Lints, T. (2005). Bakteri rakus toimuvate protsesside esitamine multiagentsüsteemina DnaA tiitrimise mudelil põhineva agentmudeli näitel. Master's thesis, Tallinn University of Technology, Estonia. In Estonian. 125 pages.

T. Lints, "Bakteri rakus toimuvate protsesside esitamine multiagentsüsteemina DnaA tiitrimise mudelil põhineva agentmudeli näitel," Master's thesis, Tallinn University of Technology, Estonia, 2005. In Estonian. 125 pages.

```
@mastersthesis{Lints05_MScThesis,
  author = {Taivo Lints},
  title = {Bakteri rakus toimuvate protsesside esitamine
multiagents\"{u}steemina {D}na{A} tiitrimise mudelil p
\~{o}hineva agentmudeli n\"{a}itel},
  year = {2005},
  school = {Tallinn University of Technology, Estonia},
  note = {In Estonian. 125 pages}
}
```

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Automaatikainstituut
Reaalajasüsteemide õppetool

Taivo Lints

Bakteri rakus toimuvate protsesside esitamine multiagentsüsteemina DnaA tiitrimise mudelil põhineva agentmudeli näitel

Magistritöö

Juhendaja:
prof. Leo Mõtus

Tallinn 2005

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

23. mai 2005

Taivo Lints

Taivo Lints

Bakteri rakus toimuvate protsesside esitamine multiagentsüsteemina DnaA tiitrimise mudelil põhineva agentmudeli näitel

Magistritöö

Annotatsioon

Käesolevas magistritöös on näidatud, et multiagentsüsteemid on sobiv vahend bakteri rakus toimuvate protsesside modelleerimiseks ja simuleerimiseks. Töö esimene pool annab laiema ülevaate: 1) modelleerimise olemusest; 2) arvutiteaduse arengust läbi erinevate paradigmade; 3) tarkvaralistest agentidest; 4) modelleerimisest bioloogias. Teises pooles on esitatud DnaA tiitrimise mudelil põhinev multiagentmudel DNA replikatsiooni ja raku pooldumise kohta: 1) kõigepealt on antud kokkuvõte varasematest relevantsetest töödest; 2) seejärel on kirjeldatud relevantseid bioloogilisi protsesse vastavalt Cooper-Helmstetter-Donachie mudelile ja initsiaatori tiitrimise mudelile; 3) ära on toodud kasutatud tarkvaralised vahendid, eelkõige JADE ja Java SDK; 4) on kirjeldatud loodud agentmudelit, mis hetkel koosneb nelja tüüpi agentidest: Keskkond, Bakter, DNA ja DnaA Tootja; 5) on esitatud simulatsioonide tulemused ja hinnatud nende adekvaatsust: leitud on kvalitatiivne vastavus reaalsete protsessidega.

Töö on kirjutatud eesti keeles ja koosneb 54 leheküljest põhitekstist (ühekordse reavahega), 62 leheküljest lisana antud lähtekoodist ja 9 muust leheküljest (viited, sisukord, käesolev annotatsioon jms.). Töö sisaldab 25 joonist.

Magistritöö tegemisele kaasa aidanud rahalised allikad:

- TTÜ Automaatikainstituut, kus töötan insenerina.
- Eesti Teadusfond: 2004. aastal olin grandi 4860 "Agendid ja nende käitumine reaalses maailmas" üks täitjatest ning 2005. aastal olen grandi 6182 "Multiagentsüsteemide uurimine heterogeenses, dünaamiliselt muutuva struktuuriga keskkonnas (Hopadhoc projekt)" üks täitjatest.
- Haridus- ja Teadusministeerium: olen sihtfinantseeritava teadusteema 0142509-s03 "Arukad komponendid ja nende ühendamise probleemid" üks täitjatest.

Taivo Lints

Using Multiagent Systems for Representing the Inner Processes of a Bacterial Cell, a DnaA Titration Model Based Agent Model as an Example

Master's Thesis

Abstract

In this thesis it is demonstrated that multiagent systems are a suitable tool for modelling and simulating the inner processes of a bacterial cell. The first half of the paper gives a broader overview of: 1) the essence of modelling in general; 2) the progress of computer science through different paradigms; 3) software agents; 4) modelling in biology. In the second half a DnaA Titration Model based multiagent model of DNA replication and cell division is given: 1) first the relevant earlier work is summarized; 2) then relevant biological processes are described according to the Cooper-Helmstetter-Donachie Model and the Initiator Titration Model; 3) used software tools are mentioned, most importantly JADE and Java SDK; 4) the created agent model is described, currently consisting of four types of agents: Environment, Bacterium, DNA and DnaA Factory; 5) the results from simulations are presented and their adequacy evaluated: a qualitative correspondence to real processes is noticed.

The thesis is written in Estonian and consists of 54 pages of main text (line spacing: single), 62 pages of source code in the appendix and 9 other pages (references, the table of contents, this abstract, and the like); it contains 25 figures.

The work on this thesis has been financially supported by:

- Department of Computer Control, Tallinn University of Technology, where I work as an engineer.
- Estonian Science Foundation: in 2004 I was one of the investigators under the research grant 4860 "Agents and Their Behaviour in Real World" and in 2005 I am one of the investigators under the research grant 6182 *"Multiagentsüsteemide uurimine heterogeenses, dünaamiliselt muutuva struktuuriga keskkonnas (Hopadhoc projekt)"* (Studying Multiagent Systems in Heterogeneous Environment with Dynamically Changing Structure (The Hopadhoc project)).
- Ministry of Education and Research of Estonia: I am one of the investigators under the research grant 0142509s03 "Intelligent Components and Their Integration Problems".

Sisukord

Jooniste nimekiri

1. Sissejuhatus

Teaduse ja tehnika areng on jõudnud parajasti sellisesse punkti, kus bioloogias on aktuaalseks muutunud organismide kui terviksüsteemide arvutiga modelleerimine ning tarkvaraarenduses ja eriti simulatsioonide loomisel on populaarsust kogumas multiagentsüsteemide kasutamine. On loogiline eeldada, et nende kahe suuna vahel tekivad sidemed ning multiagentsüsteeme hakatakse kasutama ka süsteemibioloogias. Selleks aga peavad need ideed kõigepealt kellegi peades kohtuma. Käesoleva magistritöö aluseks olevad mõtted tekkisid ajatundlike agentidega tegeleva Leo Mõtuse ja biokeemik Raivo Vilu vestluste käigus ning kuna 2003. aasta sügisel ei olnud ei neile ega minule teada ühtegi teist analoogset tööd, saigi magistritöö eesmärgiks seatud:

> Näidata multiagentsüsteemide sobivust
> bakteri rakus toimuvate protsesside modelleerimiseks-simuleerimiseks.

Kuna tegu on mitut teadusharu hõlmava teemaga, mis pealegi kasutab mitte veel väga levinud agentorienteeritud lähenemist, siis on töö esimeses pooles antud neist valdkondadest laiem ülevaade. Kohati on need peatükid üldisemad kui töö käigus loodud agentmudeli mõistmiseks vajalik, kuid sellel on oma põhjus. Nimelt ei ole käesolev magistritöö mitte eesmärk omaette, vaid pigem alus edasiseks põhjalikumaks tegevuseks bakteri modelleerimisel multiagentsüsteemina ja selles kontekstis tulekski esimesi peatükke vaadelda.

Töö koosneb järgnevatest osadest:

- Peatükis 2 käsitletakse mudelite ja modelleerimise olemust.
- Ptk-s 3 antakse ülevaade arvutiteaduse arengust, mis on viinud / viimas interaktsioonipõhiste mudelite laiema levikuni.
- Ptk 4 annab põhjalikuma ülevaate agentide olemusest ja kasutamisest.
- Ptk 5 kirjeldab lühidalt mudelite kasutamist bioloogias.
- Ptk 6 vaatleb käesoleva magistritöö seisukohast relevantseid töid rakus toimuvate protsesside modelleerimise alal.
- Ptk 7 kirjeldab loodud agentmudelit ja saadud tulemusi.
- Ptk 8 teeb kokkuvõtte,
- ja Lisas on antud agentmudeli lähtekood.

2. Mudelid ja modelleerimine

Mudelite loomine ja kasutamine on teaduses ning ka igapäevaelus niivõrd sagedane ja loomulik tegevus, et enamasti seejuures modelleerimise sügavama olemuse üle arutlema ei vaevuta. See on ka täiesti arusaadav, sest ülemäärane filosofeerimine kulutaks liigselt aega ning võiks arutleja niivõrd segadusse ajada, et reaalsed tegevused jääkski sooritamata. Kuid aeg-ajalt on siiski kasulik toimuvat ka distantsilt vaadelda ning analüüsida, loomulikuna tunduvaid protsesse "dekonstrueerida" ja teadvustada nende tegelikku olemust.

Veel 20. sajandi keskel oli mudel eelkõige (kuid mõistagi mitte eranditult kõigil juhtudel) *mudel teooria jaoks* – lihtsalt üks teooria interpretatsioonidest – omades heal juhul esteetilist, didaktilist (õpetlikku) või heuristilist väärtust, kuid olles teooria mõistmise ning eduka rakendamise seisukohast üsna vähevajalik (da Costa and French, 2000). Praegusel ajal on aga mudelite roll teaduses oluliselt kasvanud, muuhulgas ilmselt ka tänu arvutustehnika kiirele arengule. Seetõttu tuleb terminit *mudel* üha enam mõista kui *(reaalse) süsteemi mudelit*, mille eesmärgiks on teooria poolt süsteemi kohta tehtud eelduste ja oletuste konkretiseerimine, spetsifitseerimine ja/või ligikaudne realiseerimine. Järgnevas (s.t. kuni järgmise viiteni) on peamiselt Bailer-Jones (2002) põhjal täpsemalt kirjeldatud, kuidas tänapäeva teadlased suhtuvad mudelite kasutamisse teaduses. Nimetatud allikas kajastab eelkõige reaalteadlaste vaateid.

Kõigepealt peaks selgitama, mida üldse mõiste *mudel* hõlmab. Laiemas mõttes võib mudeliteks nimetada peaaegu kõike teaduse poolt loodavat kuni keerukate teooriateni välja. Tänu mudelite rolli kasvamisele ja nende kasutamise teadvustumisele on märgata sellise lähenemise laiemat levikut võrreldes varasema ajaga. Enamasti siiski peetakse mudelit teaduslikust teooriast veidi "madalamal" olevaks, kuigi selge piir nende kahe vahel puudub. Sellisest vaatepunktist lähtudes on mudelid teooriatest spetsiifilisemad – üldise teooria rakendused konkreetsematele juhtudele – ning sisaldavad märgatavaid lihtsustusi. Kui aga mudelit põhjalikult testida ja täiendada ning see üha enam ja enam tegelikkusega kooskõlas on, võib mudel muutuda teooriaks: hetkeseisuga parimaks teaduse poolt väljapakutavaks seletuseks, kus põhimõtteliselt võib küll lihtsustusi ja tahtlikke väljajätmisi leiduda, kuid need on võimalikult täpselt välja toodud ja "kontrolli all". Mudeli korral ei ole terminid "tõesus" ja "õigsus" paljudel juhtudel otseselt rakendatavad, sest mudel võib olla uurimistöö jaoks huvitav ja kasutuskõlbulik ka juhul, kui ta tegelikkusest väga tugevalt erineb või on lausa tahtlikult "valesti" loodud. Kui aga reaalsed protsessid toimuvad oluliselt teisiti võrreldes TEOORIA poolt välja pakutuga, siis öeldakse enamasti otse välja, et teooria on vale.

Teaduslik mudel on millegi esitus matemaatiliste võrranditena, väidete kogumina, visuaalselt, geomeetriliselt või muul viisil. Teaduslik mudel on eelkõige probleemi püstitus uurimist võimaldaval kujul ehk hüpoteesi esitamine formaalselt ja testitavalt. Mudeli loomisel kaasatakse eelkõige huvipakkuva süsteemi need aspektid, mis tunduvad olevat olulised, jättes (aja, raha ning teadmiste piiratuse tõttu) välja tõenäoliselt vähemtähtsad tegurid. Selles mõttes võib pidada heaks mudelit, mis esitab vaadeldava probleemi põhiolemuse, meid huvitava käitumise, nii lihtsalt kui võimalik. Seetõttu tihti ei olegi eriti oluline, kui täpselt teaduslik mudel suudab protsesse ette ennustada, vaid pigem see, mil viisil ja kui hea ettekujutuse uuritavast protsessist saab modelleerija. Näiteks väga keerukate ja suurt arvutusvõimsust vajavate arvutuslike mudelite korral võib modelleerimisprotsessi "peidetus" arvutitesse olla uurija jaoks tõsiseks probleemiks. Teisalt on siiski ka täpsus reaalsuse kirjeldamisel ning ennustamisel üks mudeli

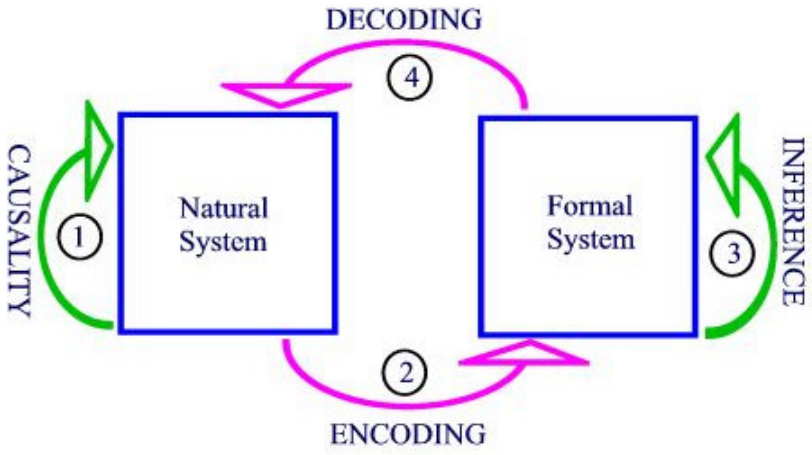
headuse mõõdupuudest. Ja kui mudel muutub nii täpseks, et näeb (näiteks arvuti-simulatsioonina) juba välja nagu modelleeritav süsteem ise, tekivad uued huvitavad probleemid ja küsimused. Probleemina võib nimetada olukorda, kus liigselt hägustub mudeli ja reaalsuse tegelik vahekord ning reaalset protsessi hakataksegi käsitlema ainult mudelist saadud ettekujutuse põhjal, unustades mudeli loomisel tehtud arvukad teadlikud ja teadmatud lihtsustused. Huvitava küsimusena aga kerkib esile simulatsiooni ja reaalsuse vahekord laiemalt: kuidas ikkagi suhtuda neisse (arvutis) simuleeritud protsessidesse? Intrigeeriva võimaluse pakub välja Grand (1997):

> Niisiis, siin on minu "suur väide" *(big claim)*: nähtuse otsene arvutisimulatsioon ei ole selle nähtuse eksemplar, ükskõik kui sarnane ta sellele paistab (algoritm, mis otseselt simuleerib osakese liikumist, ei oma ise massi ega inertsi). Aga sellistest simuleeritud ehitusplokkidest kokku pandud metanähtus on põhimõtteliselt eristamatu samast metanähtusest, mis on kokku pandud niiöelda "reaalsetest" ehitusplokkidest – nad asuvad erinevates universumites, kuid on ekvivalentsed. Simuleeritud aatomid ei ole aatomid. Simuleeritud aatomitest ehitatud molekulid on molekulid. Simuleeritud molekulidest tõetruult konstrueeritud organismid on elusad.
>
> Seega eristan ma üksteisest esimest järku ja kõrgemat järku simulatsioone. Esimest järku simulatsioonid ei ole nende poolt simuleeritava nähtuse eksemplarid, kuid kõrgemat järku simulatsioonid (mis eksisteerivad simuleeritud universumis) on. Esimest järku simulatsioon on lihtsalt algoritm. Teist järku simulatsioon on samuti (tingimata) algoritm, kuid samal ajal on ta nähtus. Üks paradigma vaatleb arvutit kui "töötlusseadet" *(processing device)*, samal ajal kui teine kasutab seda kui "kohta asjade ehitamiseks", ja need on põhimõtteliselt erinevad kontseptsioonid. Paljusus mängib siin olulist rolli – võtmetähtsusega on ilmselt esimest järku algoritmidest konstrueeritud virtuaalsete objektide populatsioonid.
>
> (Kuidas Stephen Grand selle ideeni jõudis, on humoorikal moel kirja pandud viidatud artiklis (Grand, 1997), mida on mõtlemise ergutamise ja alternatiivsete maailmavaadetega tutvumise seisukohast kasulik lugeda ka juhul, kui sealtoodud väidetega nõus ei ole).

