluslogidest tuli välja, et 60% kasutajatest kasutab järjepidevalt lause alguses väikest algustähte.

A Prototype for an Adaptive Dialogue System

Bachelor thesis

Margus Ojanurme

Abstract

The bachelor's thesis discusses the adaptability issues in dialogue systems. The dialogue system is a computer program that receives user input in a sentence (in this case) in Estonian and corresponds to the user in Estonian as well. As a result, a coherent conversation is formed, during which people in general receive information on a narrow domain.

The work is modelled on an existing dialogue system framework made by Margus Treumuth. Thus I was able to focus on further development of current results and use now thoroughly developed components (including morphological analysis, spell checking, word order solving, repetition avoiding, the human interface for helping), which implementation would have been largely repeating a realisation of existing.

The frontend of the existing framework was modified to be more compatible with modules. The current framework is web-based, which uses server-side database to find the answers. Due to modifications on the frontend, backend functions became more available for developers to use.

The modules system was built and added to the existing framework, it allows for easy and fast integration of web interface modules. In addition, some modules were created to be used by this system. For example two modules that adjust with user style were created. The first one adjusts agent's response speed based on user's typing speed. The second one adjusts agent's response capitalisation based on user's input capitalisation.

In addition a slang dictionary was added to the dialogue system. Slang in the user input is detected by comparing this input to the slang dictionary. The main problem is finding the input of slang, as slang dictionary may not contain all the forms.

There were 15 people involved in testing the dialogue system and then they were asked to fill out the questionnaire. The answers showed that in general users liked the new modules and they provided valuable feedback, which can be used in further development of these modules. From the chat logs came out that 60% of users consistently use a small initial letter in the beginning of the sentence.

Kasutatud kirjandus

1. Raul Sirel. 2009. Kohaldatava teadmusbaasiga eestikeelne dialoogsüsteem. Bakalaureusetöö.

2. Margus Treumuth. 2011. A framework for asynchronous dialogue systems: concepts, issues and design aspects. Doktoritöö.

3. Joel Edenberg. 2010. Edelaraudtee tehisintellekt. Bakalaureusetöö.

4. Muischnek, K.; Kaalep, H.-J.; Sirel, R. (2011). Korpuslingvistiline lähenemine eesti internetikeele automaatsele morfoloogilisele analüüsile. Metslang, H.; Langemets, M.; Sepper, M.-M. (Toim.). Eesti Rakenduslingvistika Ühingu aastaraamat (111 - 127). Tallinn: Eesti Rakenduslingvistika Ühing

5. Jesse James Garrett. 2005. Ajax: A new approach to web applications. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications (kontrollitud 29.04.2012)

6. XML ja JSON näited. http://json.org/example.html (kontrollitud 29.04.2012)

7. PHP: json_encode - Manual. http://ee.php.net/json_encode (kontrollitud 29.04.2012)

8. jQuery API - .trigger(). http://api.jquery.com/trigger/ (kontrollitud 29.04.2012)

9. Combines, minifies, and caches JavaScript and CSS files on demand to speed up page loads. http://code.google.com/p/minify/ (kontrollitud 29.04.2012)

10. jQuery API - .bind(). http://api.jquery.com/bind/ (kontrollitud 30.04.2012)

Lisad

Lisa 1: Juhised lisamooduli valmistamiseks

Mooduli valmistamine.
Praeguses õpetuses võtame eesmärgiks luua mooduli, mis mängib kasutajale heli, kui agent lisab ekraanile teksti.

Esiteks tuleks kopeerida *modules* kaustas olev kaust *template* soovitud nimega uude kausta, hetkel kasutame selleks *notification_sounds* nime. *Template* kaust sisaldab meile vajaminevaid faile ja peaks uue mooduli loomise vaeva lihtsustama. Kuna *notification_sounds* kaust juba eksisteerib, sest selle õpetuse käigus loodav moodul on tegelikult juba olemas, siis võiks selle kausta liigutada *modules* kausta alt ära.

Edasi võime avada oma uue *notification_sounds* kausta ning võtta lahti sealt faili *settings.js*. *Settings.js* fail sisaldab seadeid, mida moodulite süsteem kasutab, hetkel on toetatud ainult üks valik, mis näitab kas moodul on lülitatud sisse või mitte.
Seega asendame praeguse rea
var use_template = true;
uue reaga
var use_notification_sounds = true;
tähtis on, et kausta nimi oleks pärast *use_*, muidu ei oska moodulite süsteem sellest aru saada.

Järgmisena võime võtta lahti *source.js*, selles failis asub meie mooduli loogika.
Hetkel näeme, et seal on järgmine kood

```
modules.push({
  name:"Template",
  source:"template"
});
$(document).ready(function() {
```

```
  if(use_template) {
    // Do stuff
  }
});
```

Name parameeter saab olema mooduli nimi, selle võib vabalt valida, kuid märgime hetkel selleks "Helid". *Source* parameeter näitab kausta, kuhu moodul kuulub, selleks on "notification_sounds". Kuna seadete *JavaScript* failis, oli meil muutuja *use_notification_sounds* siis asendame *use_template* muutuja sellega.

Nüüd on meil põhi üles seatud ja saame asuda mooduli loogika juurde. Koha näitab kätte kommentaar *Do stuff*. Sinna saame kirjutada oma loogika. Meid huvitab sündmus, kus agent lisab teksti, selleks sündmuseks on *agent_write*. Enne kui jõuame kuulari loomiseni on vaja helifaili, selle võib saada näiteks eelmisest *notification_sounds* kaustast, mille *modules* kaustast välja tõstsime. Kopeerime vanast *notification_sounds* kaustast *sound.wav* faili meie uude kausta. Teeme kuulari, mis kasutab sündmust *agent_write* ja selle toimumisel üritaks mängida heli.

```
$("#dialoogid").bind({
  'agent_write.use_notification_sounds': function(event){
    try {
      var snd = new Audio("modules/notification_sounds/sound.wav");
      snd.play();
    } catch (err) {
    // Sounds not supported in browsers
    }
  }
});
```

Kui nüüd kõik kokku panna, siis peaks *source.js* nägema välja järgmine

```
modules.push({
  name:"Helid",
  source:"notification_sounds"
});
$(document).ready(function() {
  if(use_notification_sounds) {
    $("#dialoogid").bind({
      'agent_write.use_notification_sounds': function(event){
        try {
          var snd = new Audio("modules/notification_sounds/sound.wav");
          snd.play();
        } catch (err) {
        // Sounds not supported in browsers
        }
      }
    });
  }
});
```

Hetkel ei ole rohkem vaja muudatusi koodile teha ja kõigi eelduste kohaselt peaks pärast dialoogsüsteemi uuesti laadimist veebilehitsejas seadete alla tekkima uus vaheaken nimega “Helid”.