Modelleerimisprotsessi võib kirjeldada mitmetest erinevatest vaatenurkadest ja erineva põhjalikkusega ning suur hulk teadusfilosoofe ja teiste teadusalade esindajaid on seda ka teinud. Väljapakutud arutlustest põhjaliku ülevaate tegemine on siinkohas mõeldamatu, kuid näitena on järgnevas Mikulecky ja Gwinn' i kirjutiste põhjal lühidalt vaadeldud Robert Rosen' i kirjeldust modelleerimisest (nn*Rosen's Modeling Relation*).

Tegelikku, reaalselt eksisteerivat süsteemi nimetab Rosen *materiaalseks süsteemiks (material system). Looduslik süsteem* (või *loomulik süsteem*, *(natural system)*) on materiaalse süsteemi subjektiivselt defineeritud alamhulk või abstraktsioon, mis põhineb meie sensoorsetel tajudel ning mõtteprotsessidel. Seega juba looduslik süsteem on uurija poolt aktiivselt valitud: ei ole mingeid *a priori* juhiseid, mis meile tegelikus maailmas süsteemi piirid täpselt kätte näitaks. Meie meelte kaudu ajusse jõudvate tajuühikute *(percepts)* vahel moodustuvad automaatselt seosed, mistõttu tunnetame *põhjuslikkuse (causality)* olemasolu universumis (joonis 1, vasakul).

Üritades vähendada vaatleja rolli, proovib teaduslik meetod muuhulgas vähendada meelte subjektiivsust, kasutades mõõteriistu ning omistades tajutavale kvantitatiivseid väärtusi. Seejuures tuleks aga tähele panna, et nimetatud meetodil on ka kõrvalmõjusid:



Joonis 1. Modelleerimisprotsess Robert Rosen' i pilgu läbi (Gwinn).

teadusliku uurimise alt kipuvad välja jääma nähtused, mida on raske kvantitatiivselt väljendada ja millega tegelemisel on raske saavutada "objektiivsust".

Formaalne süsteem on inimese loodud ning koosneb lõplikust hulgast väidetest ja neile mõjuvatest reeglitest. Formaalne süsteem töötab täielikult oma tuletusmootori abil ning ei oma iseenesest mingeid viiteid välistele süsteemidele.

Järelikult on nii looduslik kui formaalne süsteem iseseisvad: nad ei sisalda midagi, mis meile otseselt ütleks, kuidas neid omavahel suhestada. Seetõttu on modelleerimine alati ka "kunst", mitte ainult puhas teadus: modelleerija valib ise, millised formaalse süsteemi elemendid panna vastavusse loodusliku süsteemi elementidega. Seejuures üritab ta mõistagi saavutada nende kahe süsteemi omavahelist kooskõla, et valitud seosed ei viiks vastuoludeni formaalse ja loodusliku süsteemi omaduste vahel. Niisiis on modelleerija jaoks oluline, et joonisel 1 toodud diagramm "kommuteeruks", sümboolselt väljendudes: (1) = (2) + (3) + (4). Ehk teisisõnu: me *kodeerime* loodusliku süsteemi formaalseks süsteemiks, proovides säilitada kooskõla kahe süsteemi vahel; formaalses süsteemis viime vastavalt tuletusreeglitele läbi teisendused, saades tulemuseks ennustused loodusliku süsteemi kohta; *dekodeerime* need ennustused ja võrdleme looduslikus süsteemis toimuvaga. Tuleb aga jällegi meeles pidada, et kuna kodeerimine ja dekodeerimine sisaldavad endas subjektiivset komponenti, "kunsti", siis diagrammi kommuteeruvuse määrab uurija, mitte mudel.

Kui leiame loodusliku ja formaalse süsteemi vahel sellise vastavuse, et võime teha ennustusi loomuliku süsteemi kohta ilma seda kõigepealt kodeerimata (s.t. puudub alumine nool eeltoodud diagrammil), siis on tegu pigem metafoori kui mudeliga. Kui (arvuti)simulatsioon on valmistatud formaalse mudeli põhjal, kuid saadud tulemusi võrreldakse otse loodusliku süsteemiga, siis võib ka sellist simulatsiooni nimetada metafooriks, kuna puudub otsene kodeerimine looduslikust süsteemist simulatsiooni.

Nagu juba mainitud, on Robert Rosen' i käsitlus mudelitest vaid üks paljudest erinevate autorite poolt väljapakututest. Siin eraldi väljatoomiseks sai ta valitud eelkõige seetõttu, et on leidnud suhteliselt laialdast vastukaja ning selle käsitluse väljatöötamisel on lähtutud ka bioloogiliste ja teiste keerukate süsteemide problemaatikast (mis siintoodud kirjeldusest eriti selgelt välja ei paista, kuna tegu on vaid äärmiselt põgusa ülevaatega Rosen' i ideedest).

3. Arvutiteaduse areng

Arvutiteadus on suhteliselt noor ala ja areneb jätkuvalt suure kiirusega. Hoolimata oma uudsusest on ta juba saavutanud väga olulise rolli ühiskonnas, kiirendades järsult info liikumist ja töötlemist. Ehk nagu Steve Jobs juba ca. 1980. aastal ajakirjas *Scientific American* avaldatud *Apple*' i reklaamis piltlikult väljendus: "Personaalarvuti on jalgratas mõistuse jaoks." (Hertzfeld). Kuna arvutite kasutamine omab järjest suuremat tähtsust kõigi teadusalade arengus, siis mõjutavad arvutiteaduse paradigmad – maailmavaated, teadmiste organiseerimise ja kasutamise viisid – teataval määral kogu teaduslikku mõtlemist (olles samas ka ise teistest uurimisvaldkondadest mõjutatud, kas siis otsese analoogia näol: simuleeritud lõõmutamine füüsikast, geneetilised algoritmid ja geneetiline programmeerimine evolutsiooniteooriast, tehisnärvivõrgud neuroloogiast, tehisimmuunsüsteemid immunoloogiast, L-süsteemid taimebioloogiast, sipelgakoloonia-optimiseerimine looduslike sotsiaalsete võrkude uurimisest (Stepney *et al.* 2004); või ka kaudsemalt, mida on kirjeldatud käesoleva peatüki teises pooles). Nimelt, kasutamaks arvutit konkreetse probleemi uurimiseks, tuleb see esitada arvutile mõistetaval kujul, mis sunnib teadlast kohandama oma mõttelist mudelit arvutis kasutatavale esitusviisile sobivamaks. Sellise protsessi sagedasel kordumisel võib teadlase mõttemudelite loomise võime muutuda püsivalt piiratumaks, nii et esialgu arvuti poolt peale sunnitud modelleerimisviisi kasutatakse ka väljaspool arvutiga suhtlemist. Seetõttu on nende mõtteviisidega tutvumine ja nimetatud ohu teadvustamine oluline mitte ainult kitsale arvutiteadlaste ringile, vaid tervele uurijate kogukonnale.

"Paradigma" on termin, mis kõlab huvitavalt, kuid mille täpne sisu jääb tihti veidi ebaselgeks, mistõttu seda kasutatakse üsna erinevatel viisidel ja mitmesugustes kontekstides. Näiteks on paradigmadeks nimetatud erinevaid lähenemisi arvutiarhitektuurile, arvutusmudelitele ja programmikoodi täitmise viisidele. Üks kõige laiemalt levinud mõiste, mis seda terminit kasutab, on ilmselt "programmeerimise paradigmad" ehk tarkvara struktureerimise meetodid (mida mõnikord nimetatakse ka tarkvara dekompositsiooni meetoditeks, kuid kuna mõistet dekompositsioon kasutatakse tihti ka veidi teistsugustes tähendustes, näiteks täpse matemaatilise definitsiooni alusel, siis segaduse vältimiseks on järgnevas eelistatud mõistet *tarkvara struktureerimine*). Hea lühiülevaate programmeerimise põhimõtete muutumisest alates esimestest arvutitest kuni objektorienteerituseni annab Stroustrup (1988) ja järgnevas ongi seda ülevaadet kokkuvõtlikult refereeritud. Seejuures tuleb tähele panna, et uued mõtteviisid mitte ei asendanud eelnevaid, vaid pigem täiendasid neid – varasemad lähenemisviisid leidsid oma koha uuemate sees. Programmeerimiskeelte paigutamine konkreetsete paradigmade alla lähtub Stroustrup' il põhimõttest: keel *loetab* vastavat programmeerimisstiili, kui ta sisaldab vahendeid, mis muudavad selle stiili kasutamise mugavaks (piisavalt kergeks, turvaliseks ja efektiivseks). Kui paradigmast lähtuva programmi kirjutamine nõuab erilisi jõupingutusi või oskusi, siis keel mitte ei toeta, vaid ainult *võimaldab* vastava stiili kasutamist.

Kõige esimene programmeerimisparadigma oli protseduuriline programmeerimine, kus peatähelepanu pöörati protseduuri disainimisele (arvutuste läbiviimiseks vajaliku algoritmi loomisele), ehk: *"Otsusta, milliseid protseduure vajad; kasuta parimaid algoritme, mida suudad leida."*. Programmikoodis korra loomiseks kasutatakse funktsioone. Peamised arutlusteemad on: kuidas funktsioonile argumente edastada, kuidas eristada erinevat liiki argumente, ja kuidas eristada erinevat liiki funktsioone (protseduurid, rutiinid, makrod jms.). Protseduurilisi keeli: Fortran, Algol-60, Algol-68, C, Pascal.

Aastate möödudes rõhuasetused muutusid ning tähelepanu pöördus andmete organiseerimisele, eesmärgiks sai andmete peitmine. Omavahel seotud protseduure ja nende poolt manipuleeritavaid andmeid nimetatakse mooduliks ja paradigma sisu on: *"Otsusta, milliseid mooduleid vajad; tükelda programm nii, et andmed oleks peidetud moodulitesse."*. Programmi struktureerimist mooduliteks toetab näiteks Modula-2.

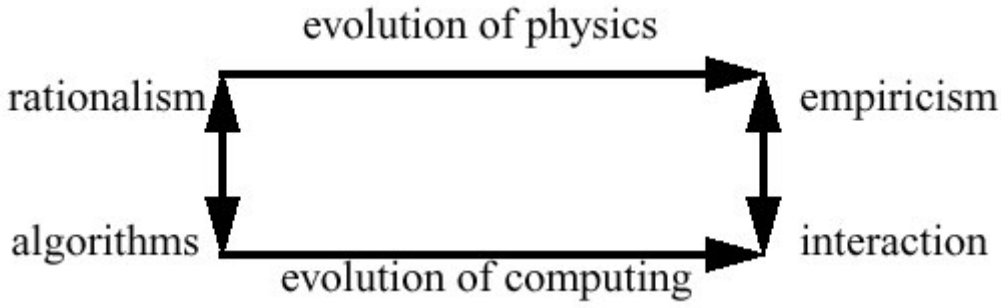
Enamasti antakse moodulite kasutamisel kontroll konkreetset tüüpi andmete üle vastava andmeliigiga tegeleva mooduli kätte. Sel viisil realiseeritud "andmetüübid" on aga selgelt erinevad keelde sisseehitatud andmetüüpidest: puudub ühtne lähenemine identifikaatorite omistamisel, ei kehti tavalised argumentide üleandmise ja muutujate skoobi reeglid. Probleemile pakkus lahenduse uus lähenemisviis, andmete abstraktsioon, kus kasutajale antakse võimalus defineerida andmetüüpe, mis käituvad (peaaegu) nagu sisseehitatud tüübid. Sellist andmetüüpi nimetatakse tavaliselt *abstraktseks andmetüübiks*. Paradigmaks saab: *"Otsusta, milliseid andmetüüpe vajad; varusta iga tüüp täieliku operatsioonide komplektiga."*. Andmete abstraktsiooni toetavad näiteks Ada, Clu, C++.

Kasutaja defineeritud andmetüübi puuduseks on vähene paindlikkus: ainus viis teda uute kasutusjuhtude jaoks kohandada on otsene definitsiooni modifitseerimine. Sellele probleemile lahenduste otsimise tulemusena sündis uus paradigma, objektorienteeritud programmeerimine, mis pakkus välja mõisted *klass* ja *pärilikkus*: kasutaja poolt loodud andmetüüp defineeritakse klassina ning selle andmetüübi kohandamisel spetsiifilisema kasutusjuhu jaoks tuletatakse defineeritud (baas)klassist uus (tuletatud/alluv/alam)klass. Paradigma sisu: *"Otsusta, milliseid klasse vajad; varusta iga klass täieliku operatsioonide komplektiga; too ühised omadused ilmutatult välja, kasutades pärilikkust"*. Objektorienteeritud keeli: Simula, C++.

Paradigmade tutvustus selles Stroustrup' i 1988. aasta artiklis lõpebki objektorienteeritud programmeerimisega. Nüüd, 17 aastat hiljem, esitatavaid samateemalisi ülevaateid vaadates selgub, et ka need lõpevad enamasti samas kohas ja uut sellise kaalukategooria paradigmat ei ole veel peale tulnud. Et võimalikku järgmist olulist tarkvara struktureerimise meetodit ette aimata, on kasulik kõigepealt vaadelda arvuti- ja ka teiste teaduste arengutrende.

Wegner (1999) toob välja paralleeli arvutiteaduse ja füüsika arengu vahel (joonis 2). Newton' i füüsika käsitleb uuritava süsteemi vaatlejat süsteemivälisena ja lähtub absoluutsetest (universaalsetest) ruumi ning aja mõistetest. Relatiivsusteooria seisukohalt kuulub ka vaatleja süsteemi ning omadused nagu mass, pikkus ja samaaegsus sõltuvad vaatleja taustsüsteemist, kuid põhjuslikkus ning aegruumi geomeetria on siiski sõltumatud, absoluutsed, reaalsusesse kuuluvad. Kvantfüüsika aga ei pea eeldust, et on olemas sõltumatu absoluutne reaalsus, üldse vajalikuks ning leiab, et vaadeldavad sündmused ei vaja põhjuslikke seletusi ning seejuures ka vaatlusakt ise avaldab uuritavale süsteemile mõju. Seega on füüsikute mõtteviis, mis esialgu oli valdavalt ratsionalistlik, muutunud aja jooksul empirismi (kogemuslikkuse) suunas.

Arvutite korral on vaadeldava objekti rollis mõni arvutis asuv olem ja vaatlejaks mõni teine olem või arvutikasutaja. Tüüpilise deterministliku vaatlusakti puhul antakse algoritmile algparameetrid, algoritm töötleb neid, lõpetab tegevuse ja väljastab tulemuse. Sama tegevuse kordamisel saadakse sama tulemus. Keerulisemal juhul aga on arvutisüsteemiga võimalik suhelda kogu töö ajal (kusjuures konkreetset töö lõpetamist ei



Joonis 2. Füüsika ja arvutiteaduse arengu sarnasus (Wegner, 1999).

pruugigi olla ette nähtud) ning seda võivad teha paljud vaatlejad korraga. Sellise interaktsioonilise arvutisüsteemi käitumine ei ole enam taandatav algoritmilisele "ratsionalistlikule" kujule. (Märkus: eeltoodu on ainult ligikaudne ja oluliselt lihtsustatud variant Wegner' i poolt esitatust, täpsemat infot saab eelpool viidatud artiklist).

Goldin ja Wegner (2004) täheldavad, et praktikas muutusid arvutid tugevalt interaktsioonilisteks juba üsna ammu, kuid suures osas arvutiteadusest on siiani valdav klassikaline puhtalgoritmiline lähenemine, mis põhineb nn. "Turing' i teesi müüdil" (kuigi probleemile juhiti tähelepanu ka teoreetilise poole pealt juba 1970. aastate lõpus). Tegelik Turing' i tees väidab: "Kui leidub töötav meetod (algoritm) matemaatilise funktsiooni väärtuste leidmiseks, siis on võimalik seda funktsiooni arvutada Turing' i Masinaga.". Mitmesugustel artiklis (Goldin and Wegner, 2004) väljatoodud põhjustel on seda teesi aga hakatud tõlgendama märksa laiemalt: "Turing' i Masin suudab teha (arvutada) kõike, mida suudab teha arvuti.", kuigi Turing ise nii ei arvanud, näidates lisaks algoritmilisele Turing' i Masinale ka teistsugust arvutusmudelit, interaktsioonilist valikumasinat *(choice machine)*, mis ei ole taandatav Turing' i Masinaks.

Viimasel ajal pööratakse interaktsioonidele järjest suuremat tähelepanu siiski ka arvutiteaduses, paljuski ilmselt praktilistest vajadustest lähtuvalt. Esiteks on arvutisüsteemid ise muutunud väga keerulisteks ja suurest hulgast komponentidest koosnevateks. Teiseks uuritakse paljudes teadusharudest järjest kasvava huvi ja põhjalikkusega keerulisi kohanevaid süsteeme *(complex adaptive systems*; loomulikult on *CAS*' ideks liigitatavaid süsteeme uuritud kogu aeg, aga nüüd tehakse seda rohkem ilmutatult, otsides seejuures ühiseid omadusi erinevatest valdkondadest pärit *CAS*' ide vahel). Keeruliste kohanevate süsteemide modelleerimisele-simuleerimisele aga oodatakse senisest paremat tuge arvutitelt. Tekkinud on suurt osa teadust puudutav ning interdistsiplinaarset koostööd soodustav vajadus uute teadmiste ja teooriate järele. Vaja on pragmaatilisi teooriaid järgmiste nähtuste ja süsteemide jaoks: ilmnevad omadused *(emergent properties)*, areng ja kasvamine, dünaamilised heterogeensed struktureerimata avatud võrgud (Milner and Stepney, 2003).

Neid arengutrende arvesse võttes võib ennustada, et järgmises olulisemas programmeerimisparadigmas on senisest suurema tähelepanu all nii interaktsioonid kui ka suhtlevate komponentide autonoomsem käitumine võrreldes ainult käske täitvate objektidega. Üks toodud tingimustele vastav ning järjest kasvava populaarsusega lähenemine tarkvara struktureerimisele on agentorienteeritud programmeerimine, või ka laiemalt agentorienteeritud lähenemine uuritavale probleemistikule. Sellest mõtteviisist tulebki juttu järgmises peatükis.

4. Ülevaade agentide olemusest ja kasutamisest

Nii täielikult tarkvaraliste kui ka hübriidsete agentide kasutamine erinevates valdkondades näitab tugevat tõusutendentsi. Käesolev peatükk annab põgusa ülevaate agentorienteeritud lähenemisest probleemide lahendamisele, seletades lahti agendi mõiste ja olemuse ning näidates agentide kasulikkust neile sobivates rakendustes. Et vältida ebarealistlike lootuste teket ning juhtida tähelepanu võimalikele probleemidele, on lõpuosas välja toodud ka agentorienteeritud tarkvaraarenduse ohud.

4.1. Agent ja tema keskkond

4.1.1. Agendi mõiste

Terminit *agent* kasutatakse paljudel erinevatel elualadel, muuhulgas luuretegevuses ja keemias, ning ka arvutiteaduses leiab ta järjest laiemat kasutust. Kuna aga sellel alal on *agent* alles suhteliselt uudne mõiste, siis ei ole veel välja kujunenud tema väga täpset määratlust ning erinevad uurijad võivad agendi all mõista veidi erinevaid asju. Sellegipoolest saab leida mõne üldisema määratluse, mis hõlmab enam-vähem kõiki termini *agent* erinevaid definitsioone arvutiteaduses.

Wooldridge (1998) ja Jennings and Wooldridge (1998) nimetavad agendiks süsteemi (enamasti arvutisüsteemi), millel on järgmised omadused:

- Autonoomsus – agent peaks olema võimeline tegutsema ilma inimese või teiste agentide otsese juhtimiseta. Võib öelda, et agent kapseldab endasse oleku (sarnaselt objektile programmeerimiskeeltes) ning lisaks sellele ka käitumise – erinevalt objektist määrab agent ise, kas ja mida ta teeb. Seetõttu ei öelda, et üks agent kutsub välja teise agendi mõne meetodi / funktsiooni / tegevuse, vaid et agendilt palutakse / nõutakse mingi tegevuse sooritamist. Seejärel agent ise otsustab, kas palve / nõue täita või mitte.
- Reaktiivsus e. reageerimisvõime – agent asub keskkonnas (näiteks arvutikeskkonnas koos teiste agentidega või reaalses füüsilises maailmas) ja on võimeline selle keskkonna muutustele reageerima.
- Proaktiivsus – agendi käitumine ei seisne pelgalt reageerimises välisteguritele, vaid ta on võimeline ise initsiatiivi üles näitama, tegutsema omaenda eesmärkide saavutamise nimel.
- Sotsiaalsus – agent peaks olema võimeline suhtlema teiste agentidega (teatavatel juhtudel ka inimestega), et saavutada enda eesmärkide täitmist ning aidata ka teisi agente. Tüüpilised sotsiaalse tegevuse vormid agentidel on läbirääkimised ning (seejärel) ühine probleemi lahendamine.

Mõnevõrra täpsema määratluse annab Ferber (1999), defineerides agenti kui füüsilist või virtuaalset olemit / objekti *(entity)*:

- mis on võimeline oma keskkonnas tegutsema,
- mis suudab teiste agentidega suhelda,
- mida juhib hulk püüdlusi *(tendencies)* kas isiklike eesmärkide kujul või siis rahulolu / ellujäämise funktsiooni näol, mida ta üritab optimeerida,
- mis omab isiklikke ressursse,
- mis on võimeline oma keskkonda tajuma (kuid piiratud määral, mitte tervikuna),
- mille ettekujutus sellest keskkonnast on ainult osaline (või puudub üldse),
- mis omab oskusi ja suudab pakkuda teenuseid,

- mis võib olla suuteline ennast taastootma / paljunema,
- mille käitumine on suunatud oma eesmärkide saavutamisele, võttes arvesse tema käsutuses olevaid ressursse ja oskusi ning sõltudes tema tajudest, ettekujutustest ning kommunikatsioonist.

4.1.2. Agentide klassifikatsioon

Kui eelnev andis agendi määratluse üldjuhul, siis tihti on kasulik agente mõne omaduse alusel klassifitseerida. Üheks levinud mõõdupuuks on agendi "intelligentsus" (Ferber, 1999):

- Kognitiivne (teadmuslik-tunnetuslik) agent – omab ettekujutust teda ümbritsevast keskkonnast ja teistest agentidest. Tal on mahukas teadmusbaas, isiklikud eesmärgid ja plaanid nende eesmärkide saavutamiseks ning ta on suhteliselt keerukas ja üsna iseseisev üksus.

- Reaktiivne agent – ainult reageerib välistele mõjuritele, plaane ei tee. On suhteliselt lihtne ja "rumal" üksus, aga suure hulgana koos tegutsedes võivad reaktiivsed agendid moodustada edukalt funktsioneeriva süsteemi (näiteks sipelgaid võib mingis lähenduses vaadelda reaktiivsete agentidena, mis koos tegutsedes moodustavad keeruka ja funktsionaalse sipelgakoloonia).

Samas tuleb tähele panna, et toodud kaks kategooriat ei ole mitte üksteisest isoleeritud täiesti erinevad agendiklassid, vaid pigem sama skaala kaks otsa. Enamikel juhtudel asub parim lahendus hoopis kusagil kahevahel (joonis 3).

Joonis 3. Kognitiivsus ja reaktiivsus.
Telje all on näidatud, millist info esitusviisi vastavad agendid kasutavad oma teadmiste töötlemisel ning säilitamisel. (Ümberjoonistus allikast (Ferber, 1999))

Agentide klassifitseerimisel võib vaadelda ka nende suhet keskkonnaga (Ferber, 1999). Arvutikeskkonnast (rakendusprogrammide kogum, võrgud, heterogeensed süsteemid) võib leida puhtkommunikatiivseid agente, kelle väljapoole suunatud tegevus seisneb ainult suhtlemises teiste agentidega. Reaalses füüsilises maailmas (või selle simulatsioonis) asuvad agendid võivad aga olla vastupidise olemusega: kogu tegevus suunatakse keskkonnale (liikumine, objektide liigutamine-muutmine jms.). Viimasel juhul ei ole muidugi välistatud ka suhtlemine teiste agentidega, aga see toimub enamasti kaudselt – keskkonna muutmise ja nende muutuste tajumise teel. Puhtkommunikatiivsed agendid paiknevad joonisel 3 toodud skaalal tavaliselt vasakul (kognitiivsel) ning füüsilised paremal (reaktiivsel) poolel.

4.1.3. Agent ja keskkond

Kuna keskkond ja selles asuvad agendid on lahutamatult seotud, siis tuleks ka keskkonna omadustele piisavalt tähelepanu pöörata. Russell ja Norvig (1995) nimetavad järgmisi olulisemaid punkte, mida silmas pidada:

- Ligipääsetav *vs.* mitteligipääsetav. Keskkond on agendile ligipääsetav juhul, kui agent saab oma sensorite abil kindlaks teha keskkonna oleku. Ligipääsetavus võib olla täielik või osaline, viimane omakorda piisav või ebapiisav sõltuvalt sellest, kas agent saab oma otsuste tegemiseks vajaliku informatsiooni kätte või ei saa. Raskesti ligipääsetava keskkonna korral peab agendil olema piisavalt põhjalik sisemine ettekujutus oma maailmast, nii et ta suudaks ka vähesest infost piisavalt täpselt tuletada keskkonna hetkeoleku ning tänu sellele mõistlikult käituda.
- Deterministlik *vs.* mittedeterministlik. Keskkond on deterministlik juhul, kui tema järgmine olek on täielikult määratud praeguse oleku ning agentide poolt sooritatavate tegevustega. Mittedeterministlikkus võib olla keskkonna enda omadus, aga ka tuleneda mitteligipääsetavusest – kui agendil ei ole keskkonna kohta piisavalt informatsiooni, võib selle käitumine näida ettemääramatuna.
- Episoodiline *vs.* mitteepisoodiline. Episoodilises keskkonnas võib agendi tegevuse jagada nn. "episoodideks", mis koosnevad agendi poolt keskkonna tajumisest ning seejärel vastavalt saadud infole sobiva tegevuse sooritamisest. Need episoodid on omavahel sõltumatud ning seetõttu tegevuse edukus sõltub ainult konkreetsest episoodist, mis omakorda vähendab oluliselt nõudmisi agendi keerukusele – ei ole vajadust midagi ette planeerida.
- Staatiline *vs.* dünaamiline. Keskkond on konkreetse agendi jaoks staatiline, kui keskkonna muutused ei sõltu ajast, vaid ainult selle agendi tegevusest. Kui aja kulgedes keskkond ei muutu, küll aga agendi tegude efektiivsus (näiteks hiljem sooritatu võib olla vähem efektiivne), nimetatakse keskkonda pooldünaamiliseks.
- Diskreetne *vs.* pidev. Keskkond on agendi jaoks diskreetne, kui eksisteerib lõplik arv võimalikke sisendväärtusi sensoritelt ja samuti lõplik arv võimalikke keskkonnale suunatud tegevusi (näiteks malemängu korral on igal ajahetkel piiratud arv lubatud käike).

Enamasti kehtib reegel: mida komplitseerituma keskkonnaga on tegu, seda keerukam peab olema ka agent, et edukalt toime tulla. Raskeim olukord on ilmselt juhul, kui keskkond on mitteligipääsetav, mittedeterministlik, mitteepisoodiline, dünaamiline ja pidev.

4.1.4. Agendipõhine süsteem

Agentidel põhinev süsteem võib sisaldada mistahes nullist erineva arvu agente. Enamikel juhtudel on tegu *multiagentsüsteemiga*, kus süsteemi moodustavad omavahel koostööd tegevad agendid. Teatud juhtudel võib aga piisavaks osutuda ka ainult ühe agendi kasutamine. Näitena võib tuua ekspertabilistena tuntud süsteemid, kus üksainus keerukam agent aitab inimest mõne ülesande täitmisel (enamasti kergendab arvuti kasutamist mingi probleemi lahendamisel või informatsiooni kogumisel). (Jennings and Wooldridge, 1998)

4.2. Milleks agendid?

4.2.1. Milleks hajutamine?

Agentide kasutamisel arvutiteaduses on kandvaks ideeks funktsionaalsuse ja "intelligentsuse" jagamine eraldiseisvateks osadeks. Aga MIKS seda üldse tegema peaks? Selgub, et põhjusi on mitmeid ja seejuures küllaltki kaalukaid (Ferber, 1999):

- Probleemid / ülesanded on füüsiliselt hajutatud. Paljudes valdkondades on hajutatus loomulik omadus. Transpordivõrgustikud ongi olemas justnimelt vajaduse tõttu geograafiliselt hajutatud punktide vahel materjale liigutada, aga teatavas ulatuses on ruumiline hajutatus paratamatu ka mujal, näiteks vabrikute koosteliinides või keemiatehaste tootmiskompleksides.
- Probleemid on funktsionaalselt hajutatud ja heterogeensed. Keeruka toote, näiteks võidusõiduauto või reisilennuki, valmistamine vajab suure hulga spetsialistide koostööd, kellel igaühel on kindlad spetsiifilised oskused. Üksikindiviid ei suudaks omandada tohutut hulka teadmisi, mida selliste süsteemide edukaks valmistamiseks vaja oleks.
- Võrgud nõuavad hajutatuse mõistmist ja kasutamist. Äärmiselt suurte keerukate võrkude (nt. Internet) tööshoidmiseks ja kasutamiseks peab mõtlema pigem mitte globaalselt-ühtlustavalt, vaid lähtuma avatud süsteemide ideoloogiast: *erinevad* süsteemid peavad suutma teatud ulatuses koostööd teha. Multiagentsüsteemid on üks tõsiseltvõetavaid kandidaate avatud, hajutatud, heterogeensete ja paindlike arhitektuuride loomiseks, mis ei nõua struktuuri eelnevat paikapanekut.
- Keerukaid probleeme ei saa tervikuna analüüsida ja lahendada. Enamasti aga on võimalik suurt ülesannet jaotada väiksemateks osadeks, millest jõud üle käib. Näiteks juhtida terve maakera lennuliiklust on äärmiselt keerukas, kuna mitmesuguste parameetrite ja piirangute arv on tohutu ning pidevalt muutuv. Praktikas kasutatakse lokaalseid lennujuhtimiskeskusi, mis üsna edukalt toimivad.
- Süsteemid peavad suutma kohaneda nii sisemiste kui väliskeskkonna muutustega. Avatud süsteemis lisatakse, muudetakse ning eemaldatakse erinevaid komponente ning pidevalt võivad muutuda välistingimused. Kuna multiagentsüsteemide olulisteks omadusteks on lokaalsus ning koostöö, samuti agentide lisamise ja eemaldamise võimalus isegi töötavas süsteemis, siis on neil hea eeldus pakkuda lahendusi avatud süsteemide realiseerimiseks.
- Tarkvaratehnika liigub disainide suunas, mis baseeruvad autonoomsetel interaktiivsetel üksustel. Tarkvaraarenduse ajalugu näitab, et arvutiprogrammide loomisel liigutakse üha enam hajutatuse suunas ning kasutatakse üha iseseisvamaid komponente.

Seega ei ole hajutamises tegelikult midagi uut. Lihtsalt seoses arvutiteaduse suhteliselt noore eaga ei ole funktsionaalsuse ja "intelligentsuse" hajutamine veel piisavalt omaks võetud ning selles suunas alles liigutakse.

4.2.2. Agendid *vs.* objektid

Milleks kasutada agente, kui on juba olemas laialt levinud ning populaarne objektorienteeritud programmeerimine? Tõepoolest, põhimõtteliselt saaks samad probleemid lahendatud ka objektide abil. Enamus agentide valmistamise keskkondi ongi kirjutatud objektorienteeritud keeltes. Küsimus on pigem lähenemisnurgas ja abstraktsioonitasemes. Objektorienteerituse ideoloogia ei räägi autonoomsest käitumisest (reaktiivsus, proaktiivsus, sotsiaalsus), mistõttu kõik sellega seonduvad probleemid tuleb programmeerijal endal lahendada. Samuti on standardses objektidel baseeruvas süsteemis üks juhtlõim *(thread of control)*, agentsüsteemid aga nõuavad juba oma autonoomse olemuse tõttu hargtöötlust *(multithreading)*, mistõttu igale agendile antakse tavaliselt oma lõim (Chanchalani, 2003). Seega mõne olemasoleva agentide valmistamise keskkonna kasutamine lihtsalt paljudel juhtudel kiirendab ja kergendab programmeerija tööd.

Agentide kasutamine võib tarkvaraarendust lihtsustada veel mitmel moel. Kuna agendid on suhteliselt autonoomsed ja tihti üsna standartsete suhtlemisliidestega, siis suurte projektide korral, kus tegevuses on palju tarkvarainsenere, võib süsteemi kokkupanek olla oluliselt lihtsam (analoogselt näiteks objektorienteerituse eelistega "tavalise", mitteobjektorienteeritud programmeerimise ees). Lisaks sellele võimaldab agentide kasutamine küllaltki lihtsalt kaasata uude süsteemi vanu olemasolevaid tarkvarakomponente: need saab "ümbritseda" kihiga, mis tõlgib vana komponendi suhtluse agentidele mõistetavasse keelde ja vastupidi (Jennings and Wooldridge, 1998).

4.3. Agentide rakendused

4.3.1. Kasutusvaldkonnad

Eelmises jaotuses *"Milleks agendid?"* toodud põhjendusi vaadeldes saab juba ilmselt esmase ettekujutuse, kus agenttehnoloogiast abi võiks olla. Samuti ilmneb, et potentsiaalseid rakendusi on äärmiselt palju ja erinevaid. Nendest võimalikest valdkondadest kompaktse nimekirja koostamine ei ole just lihtne, kuid ühe variandi on oma raamatus välja pakkunud Ferber (1999):

- Probleemide lahendamine – laiemas mõttes kõik situatsioonid, kus kasutatakse tarkvaralisi ilma füüsilise struktuurita agente mingi ülesande täitmiseks / lahendamiseks. Võib omakorda jaotada:
  - Probleemide hajutatud lahendamine – kui tegu on suure ja keerulise probleemiga, siis on lahendamiseks vaja erinevate teadmiste ja oskustega üksuste (agentide, inimeste, ...) koostööd.
  - Hajutatud probleemide lahendamine – ülesanne ise on hajutatud, üldjuhul füüsiliselt (näiteks elektrijaotusvõrgu töö tagamine).
- Multiagentsimulatsioonid – võimalus simuleeritava süsteemi igale osale (näiteks molekul, putukas, inimene, liiklusvahend) vastavusse panna agent, mille tulemusena saadav mudel vastab paremini tegelikkusele ja on samas ka kergemini mõistetav (erinevalt näiteks diferentsiaalvõrranditest ja muudest puhtmatemaatilistest mudelitest).
- Tehismaailmade loomine – erineb eelmisest selle poolest, et ei proovita võimalikult täpselt simuleerida olemasolevat süsteemi, vaid pigem uuritakse põhjalikumalt mõnda agentidevahelise interaktsiooni mehhanismi ja selle mõju.

- Kollektiivrobootika – roboteid ehitatakse koostöö põhimõttest lähtuvalt:
  - robotisse võib koosneda moodulitest, mis omavahel koostööd teevad,
  - või siis on roboteid mitu ja nad käituvad ülesande lahendamisel koordineeritult.
- Tarkvara disain – üleminek objektorienteerituselt või muudelt lähenemisviisidelt agentorienteeritusele (rakendustes, kus see mõistlikuks osutub).

Toodud valdkonnad on kohati üsna laialivalguvad, mistõttu parema ettekujutuse saamiseks agentide võimalustest ja kasulikkusest on järgnevas esitatud kolm näidet multiagentsüsteemide rakenduste kohta tööstus- ja ärikeskkonnas (agentide kasutamine teaduses tuleb vaatluse alla hilisemates peatükkides). Esimene näide demonstreerib psühholoogilisi probleeme agenttehnoloogia kasutuselevõtul tööstuses; teisel juhul selliseid probleeme ei tekkinud ja lahendust rakendatakse edukalt tootmistöös; kolmandas näites kasutati tarkvaralisi agente kõigepealt ettevõtte tööprotsesside simuleerimiseks, et saadud tulemuste alusel neid protsesse efektiivsemaks muuta, ning agenttehnoloogia otsese ettevõtte töös rakendamise suunas alles liigutakse.

4.3.2. Näide 1: Autode värvimise töökoda

Tööstusseadmete juhtsüsteeme programmeeritakse tihti redeldiagrammide abil, mis on põhimõtteliselt tuletatud füüsiliste releede ühendamise diagrammidest (füüsilistel releedel realiseeriti juhtautomaatikat juba sadu aastaid tagasi). Kuigi tehnilised võimalused on kaasajal oluliselt teistsugused, kasutatakse juhtmeskeemide metafoori jätkuvalt väga laialdaselt, sest see on kergestimõistetav ja paljudel juhtudel väga sobiv kontseptsioon. Samas on tema võimalused mõnevõrra piiratud – mõnikord oleks otstarbekam kasutada keerukamaid juhtseadiseid, millele usaldatakse oluliselt rohkem otsustusõigust, s.t. tootmisüksuse käitumine ei ole täielikult inimese poolt ette määratud (see iseenesest muidugi ei välista tavaliste programmeeritavate kontrollerite kasutamist, pigem nõuab teatavat mõtteviisi muutust).

Hea näide multiagentsüsteemide efektiivsusest tööstuses on General Motors' i (GM) poolt Fort Wayne' i koostetehases (Indiana, USA) aastal 1992 kasutusele võetud autovärvimissüsteem. Probleem seisnes järgnevas:

> Pärast veoauto kokkupanekut tuleb ta värvida vastavalt tellija soovile. Värvimisjaamu on aga vähem kui erinevaid võimalikke värve. Seetõttu tuleb aegajalt jaamas värvi vahetada, mis kulutab palju aega ning suur hulk värvi läheb samuti kaduma.

Esialgne optimeerimissüsteem kasutas klassikalist programmi ning seda oli väga kulukas reguleerida ning ülal pidada. Lisaks pidi kogu töö väga detailselt ette planeerima ning iga rike värvimisjaamades tekitas koheselt suure probleemi.

Värvimistöökoja juhtimiseks valmistati multiagentsüsteem, kus iga värvimisjaama esindas üks agent ning tööpõhimõte oli järgmine: kui jaam on vaba, siis ta võtab ette järgmise veoauto ootejärjekorrast (mille pikkus on umbes 100 autot), järgides kolme reeglit:

1. Valida järjekorrast esimene selline auto, mis vajab parajasti jaamas olevat värvi.
2. Kui ükski auto seda värvi ei vaja, siis valida suurima prioriteediga auto ning lülituda ümber tema poolt vajatavale värvile.
3. Kui ei ole prioriteetseid autosid, võtta järjekorrast esimene auto ning lülituda ümber tema poolt vajatavale värvile.

Lisaks sellele muudeti agendipõhiseks ka üksikute seadmete (niisutajad, kuumutajad, jahutajad jne.) juhtimine põhimõttel, et seadmed ise "vastutavad" oma tegevuse eest, käitudes vastavalt kohapealsele olukorrale. Hoolimata sellest, et sisuliselt olid kõik need muudatused üsna lihtsad ega kasutanud mingeid keerukaid teooriaid, säästeti tänu uuele süsteemile aastas 2 miljonit dollarit, juhtimistarkvara hulk vähenes 40% ja üksikud rikked värvimisjaamades ei tekitanud enam tõsisemat probleemi.

Hoolimata edust lõpetas GM kolme aasta pärast süsteemi kasutamise. Green' i (1995) sõnul põhjendas veoautode tootmise juht Ernest Vahala olukorda sellega, et kuna kompanii asendab hüdraulilised värvimisrobotid elektrilistega, võetakse kasutusele uus tarkvara. Vahala lisas, et tõenäoliselt uurib GM kasutatud lähenemisviisi kunagi hiljem edasi, kuid: *"There continues to be resistance to chaos because people don't understand it. It defies good logic."* ("Kaosele on jätkuvalt vastuseis, kuna inimesed ei saa sellest aru. See ei lähe kokku hea loogikaga.". Seletus: sel ajal seostati nimetatud süsteemi eelkõige kaoseteooriaga, kuna masinatele anti üsna suur autonoomia ning need käitusid vastavalt oma "soovidele", kõrvaltvaataja jaoks veidi "kaootiliselt", mitte range tegevuskava järgi.).

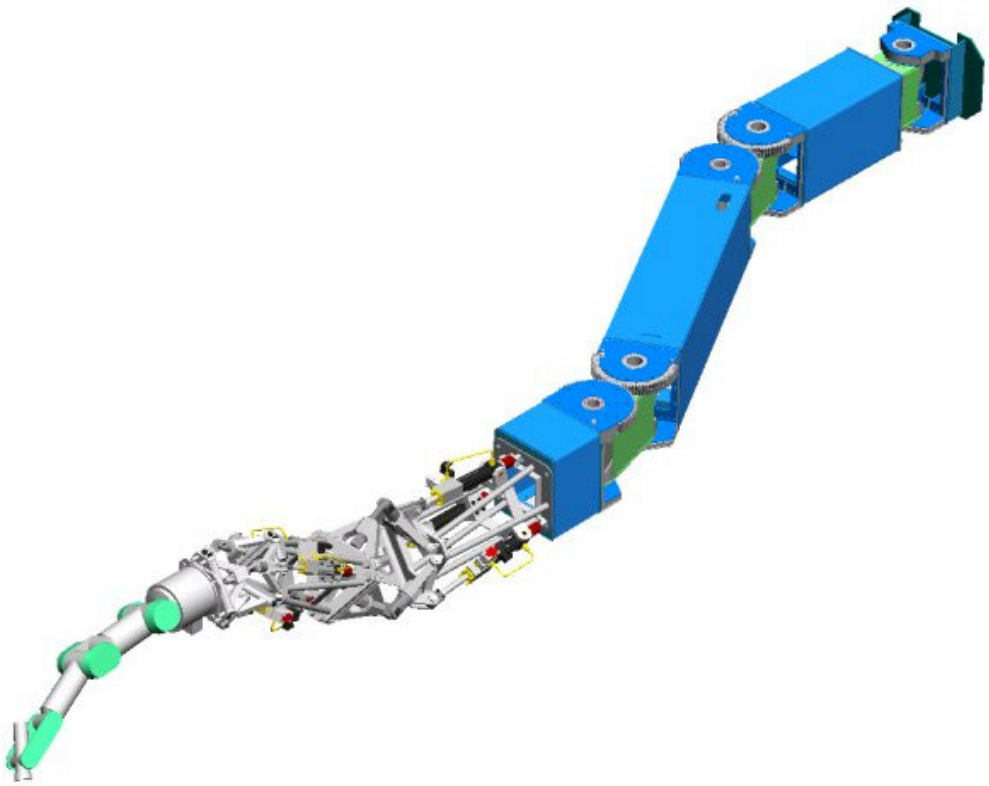
// Näide 1 on kompileeritud allikatest Parunak (1998), Ferber (1999) ja Green (1995).

4.3.3. Näide 2: AMROSE

Laevanduses kasutatakse järjest enam topeltkerega ehk kahepõhjalisi aluseid. Nende laevade kere koosneb kahest terviklikust teraspinnast, mille vahel on tühi ruum. Kere valmistamise ajal on vajalik teatavad operatsioonid (näiteks keevitamine) läbi viia ajal, mil mõlemad terasekihid on juba paigas. Tavaliste vahenditega on tööriistade täpne juhtimine selles kitsas ruumis kahe terasekihi vahel äärmiselt keeruline.

AMROSE *(Autonomous Multiple Robot Operation in Structured Environments)* on süsteem, mis suudab juhtida paljusegmendilist robotkätt (joonis 4) eelnimetatud kitsas ruumis laevakere sees. Iga manipulaatori lüli on üks agent, kes teab oma asukohta. Robotkäe ots – tööriista hoidev segment – on see, kellele antakse konkreetsed juhtnöörid tööoperatsiooni läbiviimiseks. Juhul, kui otsmine lüli ei ulatu tööriistaga nõutud kohani, teatab ta endast järgmisele segmendile oma asukoha muutmise soovidest kujul "liiguta mind kohta X". See segment omakorda käitub samade reeglite järgi, andes vajadusel juhtnööre järgnevale lülile, jne. (Parunak, 1998)

Kirjeldatud süsteemi kasutatakse näiteks Odense Laevatehases Taanis.



Joonis 4. Paljusegmendiline manipulaator. Pilt allikast (The Snake Robot).

4.3.4. Näide 3: Procter & Gamble'i varustusvõrgustik

Procter & Gamble (P&G) on kompanii, mille toodete hulka kuuluvad Tide, Pringles, Pampers ja Clairol ning millel on kuni 5 miljardit tarbijat 140 riigis. Üheksakümnendate aastate lõpul oli P&G-l tõsine probleem. Esiteks: toodete jaotus- ja müügisüsteemis oli pidevalt liikvel / ladustatud 3,8 miljardi dollari väärtuses kaupa – tohutult palju. Teiseks: marketite 2000 populaarseimast tooteartiklist (millest P&G tooted moodustavad märkimisväärse osa) olid 11% pidevalt kauplustest otsas, mis on väga ebasoovitav – mõned kliendid võivad küll ostmise edasi lükata, aga suurem osa ostab hoopis teise firma kauba või jätab ostu üldse ära. Veider oli aga see, et jaotussüsteemi saadetava kaubavoo suurendamine ei vähendanud kaupade otsasolekut poodides. Sageli tuli ette hoopis olukordi, kus riiulitelt lõppenud kaup oli küll kauplusel olemas, aga ladude ülekoormatuse tõttu ei leitud seda üles. P&G seadis eesmärgiks vähendada jaotussüsteemis oleva kauba hulka 1 miljardi dollari võrra ilma kaupade otsasolekut suurendamata.

Koostöös konsultatsiooni- ja tarkvarafirmaga BiosGroup (Santa Fe, New Mexico, USA; firma asutajate hulka kuulub ka Stuart A. Kauffman) töötati välja agentmudel, kus agentideks olid P&G jaotusahela üksikud komponendid (veoautod, autojuhid, kauplused jms.). Iga agendi käitumist juhtisid reaalse elu vaatlemisel leitud reeglid, nagu näiteks "Saada veoauto välja ainult siis, kui ta on kaupa täis." või "Tee rohkem šampooni, kui laoseis jääb väiksemaks x päeva jooksul ostetavast.".

Valminud keskkonnas sai simuleerida mitmesuguste uute logistikareeglite mõju süsteemile. Peamisteks vaatlusalusteks suurusteks olid kauba hulk süsteemis, transpordikulud ja toodete lõppemine kauplustes. Katsetati erinevaid tellimis- ning kohaleveosagedusi, mitmesuguseid jaotuskeskuste ladustamispoliitikaid erinevate toodete suhtes, nõudluse ennustamise meetodeid jpm.

Mõned järeldused läbitehtud simulatsioonidest olid juba ette aimatavad ja senini seisnud ainult andmete vähesuse taga, kuid oli ka üllatavaid tulemusi. Näiteks selgus, et tihti on kasulikum saata välja mittetäielikult laaditud veoauto. Senini oli P&G transpordikulude optimeerimise seisukohast nõudnud alati täiskoormat või pooliku laadungi korral lisatasu tellijalt. Selliste reeglite tõttu tellisid kauplused alati terve koorma. See aga tekitab mitmeid probleeme: turundustsükli lõpul peab tootja ülejäägid endale tagasi vedama, samuti tekib eelpool nimetatud fenomen, kus kauba rohkus kaupluste laoruumide väiksuse tõttu hoopis raskendab müügiriiulite pidevalt varustatuna hoidmist. Lisaks kaotab P&G olulist infot reaalse nõudluse kohta, kuna kauplused tellivad vajatust rohkem. Logistikasüsteemi muudatuste simuleerimine võimaldas veenduda, et kardinaalsed muutused tõepoolest end ära tasuvad.

Läbiviidud uuringute ja simulatsioonide põhjal otsustati:

- Lõdvendada reegleid, tihti vastuolus tavamõtlemisega, et jaotusvõrgu koguefektiivsust suurendada. Kohati oli üsna keerukas veenda kaubavedude organiseerijaid, et vahel on tõepoolest kasulik lasta pooltäis autol lahkuda.
- Muuta tootmine paindlikumaks. Mudelist saadud info põhjal muudab P&G oma tootmisprotsessi põhjalikult, vahetades suurte partiide tootmise välja valmisolekuga valmistada mistahes toodet iga päev.
- Muuta jaotusvõrk paindlikumaks. Näiteks selgus, et jaemüüjat on võimalik varustada 24 tunni jooksul, mitte 48 või 72 nagu senini.

2003. aasta alguse seisuga ennustas P&G, et eesmärk vähendada jaotussüsteemis oleva kauba hulka 1 miljardi dollari võrra ilma kaupade otsasolekut suurendamata saab täidetud juunis 2003.

P&G jaoks oli ja on see kokkuvõttes äärmiselt edukas IT projekt. Investeerides vähem kui 3 miljonit dollarit on saavutatud varasemaga võrreldes kokkuhoid 300 miljonit dollarit aastas, kusjuures paljud muutused on alles algusjärgus. Järgmine suurem hüpe loodetakse saavutada aastaks 2008: kui praegu kasutab P&G agenttehnoloogiat vaid modelleerimiseks, siis selleks ajaks on plaanis võtta tarkvaralised agendid kasutusele ka tegelikes tootmis- ja varustussüsteemides.

// Näide 3 on kompileeritud allikatest Seibel and Kellam (2003) ja Anthes (2003).

4.4. Agentorienteeritud tarkvaratööriistad

Tänu järjest suurenevale huvile agenttehnoloogia vastu on viimasel ajal loodud arvukalt mitmesuguse tasemega abivahendeid tarkvaraliste agentsüsteemide valmistamiseks. Et need tööriistad, nagu valdkond isegi, on küllaltki uued, siis on ka nende poolt pakutav tugi ja võimalused paljudel juhtudel suhteliselt piiratud. Samuti ei ole välja kujunenud selgeid turuliidreid. Tõsi küll, osaliselt on see olukord põhjendatav arendusvahendite spetsialiseerumisega: mitmed tööriistad keskenduvad mõne konkreetse valdkonna problemaatikale ega soovigi väga laiahaardeliseks muutuda. Üldjoontes aga võib tänu intensiivsele arendustööle ilmselt lähiajal oodata tööriistade poolt pakutavate võimaluste hulga ning kvaliteedi olulist kasvu.

Vähegi põhjalikum ülevaade olemasolevatest vahenditest ei tule käesoleva töö piires kõne alla, seetõttu on järgnevas pigem illustratiivsel eesmärgil toodud mittetäielik

nimekiri koos äärmiselt lühikeste kirjeldustega. Loetelu koostamisel on muuhulgas kasutatud autorite Tesfatsion, Eikenberry ja Vidal poolt koostatud viidete kogumikke, kust igaüks võib huvi korral siin väljatoodud tarkvaratööriistadele veel lisa leida, ning vahendite enda koduleheküIgi.

ABLE *(Agent Building and Learning Environment)* – IBM' i poolt valmistatud Java raamprogramm, komponenditeek ja töövahend intelligentsete agentide ehitamiseks. Teegis on muuhulgas vahendid andmebaasidega suhtlemiseks, Boole' i ja hägusa loogika, närvivõrkude, Bayes' i valemite ning otsustuspuude kasutamiseks. (http://www.alphaworks.ibm.com/tech/able)

Agent Development Kit (tasuline) – Java-põhine vahend suurte turvaliste hajussüsteemide ehitamiseks. Toetab kaughaldust, agendid kasutavad suhtlemiseks XML' i. (http://www.tryllian.com/technology/product1.html)

AgentFarms – keskkond multiagentsimulatsioonideks, sisaldab näiteks graafilist mudelite loomise vahendit, abivahendeid logimiseks, muutujate väärtuste mõõtmiseks, kogumiseks ja esitamiseks, keerukate struktuuride dünaamika visualiseerimiseks ning salvestamiseks. (http://www.agentfarms.net/)

AgentSheet (tasuline) – võimaldab luua agente, interaktiivseid mänge, virtuaalseid maailmu jms. ka programmeerimisoskuseta inimestel, eelkõige koolilastel. (http://agentsheets.com/)

Aglets – IBM' i poolt loodud Java-põhine agendiplatvorm ja teek*Aglet* on Java agent, mis suudab iseseisvalt liikuda mitmest arvutist koosnevas süsteemis. (http://aglets.sourceforge.net/)

Ara – agendiplatvorm, kus põhitähelepanu on pööratud turvalisusele ja agentide mobiilsusele; vähe on tuge agentide koostöö, intelligentsuse jms. arendamiseks. (http://wwwagss.informatik.uni-kl.de/Projekte/Ara/index_e.html)

Ascape – Java-põhine raamprogramm agentmudelite loomiseks ja analüüsimiseks, kus agendid asuvad ruumis üksteise suhtes kindlatel positsioonidel, nt. reas või võrel. "Agendimaastike" *(agent landscapes)* graafiline esitus, statistilised vahendid. Võimaldab loodud mudelite lisamist internetilehekülgedele (Java rakendusena). (http://www.brookings.edu/es/dynamics/models/ascape/main.htm)

Bee-gent *(Bonding and Encapsulation Enhancement aGENT)* – agentidena on esitatud ka agentidevahelised sõnumid, kusjuures need transpordiagendid võivad sõltuvalt antud ülesandest ise valida eesmärgi saavutamiseks sobivaima tegutsemisviisi. *Bee-gent* võimaldab "agentifitseerida" ka juba olemasolevaid rakendusi, pakkudes vastavaid agentliideseid ehk -ümbriseid (sel juhul tervet rakendust käsitletakse ühe agendina). (http://www2.toshiba.co.jp/beegent/index.htm)

Bond – Java-põhine FIPA standarditele vastav raamprogramm koos vahenditega agentide sees toimuva uurimiseks, visualiseerimiseks jms. (http://bond.cs.ucf.edu/)

Brahms – inimeste ja masinate käitumise modelleerimiseks mõeldud vahend, mis võimaldab esitada agentide tegevust geograafilist infot sisaldavas virtuaalmaailmas. Tegevuste kirjeldamine on reeglipõhine, omab mõningast sarnasust BDI *(belief-desire-intention)* arhitektuuriga. (http://www.agentisolutions.com/brahms.htm)

Cormas – (loodus)ressursside (ühis)kasutamise modelleerimiseks mõeldud keskkond, loodud keeles SmallTalk. (http://cormas.cirad.fr/indexeng.htm)

Cougaar – DARPA poolt finantseeritud Java-põhine arhitektuur suuremõõtmeliste hajutatud agentrakenduste loomiseks. (http://www.cougaar.org/)

CybelePro (tasuline) – Java-põhine infrastruktuur, kus suurt tähelepanu on pööratud ka turvalisusele. (http://www.cybelepro.com/)

DECAF *(Distributed, Environment-Centered Agent Framework)* – Java-põhine raamprogramm agentsüsteemide loomiseks. Sisaldab graafilist kasutajaliidest, mis võimaldab agentide visuaalset programmeerimist. (http://www.eecis.udel.edu/~decaf/)

DIET Agents – Java-põhine, rõhuga platvormi väikesel ressursinõudlikkusel. (http://diet-agents.sourceforge.net/)

D-OMAR *(Distributed Operator Model Architecture)* – osaliselt Java'l baseeruv, kuid agentide kirjeldamiseks kasutatakse spetsiaalset LISP'i laiendust. (http://omar.bbn.com/)

Echo – simulatsioonivahend eelkõige bioloogilistes populatsioonides mitmekesisust ja infotöötlust reguleerivate mehhanismide uurimiseks. Agendi tegevuse määrab individuaalne genotüüp. (http://www.santafe.edu/projects/echo/)

FIPA-OS – tööriist FIPA standarditele vastavate agentide valmistamiseks. (http://www.emorphia.com/research/about.htm)

Hive – MIT MediaLab'i Java-platvorm hajusrakenduste loomiseks, eelkõige mitmesuguste väikeste (kantavate või nt. "targas kodus" asuvate) arvutite ühendamiseks. (http://hive.sourceforge.net/)

Ingenias Development Kit – multiagentsüsteemide arendusprotsessi toetav tööriist, mis võimaldab süsteeme kirjeldada üldiste agendimõistete abil ning genereerida neist kirjeldustest töötavat koodi. (http://ingenias.sourceforge.net/)

Jacomma *(Java Communicating Agents)* – võimaldab ICM'il (*Interagent Communication Model)* vastavate reaktiivsete üle võrgu hajutatud infoagentide loomist nii Java kui JPython'i abil. (http://jacomma.sourceforge.net/)

JADE *(Java Agent Development Framework)* – Java-põhine raamprogramm FIPA spetsifikatsioonile vastavate mobiilsete agentide loomiseks. Sisaldab graafilisi abivahendeid nii arendus- kui haldusprotsessi lihtsustamiseks. (http://jade.tilab.com/)

MadKit – kirjutatud Java's, kasutab AGR (*Agent/Group/Role)* organisatsioonimudelit, milles agendid kuuluvad gruppidesse ning täidavad rolle. Agentide kirjeldamiseks saab kasutada keeli / vahendeid: Java, Scheme (Kawa), Jess, BeanShell. (http://www.madkit.org/)

MAML *(Multi-Agent Modeling Language)* – põhineb agendiarendusvahendil Swarm, proovides selle kasutamist oluliselt lihtsamaks muuta. (http://www.maml.hu/)

MASON – Java teek multiagentsimulatsioonide loomiseks rõhuga kiirusel ja väiksusel. Sisaldab 2D ja 3D visualiseerimisvahendeid ning piltide ja videote salvestamise võimalust. (http://cs.gmu.edu/~eclab/projects/mason/)

Mobidyc *(MOdeling Based on Individuals for the DYnamics of Communities)* – võimaldab ökoloogiliste ja bioloogiliste indiviidipõhiste mudelite loomist ka ilma põhjalikumate arvutiteadmisteta. Kirjutatud keeles Cincom VisualWorks Smalltalk. Kasutab kolme tüüpi agente: paigalseisvaid, mobiilseid ja ilma asukohata. Sisaldab mitmesuguseid visualiseerimis- ja salvestusvahendeid. (http://www.avignon.inra.fr/internet/unites/biometrie/mobidyc_projet/version_index_html)

NetLogo – nii õppuritele kui ka teadlastele mõeldud keerukate süsteemide modelleerimise keskkond. Sisaldab suurel hulgal näidismudeleid. Lisaks omab klassiruumis kasutamiseks mõeldud vahendit *HubNet*, kus iga õpilane saab oma arvuti või Texas Instruments (TI-83+) kalkulaatori kaudu juhtida-kontrollida ühte simulatsioonis osalevat agenti. Kirjutatud Java' s, võimaldab mudelite eksporti Java aplettideks. (http://ccl.northwestern.edu/netlogo/)

Network Agents (ICM, April, Go!, DialoX) – grupp süsteeme intelligentsete agentide ehitamiseks: ICM – agentide suhtlemise infrastruktuur; April – agentide ehitamise keel; Go! – loogikakeel; DialoX – XML-põhine kasutajaliidese mootor. (http://sourceforge.net/projects/networkagent/)

Open Agent Architecture – raamprogramm, mis võimaldab tarkvaraliste hajutatud agentide kogumil üheskoos ülesandeid lahendada. Defineerib ise agentidevahelise suhtlemise keele; toetab mitmesuguseid programmeerimiskeeli. (http://www.ai.sri.com/~oaa/)

OpenCybele – avatud lähtekoodiga versioon agendiinfrastruktuurist Cybele. (http://www.opencybele.org/)

Ps-i *(Political Science - Identity)* – valmistatud eesmärgiga võimaldada ilma varasema programmeerimisoskuseta sotsiaalteadlastel agentmudeleid luua. Sisaldab deklaratiivset keelt mudeli spetsifitseerimiseks. (http://ps-i.sourceforge.net/)

Quicksilver – väheste võimalustega vahend Java' s kirjutatavate agentmudelite loomisprotsessi mõningaseks kergendamiseks. (http://quicksilver.tigris.org/)

Repast *(Recursive Porous Agent Simulation Toolkit)* – sarnaneb Swarm' ile, kuid omab implementatsioone mitmetes programmeerimiskeeltes ning sisaldab mõningaid lisavahendeid kohanemise modelleerimiseks, nt. geneetilisi algoritme. Loodud eelkõige elusate sotsiaalsete agentide modelleerimiseks. Omab visualiseerimisvahendeid, toetab geograafilise info kasutamist. (http://repast.sourceforge.net/)

SDML *(Strictly Declarative Modelling Language)* – kõik teadmised on esitatud deklaratiivselt, reegli- ja andmebaasidena. Võimaldab mitmest lihtsamast agendist moodustada ühe keerukama. (http://cfpm.org/sdml/)

SeSAm *(Shell for Simulated Agent Systems)* – agentsimulatsioonide keskkond koos mitmesuguste visualiseerimisvahenditega. Agendi kirjeldamiseks kasutatakse UML' i sarnaseid diagramme. (http://www.simsesam.de/)

Sim_Agent Toolkit – algselt loodud toetamaks inimesesarnaste intelligentsete agentide uurimist, kuid tudengite poolt kasutust leidnud ka interaktiivsete mängude valmistamisel. Toetab keeruka sisemise struktuuriga (närvivõrgud, sümbolloogika jms.) agentide loomist. (http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html)

SimBioSys – C++ raamprogramm evolutsiooniliste agentsimulatsioonide tarvis. (http://www.kumo.com/~david/SimBioSys/)

StarLogo – kui tuntud õppekeeles *Logo* saab ekraanil juhtida ühte "kilpkonna", siis StarLogo' s on kilpkonni palju (kasvõi tuhandeid) ja nad liiguvad mööda muudetavate omadustega maapinda. (http://education.mit.edu/starlogo/)

StarlogoT – StarLogo laiendus, töötab ainult Macintosh arvutitel. (http://ccl.northwestern.edu/cm/)

Swarm – Santa Fe Instituudis loodud tarkvarapakett keerukate süsteemide simuleerimiseks. Koosneb Objective-C' s ja Tk' s kirjutatud teekidest, võimaldab ka Java kasutamist. (http://www.swarm.org/wiki/Main_Page)

ZEUS – *British Telecommunications*' i Java-põhine tööriistakomplekt agentsüsteemide valmistamiseks. Iga ZEUS agent koosneb definitsiooni- (agendi eesmärgid, ressursid, oskused, eelistused; "arutlemis-" ja õppimisvõime), organisatsiooni- ja koordinatsioonikihist. (http://more.btexact.com/projects/agents/zeus/index.htm)

4.5. Agentorienteeritud tarkvaraarenduse ohud

Agenttehnoloogial on palju positiivseid omadusi ning mitmetes rakendustes on ta selgelt üle varasematest tehnoloogiatest. Samas on aga kasulik endale varakult teadvustada, millised on võimalikud ohud agentide kasutamisel. Alusetud lootused võivad purunemise korral tekitada frustratsiooni ning soovimatust agenttehnoloogiat üleüldse kusagil kasutada. Realistlikult lähenedes ning võimalikke takistusi teades on kergem agente rakendada just neile sobivates valdkondades ja tänu sellele oma probleeme edukamalt ning efektiivsemalt lahendada.

Järgnev võimalike ohtude kirjeldus on oluliselt lühendatud refereering artiklist Wooldridge and Jennings (1998).

4.5.1. Poliitilised ohud

- Agentide liigne reklaamimine. Agenttehnoloogia kasutamine võib küll mitmetes rakendustes väga edukas olla, kuid ta ei ole maagiline ideoloogia kõigi olemasolevate probleemide lahendamiseks. Kui ülesanne oli väga keerukas varasemaid tarkvaratehnoloogiaid kasutades, siis on üsna tõenäoline, et ka agentide kasutamine selle lahendamiseks saab olema raske töö. Teine tõsine oht ülereklaamimisel on agentide kasutamise võrdsustamine intelligentse probleemilahendusega. Selle kahjulikkuse suurepäraseks illustratsiooniks on kunagine ülioptimistlik ekspertsüsteemide võimaluste väljapakkumine, milledest enamus aga reaalsuseks ei ole saanud ja seetõttu on paljud inimesed ekspertsüsteemides pettunud, kuigi mitmetes valdkondades on nende kasutamine igati õigustatud ja edukas.
- Agentide suhtes religioosseks või dogmaatiliseks muutumine. Agenttehnoloogia ei ole universaalne lahendus. Paljudes rakendustes on teised paradigmad, näiteks

objektorienteeritus, palju sobivamad ja efektiivsemad. Isegi kui on alust arvata, et konkreetse probleemi lahendamisel on agenttehnoloogia enam-vähem sama efektiivne kui mõni varasem lähenemine, siis üldjuhul on kasulik valida viimati-nimetatu, sest selles on tarkvarainsenerid enamasti kogenumad ja oskuslikumad. Dogmaatilisuse oht väljendub oma agendidefinitsiooni täielikus järgimises ka juhul, kui selleks puudub vajadus. Näiteks kui keegi leiab, et *liikuvus* on agendi lahutamatu omadus, siis ta tõenäoliselt üritab panna oma agente keskkonnas ringi liikuma isegi juhul, kui staatilised agendid oleks palju parem lahendus.

4.5.2. Ohud projekti haldamisel

- Sa ei tea, miks justnimelt agente tahad kasutada. Projektijuhid võivad ajakirjanduse mõjul agenttehnoloogiast liigselt innustuda, eriti kuna *agent* on kontseptuaalselt suhteliselt kergesti arusaadav mõiste. Kui aga alustada projekti vähese ettevalmistusega ning pelga lootusega, et küll uus tehnoloogia edukas on, siis see projekt eriti kaugele ei jõua.

- Sa ei tea, milleks täpselt agendid kasulikud on. See on seotud eelneva punktiga. Mitte mõistes agenttehnoloogia rakendamise võimalusi erinevates valdkondades võib jõuda näiteks olukorda, kus mõne konkreetse rakenduse jaoks välja töötatud spetsiifilist agentarhitektuuri üritatakse kasutada ka kõigis järgnevates projektides, mis võivad olla hoopis teiste nõuetega.

- Konkreetse ülesande jaoks universaalse lahenduse loomine. Oht seisneb selles, et proovitakse valmistada agentarhitektuur, mida saaks kasutada ka kõikvõimalike muude ülesannete lahendamisel. Üldjuhul on aga tulemuseks ajakulu ning projekti kallinemine, samuti lahenduse liigne keerukus ja ebaefektiivsus. Tarkvaratehnika praktika näitab, et komponentide taaskasutus on enamasti mõistlik ainult sarnaste probleemide lahendamisel ning suuremad erinevused lähteülesandes nõuaks juba liigseid ja ebapraktilisi muutusi komponentides.

- Prototüüpide ja reaalsete süsteemide segiajamine. Agenttehnoloogias on suhteliselt lihtne valmistada vähese agentide arvuga prototüüp, mis midagi kasulikku teeb. See on aga veel väga kaugel töökindlast ja usaldusväärsest süsteemist, mida praktikas kasutada võiks.

4.5.3. Kontseptuaalsed ohud

- Uskumine, et agenttehnoloogia on "hõbekuul". Selliselt nimetatakse tarkvara-arenduses tehnikat, mille kasutuselevõtt tõstaks tarkvara loomise efektiivsust suurusjärgu võrra. Kuigi agenttehnoloogia mõningatel juhtudel tõepoolest võib olukorda paremaks teha, ei tasu sellegipoolest imesid oodata. Ilmselt on agendid järjekordne abstraktsiooniaste, mis võimaldab inseneril keerukusega paremini toime tulla, nagu seda olid / on ka protseduurid, struktuurprogrammeerimine, abstraktsed andmetüübid ja objektid, aga iga uue asja positiivsete ja negatiivsete külgede ilmnemine ning inseneride oskuste väljakujunemine võtab oma aja.

- Üleshaibitud sõna ja reaalse kontseptsiooni segiajamine. Üks agenttehnoloogia populaarsuse põhjusi on termini *agent* intuitiivsus-arusaadavus. Ühest küljest on see hea, näidates tema kasulikkust ja laia rakendatavust, teisalt aga kalduvad tarkvaraarendajad uskuma, et neil on asjast sisuline ja põhjalik arusaam, kuigi tegelikult ei ole.

- Unustamine, et tegeldakse t a r k v a r a loomisega. Kuna agenttehnoloogia on alles väljakujunemise staadiumis, seisneb agentsüsteemi loomine enamasti pidevas eksperimenteerimises. Konkreetse projekti korral (eriti kui tegeldakse kliendi soovide täitmisega, mitte lihtsalt uurimistööga) võivad aga tekkida tõsised komplikatsioonid, kui kogu aeg ja raha kulutataksegi ainult agentarhitektuuri uurimisele ja väljaarendamisele, unustades kõik muud olulised arenduse aspektid alates kasutajanõuete analüüsist kuni põhjaliku testimiseni.

- Unustamine, et luuakse h a j u s tarkvara. Hajussüsteemid on üks kõige keerulisem liik arvutisüsteeme ning aja jooksul on tehtud palju uurimistööd, et selle keerukusega hakkama saada. Multiagentsüsteem on *vähemalt* sama keeruline kui tavaline hajussüsteem, seetõttu ei maksa juba olemasolevat kogemustepagasit kõrvale heita. Ka agentsüsteemi mõistliku töö tagamiseks tuleb arvestada teadatuntud probleemidega nagu sünkroniseerimine, ühiste ressursside kasutamine, ummikud *(deadlock* ja *livelock)*.

  // Ääremärkus: *Livelock* on olukord, kus kaks või enam protsessi pidevalt muudavad oma olekut vastusena teis(t)e protsessi(de) muutus(t)ele, seejuures ilma mingit kasulikku tööd tegemata. Erineb *deadlock*' ist selle poolest, et ükski protsess ei ole blokeeritud ega oota midagi. Analoogiline olukord reaalelust on juhtum, kui kaks vastassuundades kõndivat inimest üritavad teineteisest mööduda ja teevad selleks sammu kõrvale, kuid mõlemad alati samale poole, mille tulemusena nad vasakuleparemale "kõikuma" hakkavad. // Allikast (Livelock from FOLDOC).

4.5.4. Analüüsi- ja disainiohud

- Ei kasutata ära olemasolevaid tehnoloogiaid. Agentsüsteemi loomisel moodustab agendispetsiifiline osa (nt. komponentide koostöö või läbirääkimiste teostamine) suhteliselt väikese osa kogu tööst. Ülejäänu valmistamisel on tulusam kasutada juba olemasolevaid tehnoloogiaid (näiteks hajusarvutusplatvorme ja andmebaasisüsteeme) ja meetodeid vältimaks mõttetut ressursikulu "jalgratta leiutamisele".

- Ei kasutata ära paralleelsust. Tüüpiline disainiviga avaldub selles, et kõigepealt üks agent teeb natuke tööd, annab andmed järgmisele ning jääb ootere•iimi. Järgmine agent käitub sarnaselt, jne. Sellisel juhul on kasutamata jäänud agentsüsteemide üks kõige olulisemaid omadusi: paralleelsus. Normaalses agentsüsteemis töötavad agendid samaaegselt (või ühe protsessori korral vähemalt simuleeritakse samaaegsust).

4.5.5. Mikrotasandi (agenditasandi) ohud

- Soov välja töötada omaenda agentarhitektuur. Kuigi esialgu võib tunduda, et ükski olemasolev arhitektuur konkreetse ülesande lahendamiseks ei sobi, tasub see ikkagi põhjalikumalt järele uurida. Piisavalt usaldusväärse ja kõigi vajalike omadustega arhitektuuri väljatöötamine võib võtta aastaid ning kui enamvähem sobiv variant on juba kättesaadav, siis ei ole enamasti mõtet ise proovida uut luua. Tegelikult pole paljude rakenduste korral üldse formaalset agentarhitektuuri tarviski – piisab ainult agentide valmiskirjutamisest konkreetsele rakendusele sobivas keeles.

- Arvamine, et loodud arhitektuur on geneeriline / üldsobiv. Kui mingil põhjusel on ikkagi loodud uus agentarhitektuur, siis ei tasu loota, et seda saab edaspidi kõikjal

kasutada. Ka arhitektuurid on enamasti probleemispetsiifilised – kasutatavad ainult sarnaste ülesannete lahendamiseks.

- Agendid kasutavad liiga palju tehisintelligentsi. Tihti tekib soov kasutada kõikvõimalikke eksperimentaalseid tehnikaid tehisintelligentsi vallast, mille tagajärjeks on ülekoormatud kasutuskõlbmatu süsteem. Üldjuhul on tulusam alustada minimaalse vajaliku loomisest ning alles hiljem, kui esialgne süsteem juba töötab, vajadusel lisandusi teha.

- Agentidel ei ole üldse intelligentsi. Täiesti tavaliste hajussüsteemide komponentide või skripte kasutavate võrgulehekülgede nimetamine agentideks viib agendi mõiste devalveerumiseni. Selline käitumine tekitab pettumust klientides, kes lootsid saada midagi uudset ja erilist, ning küünilisust tarkvaraarendajates, kellel tekib arvamus, et *agent* on järjekordne mõttetu turundustermin.

4.5.6. Makrotasandi ohud

- Agentide nägemine kõikjal. Objektorienteeritud keelt nimetatakse kontseptuaalselt puhtaks, kui selles keeles on ainult objektid. Sarnaselt võib läheneda ka agentsüsteemidele, kuid enamasti ei ole see kasulik – kui absoluutselt kõik kuni liitmis- ja lahutamisoperatsioonini välja realiseerida agentidena, tekib tohutu efektiivsusekadu.

- Liiga palju agente. Mida rohkem agente, seda raskeminiennustatavalt ja kaootilisemalt süsteem käitub. Sõltub küll konkreetsest probleemist, kuid paljudel juhtudel soovitakse siiski saada süsteemi, mille käitumine püsib etteantud raames.

- Liiga vähe agente. Sel juhul ei kasutata ära multiagentsüsteemide võimalusi (analoogne objektorienteeritud keeles kogu funktsionaalsuse kokkukirjutamisega ainult ühte-kahte klassi).

- Kogu aja kulutamine infrastruktuuri rajamisele. Seoses multiagentsüsteemide loomiseks mõeldud ning üldkasutatavuse saavutanud arenduskeskkondade vähesusega / piiratusega kulub suur osa ressurssidest infrastruktuuri loomisele (sõnumite haldamine ja monitooring, töötava süsteemi haldamise vahendid jms.), mis pealegi ei saa tavaliselt olema eriti kõrgekvaliteediline (sest agendiprojektides on tegevad pigem tehisintelligentsi kui võrgunduse ja hajussüsteemide taustaga inimesed). Kui lõpuks agentide enda arendamiseni jõutakse, on väheks jäänud nii aega, raha kui entusiasmi.

- Süsteem on anarhiline. On levinud väärarvamus, et agentsüsteemi valmistamiseks tuleb lihtsalt suur hulk agente ühtekokku kuhjata, et pole vaja tõsisemalt struktuuri peale mõelda ning kõik agendid on võrdõiguslikud. Paljude rakenduste puhul see tõele ei vasta. Näiteks võib oluliselt efektiivsemaks osutuda agentide hierarhia kasutamine, agentidest meeskondade moodustamine ja / või väikese arvu vahendusagentide loomine, kelle kaudu ülejäänud agendid suhtlevad.

- Simuleeritud ja reaalse paralleelsuse segiajamine. Enamikul juhtudel luuakse multiagentsüsteem esialgu prototüübina, kus kõik agendid asuvad ühesainsas arvutis ning paralleelsust ja hajutatust simuleeritakse. On ohtlik eeldada, et süsteem ka reaalse hajutatuse korral täpselt samamoodi käituma hakkab. Üks oluline probleem seisneb selles, et simuleeritud hajutatuse korral on ikkagi võimalik tsentraalne juhtimine, mida paljud prototüübid kipuvad ära kasutama,

kuid mis reaalse hajutatuse korral võimatuks osutub (või võimalikkuse korral hajutatuse kasulikud omadused annulleerib).

4.5.7. Ohud süsteemi teostamisel *(implementation)*

- *Tabula rasa*. Uut tehnoloogiat kasutusele võttes arvatakse, et kõike tuleb täiesti puhtalt lehelt alustada. Tihti aga on süsteemi toimimiseks vajalikud komponendid juba olemas, kuigi tehniliselt vananenud, ning neid on raske uuesti valmistada. Agenttehnoloogia puhul on lahenduseks vanadele komponentidele *agendikihi* lisamine, mis võimaldab neid agentsüsteemiga liita ilma funktsionaalset osa uuesti kirjutamata.

- *De facto* standardite ignoreerimine. Kuna uutes tehnoloogiavaldkondades ei ole alguses tugevaid rahvusvahelisi standardeid, siis arendajad paljudel juhtudel loovad ise kõik oma süsteemile vajaliku. See aga tähendab erinevate organisatsioonide poolt valmistatud komponentide kokkusobimatust. Kuid hoolimata kinnitatud standardite vähesusest on siiski olemas mitmed praktikas kasutatavad *de facto* standardid, mida enamasti on kasulik järgida.

5. Modelleerimine bioloogias

Bioloogias kui väga mitmekesiseid ja keerukaid looduslikke süsteeme uurivas valdkonnas kasutatakse laialdaselt erinevaid mudeleid. Järgnevas on vaatluse all eelkõige matemaatilised ja arvutuslikud mudelid.

Üks võimalus bioloogias kasutatavate mudelite liigitamiseks on toodud artiklis Peck (2004):

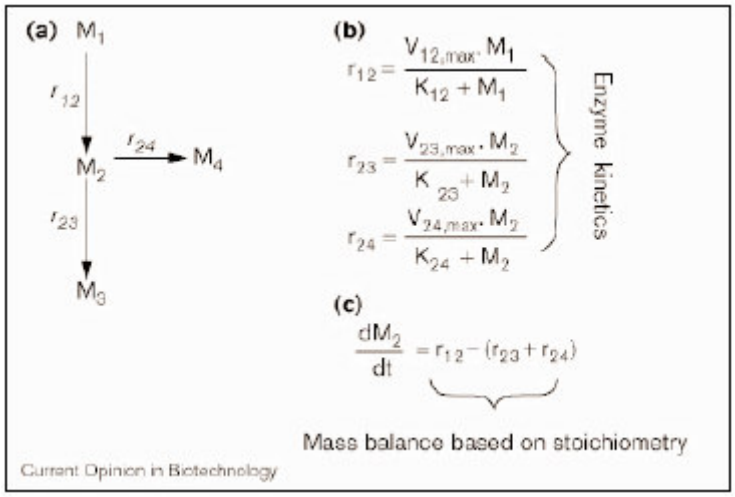
- Statistilised ennustavad mudelid – ennustusvõime saavutatakse statistilise valemi parameetrite selliste väärtuste leidmisega, mida kasutades valemi poolt antavad tulemused on võimalikult lähedased reaalsest süsteemist saadud andmetega. Statistilises mudelis puuduvad põhjuslikud mehhanismid, mis seletaks tulemuste taga seisvaid võimalikke bioloogilisi protsesse.
- Fenomenoloogilised mudelid – mudelisse on sisse kirjutatud põhjuslikud mehhanismid, mida arvatakse eksisteerivat reaalses süsteemis. Fenomenoloogiline mudel võib olla deterministlik, näiteks tavaline diferentsiaalvõrrand, mida saab täpselt ja põhjalikult analüüsida matemaatiliste vahendite abil. Kuid ta võib olla ka keerukas süsteemi simulatsioon arvutis, millest kasuliku info kättesaamiseks tuleb võtta pigem eksperimentaatori kui matemaatilise analüüsija hoiak.

Kui konkreetsemalt vaadelda metabolismi modelleerimist (metabolism = kõik eluga seotud keemilised protsessid), mis on oluline käesoleva magistritöö seisukohast, siis kaks peamist mudelite gruppi on (Gombert and Nielsen, 2000):

- Stöhhiomeetrilised mudelid – baseeruvad metaboolsete reaktsioonide võrgustiku ajast sõltumatutel omadustel. Peamiseks vahendiks on metaboolsete voogude analüüs, kus massi jäävust arvestades leitakse vood läbi metaboolsete reaktsioonide võrgustiku harude. Stöhhiomeetriliste mudelite ennustusvõime on mõnevõrra piiratud, kuna arvesse ei ole võetud mitmesuguseid metabolismi reguleerivaid mehhanisme.
- Kineetilised mudelid – baseeruvad ühest küljest stöhhiomeetrial ja teisest küljest protsesside (nt. ensüümi poolt katalüseeritud reaktsioonide, valkude interaktsioonide) kineetikal (joonis 5). Vajavad detailset informatsiooni protsesside kineetika kohta, mistõttu paljudel juhtudel ei ole info vähesuse tõttu rakendatavad.

Lisaks on viimasel ajal välja pakutud ka mitmesuguseid teisi peamiselt arvutisimulatsioonidel põhinevaid lähenemisi, millest paljud kaasavad ka stöhhiomeetria ja reaktsioonide kineetika vahendeid, osad aga kasutavad mõnevõrra teistsuguseid mõtteviise. Näiteid erinevate raku metabolismi ja üldse kogu rakus toimuva arvuti abil modelleerimise võimaluste kohta on toodud järgnevas peatükis.

Mudelite valmistamisel on äärmiselt oluline roll uuritava loodusliku süsteemi kohta teada olevatel andmetel. Hetkel on aga bioloogia käsutuses olevate andmetega suhteliselt omapärane olukord. Ühest küljest toimub, osaliselt tänu automatiseeritud katseseadmetele, väga võimas uute andmete juurdevool, näiteks genoomikas ja proteoomikas. Et kogunevast tohutust infohulgast uusi teadmisi kätte saada, kasutatakse muuhulgas andmekaevandustehnikaid, mis automaatselt otsivad andmebaasides sisalduvate objektide vahel seoseid ja millede abil saadud tulemused näitavad teadlastele kätte uusi suundi, kuhu uurimistööga edasi liikuda (Garner and Pertsemlidis, 2003). Teisest küljest aga on



Joonis 5. Näide kineetilisest modelleerimisest: stöhhiomeetriline reaktsioonivõrgustik **(a)** kombineeritakse ensüümide kineetikat kirjeldavate võrranditega **(b)** ning saadakse tulemusena metaboliidi M_2 massi kirjeldav võrrand **(c)** (Gombert and Nielsen, 2000).

just andmete vähesus see, mis pidurdab bioloogiliste süsteemide dünaamiliste mudelite valmistamist, uue olulise ala – süsteemibioloogia – lähenemisviisi organismide uurimisele (Wolkenhauer, 2002), (Wolkenhauer and Klingmüller, 2004). Põhjuseks on eelnimetatud suurte andmehulkade oluline puudus: tegu on enamjaolt bioloogilises süsteemis mingil hetkel leidunud ainete nimistu ja hulga jäädvustustega, mis ei sisalda ei ajalist ega ruumilist informatsiooni. Nimelt on protsesside dünaamika kohta käivate andmete saamine bioloogias tihti äärmiselt keeruline, nagu näiteks kirjutab Sorger (2005): "... mõõta tosina kinaasi tegevusi imetaja signaaliahelas kasvõi paaril ajahetkel on doktorikraadi omavale bioloogile suur töö".

Kuna nii andmete kui arvutisimulatsioonide hulga järsk kasv on toimunud üsna kiiresti, siis on tõsiseks probleemiks ka omavahelise ühilduvuse puudumine, mistõttu on raskendatud väga erinevaid formaate kasutavatest andmebaasidest vajaliku info leidmine ning erinevate arvutipõhiste mudelite omavaheline integratsioon (Kell, 2004), (Morris *et al.* 2005). Üks võimalus probleemi lahendada on luua programmid, mis suudavad erinevaid formaate tõlkida, omavahel teisendada. Teine ja pikemas perspektiivis ilmselt mõistlikum viis on andmestandardite väljatöötamine.

Kuigi nimetatud probleemid – sobivate andmete ning andmestandardite vähesus – on hetkel mõnevõrra limiteerivad tegurid, toimub bioloogiliste süsteemide mudelite loomine üha kasvavas tempos ja küllap ületatakse ka need takistused. Näiteks biokeemiliste reaktsioonide mudelite esitamiseks arvutile arusaadaval moel ongi juba valmimas standardid SBML (*The Systems Biology Markup Language*, http://sbml.org/) ja CellML (http://www.cellml.org/). Kuid eksisteerib ka üks oht, mis valdkonna arengule võib negatiivselt mõjuma hakata ja mida hetkel veel eriti märgata ei ole: intellektuaalomandi õigustega seonduvad probleemid. Mitmed arendajad, näiteks *Genomatica*, *CyberCell* ja *Gene Network Sciences*, on juba esitanud patenditaotlusi nii mudelitele kui ka modelleerimistehnikatele (Burbeck and Jordan, 2004).

6. Relevantsete tööde lühiülevaade

Kõikvõimalike rakus toimuvate protsesside analüüsimiseks ja simuleerimiseks on loodud arvukalt tarkvaralisi tööriistu ning veel kordades rohkem välja pakutud modelleerimisvahendite prototüüpe ja praktikas realiseerimata ideid. Mõistagi on raku modelleerimisega tegeledes kasulik tutvuda võimalikult paljude erinevate lähenemisviisidega, kuid käesoleva töö raames on mõistlik piirduda lühiülevaatega kahest kõige otsesemalt tööga seotud modelleerimisvahendite ja -ideede grupist:

1. Tarkvaralised vahendid, mis võimaldavad simuleerida võimalikult suurt osa raku metabolismist (s.t. väga spetsiifilisi, ainult mingit väga konkreetset protsessi uurivaid lahendusi ei ole järgnevas välja toodud).
2. Ideed rakusiseste protsesside modelleerimise kohta multiagentsüsteemina (raku kujutamine ainult ühe agendina ei ole siinkohal piisavalt relevantne, sest sel juhul hakkab ilmnev käitumine toimima alles populatsiooni tasandil).

6.1. Tarkvaralised vahendid raku metabolismi simuleerimiseks

Nagu eelmises peatükis mainitud, modelleeritakse metabolismi enamasti stöhhiomeetriliste (ainevood) ja kineetiliste mudelite (peamiselt diferentsiaalvõrrandid) abil. Sellest tulenevalt kasutab ka enamus tarkvaralisi lahendusi just sellist lähenemisviisi. Voopõhised on näiteks:

- MetaboLogica (http://www.metabologica.com/),
- MetaFluxNet (http://mbel.kaist.ac.kr/mfn/),
- INSILICO discovery (tasuline, http://www.insilico-biotechnology.com/f_products.html),
- ja MATLAB' ile mõeldud pakett FluxAnalyzer (http://www.mpi-magdeburg.mpg.de/en/research/projects/1010/1014/1020/mfaeng/fluxanaly.html).

Diferentsiaalvõrranditel põhinevad muuhulgas:

- KINSOLVER (http://lsdis.cs.uga.edu/~aleman/kinsolver/),
- BioNetPack (http://cellsignaling.lanl.gov/bionetgen/index.shtml),
- BioUML (http://www.biouml.org/),
- PySCeS (http://pysces.sourceforge.net/),
- SBML ODE Solver (http://www.tbi.univie.ac.at/~raim/odeSolver/),
- A-Cell (http://www.f.waseda.jp/ichikawa/A-Cell.htm),
- Simpathica (http://bioinformatics.nyu.edu/Projects/Simpathica/)

ning vististi ka:

- MMT2 (http://www.simtec.mb.uni-siegen.de/software_mmt2.0.html),
- BioPathway Explorer (tasuline, http://biosoftwareinc.com/OurProduct.htm),
- ja TERANODE Design Suite (tasuline, http://teranode.com/products/tds/index.php).

Diferentsiaalvõrrandeid kasutab ka hübriidsüsteem BioCharon (Bio Sketch Pad + Charon, http://www.cis.upenn.edu/mobies/charon/index.html), millest tuleb põhjalikumalt juttu järgmises alapunktis, sest lisaks on seal tegu agendikontseptsiooni rakendamisega.

Vaadeldes keemilisi reaktsioone mikrotasandil, tuleb arvesse võtta molekulide juhuslikku liikumist: reaktsioon saab toimuda vaid siis, kui molekulid omavahel kohtuvad. Seetõttu kasutavad paljud tööriistad metabolismi protsesside modelleerimisel stohhastilisi meetodeid alates klassikalisest Gillespie algoritmist kuni uute efektiivsemate lahendusteni. Stohhastilisel simuleerimisel põhinevad näiteks:

- Stocks (http://www.sysbio.pl/stocks/index.html),
- StochSim (http://www.anat.cam.ac.uk/groups/comp-cell/StochSim.html),
- Stochastirator (http://opnsrcbio.molsci.org/stochastirator/stoch-main.html),
- BioNetS (http://x.amath.unc.edu/bionets/),
- MesoRD (http://mesord.sourceforge.net/),
- Moleculizer (http://www.molsci.org/~lok/moleculizer/),
- RouletteCyte (http://biodynamics.indiana.edu/CellModeling/),
- ja M-Cell (http://www.mcell.cnl.salk.edu/).

Pakkumaks uurijale võimalust valida oma probleemi lahendamiseks sobivaim meetod, on tihti ühte tarkvaravahendisse sisse ehitatud kõik eelnimetatud lähenemisviisid ja modelleerija kasutab vastavalt oma soovile kas diferentsiaalvõrrandeid või stohhastilisi algoritme. Näiteid sellistest programmidest:

- Cellware (http://www.bii.a-star.edu.sg/research/sbg/cellware/index.asp),
- Dizzy (http://labs.systemsbiology.net/bolouri/software/Dizzy/),
- Dynetica (http://www.its.caltech.edu/,you/Dynetica/Dynetica-page.html),
- Jarnac ehk Scamp II (http://www.sys-bio.org/software/softwaremain.htm),
- SigTran (http://csi.washington.edu/teams/modeling/projects/sigtran/index.html),
- Gepasi (http://www.gepasi.org/),
- ja selle järglane Copasi (http://www.copasi.org/).

Mõnevõrra teistsugune vahend on BIOCHAM (*The Biochemical Abstract Machine*, http://contraintes.inria.fr/BIOCHAM/), kus biokeemiliste süsteemide modelleerimine toimub temporaalloogikat CTL *(Computation Tree Logic)* kasutades.

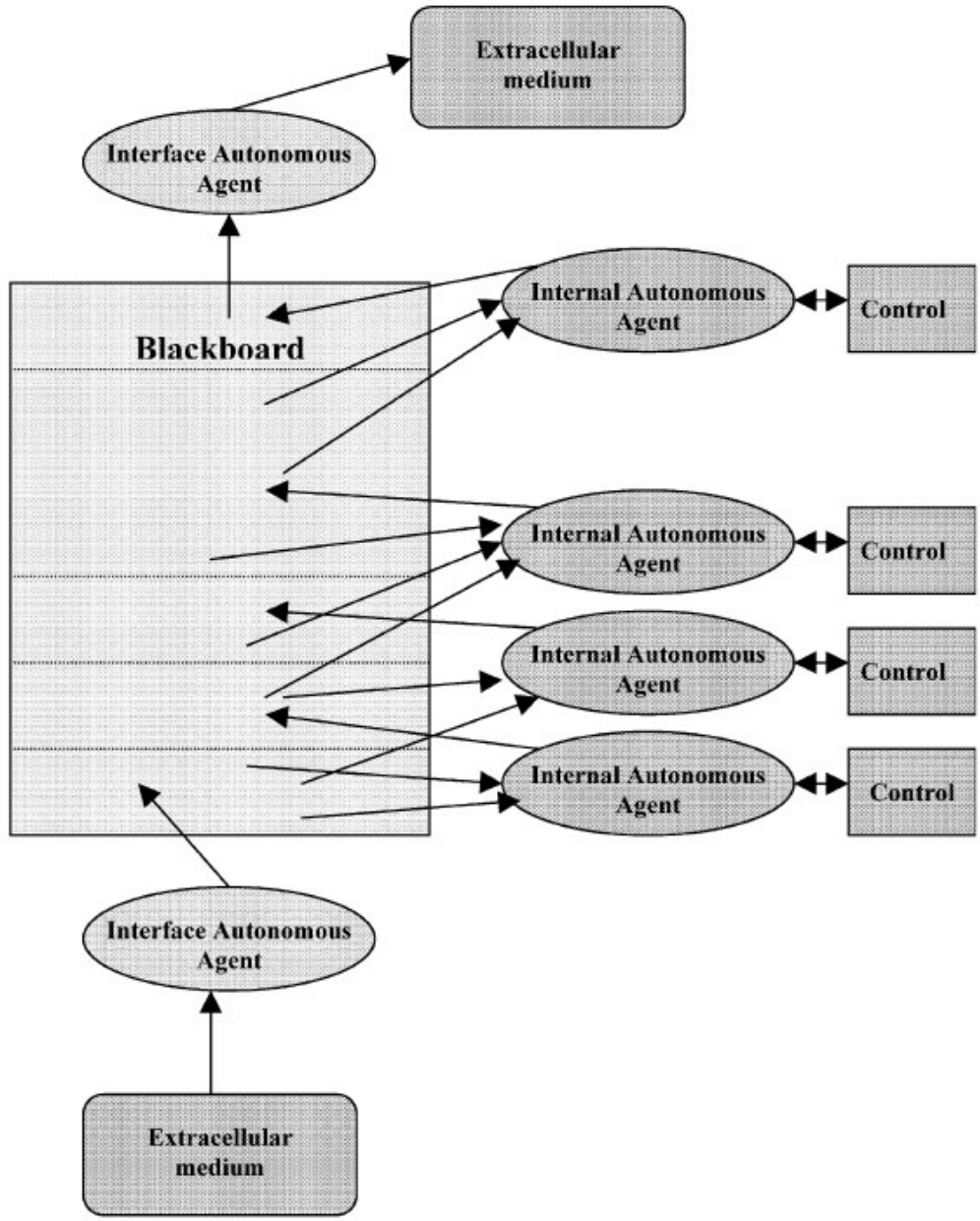
Kui eelnimetatute puhul oli tähelepanu keskpunktis eelkõige metabolismi reaktsioonidevõrgustik iseenesest, siis mõned tööriistade loojad on eesmärgiks võtnud ka raku kui terviku simuleerimise. Näiteks V-Cell (http://www.nrcam.uchc.edu/vcell_development/vcell_dev.html) võimaldab mikroskoobiga saadud piltide alusel luua raku struktuuri mudeli ning seejärel sellel bioloogilisi protsesse modelleerida. Programm V-Cell asub Ameerika Rahvuslikule Terviseinstituudile *(National Institutes of Health, NIH)* kuuluvas arvutiklastris ning on kasutatav ainult üle Interneti, nagu ka Indiana Ülikooli loodud programm Karyote (http://biodynamics.indiana.edu/CellModeling/). Kasutaja arvutisse laaditavad on näiteks SmartCell (http://www.embl.de/ExternalInfo/serrano/smartcell/), Cell-X (http://biodynamics.indiana.edu/CellModeling/), E-Cell (http://www.e-cell.org) ja SimPheny (tasuline, http://www.genomatica.com/solutions_simpheny.shtml). Mõned vahendid piirduvad ainult lihtsamate keemiliste reaktsioonide arvutamisega kogu rakus, kuid mitmed – Karyote, SmartCell, SimPheny – simuleerivad siiski ka raku kui terviku toimimises väga olulist rolli mängivaid DNA' ga seotud protsesse, kus DNA' l asuva info põhjal tekivad valgud.

6.2. Rakusiseste protsesside modelleerimine multiagentsüsteemina

Idee bioloogilise raku modelleerimisest multiagentsüsteemina ei ole veel eriti levinud ning alles hakkab seoses agentorienteerituse populaarsuse tõusuga laiemat huvi äratama. Seetõttu on teemakohaseid artikleid avaldatud väga vähe. Järgnevas on antud ülevaade käesoleva magistritöö autorile teadaolevatest artiklitest.

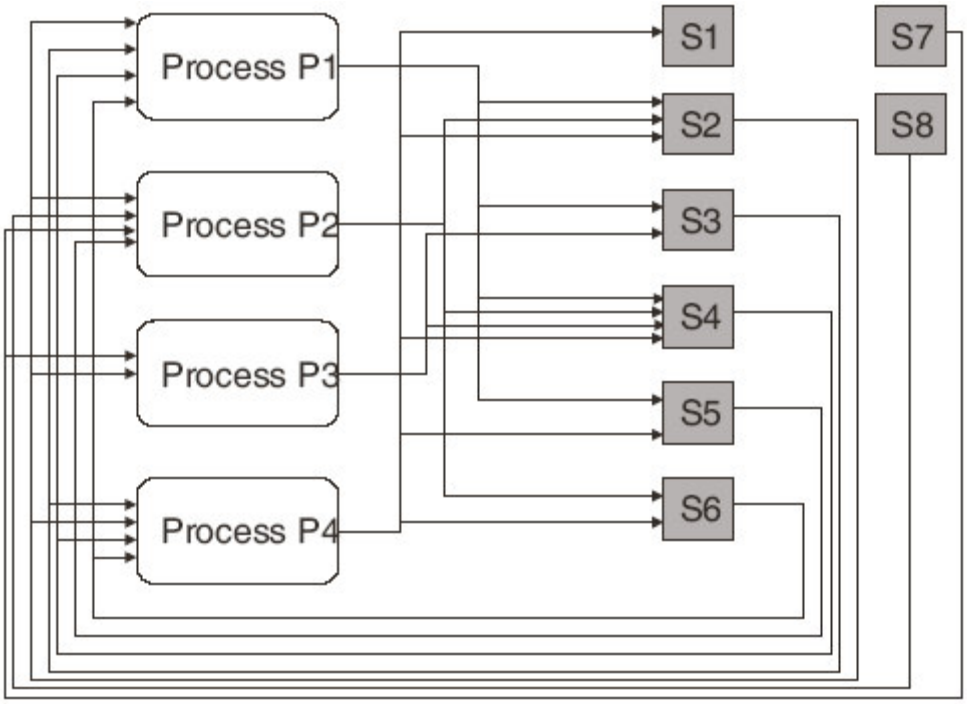
Mõnevõrra üllatuslikuna leiab juba ühest üle kümne aasta vanusest artiklist (Paton, 1993), mis pakub välja mitmesuguseid huvitavaid võimalusi raku kirjeldamiseks, idee: "... Rakke saab kirjeldada sellisel viisil, avatud süsteemidena (Huberman and Hogg, 1988), mis koosnevad autonoomsete arvutuslike üksteisega interakteeruvate agentide kogumitest.". Lisaks on seal viidatud veelgi varasematele katsetele aastast 1987 kirjeldada rakku ühiskonnana stiilis "molekulaarne demokraatia" ja "supramolekulaarne sotsialism". Sama autori (Paton) osalusel valminud hilisemates artiklites Fisher *et al.* (1999) ja Fisher *et al*. (2000) on multiagentsüsteemi kasutamise ideed edasi arendatud ja konkreetsemaks muudetud, kuid mudeli realiseerimiseni arvutis ei ole siiski veel jõutud.

// Peatükk jätkub järgmisel lehel...



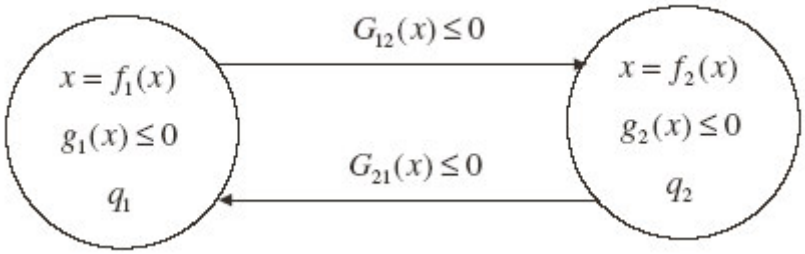
Joonis 6. Cellulat' arhitektuur (Gonzalez *et al*. 2003).

Süsteem Cellulat (Gonzalez *et al*. 2003) esitab rakusisestes signaaliahelates osalevaid valke ja teisi komponente "sisemiste" agentidena ning suhtlemine väliskeskkonnaga toimub läbi liidesagentide. Kogu agentidevaheline suhtlemine toimub kaudselt, "tahvli" *(blackboard)* ehk ühiskasutatava andmestruktuuri vahendusel (joonis 6). Tahvel koosneb tasemetest, mis sümboliseerivad raku erinevaid piirkondi, ja tahvlile salvestatavad objektid kujutavad mitmesuguseid rakusiseseid signaalmolekule ja muud liiki signaale. Joonisel 6 kujutatud juhtplokid *(Control)* teostavad erinevate agentide toimingute järjestamist, sisaldades võimalust agentide omavaheliseks konkurentsiks õiguse üle järgmisena tahvlit kasutada, s.t. mõnda rakus toimuvat protsessi teostada.



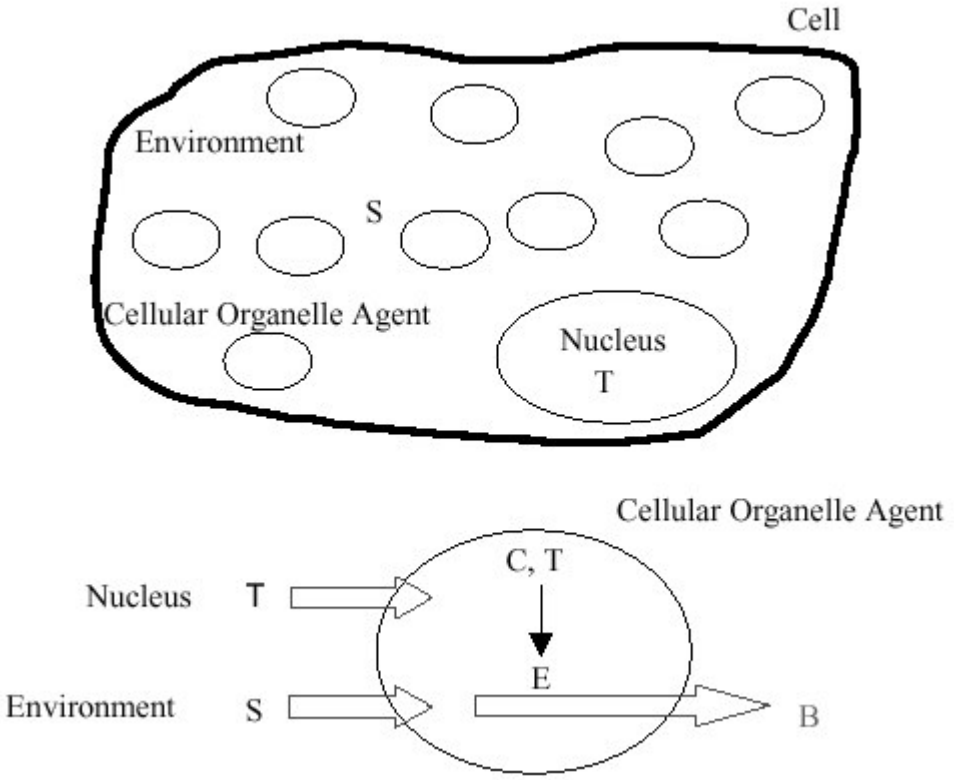
Joonis 7. P- ja S-agentidena esitatud biomolekulaarne võrgustik (Alur *et al*. 2002).

Artikkel Alur *et al*. (2002) tutvustab põhjalikumalt eelpool nimetatud tarkvaralise vahendi BioCharon aluseks olevat lähenemisviisi, milleks on nii diferentsiaalvõrrandeid kui diskreetseid sündmusi sisaldava hübriidsüsteemi kasutamine. Agente on süsteemis kahte tüüpi: P-agendid ehk protsessiagendid ja S-agendid ehk süsteemiagendid. P-agendid kirjeldavad transkriptsiooni, translatsiooni, valk-valk interaktsioonide, raku kasvu ja valkude sidumise dünaamikat. P-agendi sisenditeks on vastava protsessi jaoks oluliste ainete vms. kontsentratsioon või arv, mille ta saab S-agentide väljunditelt ja / või võrgu-välistelt allikatelt (joonis 7). Väljundiks on protsessi toimumise kiirus. S-agentidena on esitatud erinevate molekulide ja muude objektide liigid. S-agendi sisenditeks on tema suhtes relevantsete protsesside kiirused. Nende alusel arvutab ta vastavasse liiki kuuluvate molekulide uue kontsentratsiooni või arvu ning paneb tulemuse oma väljundile.



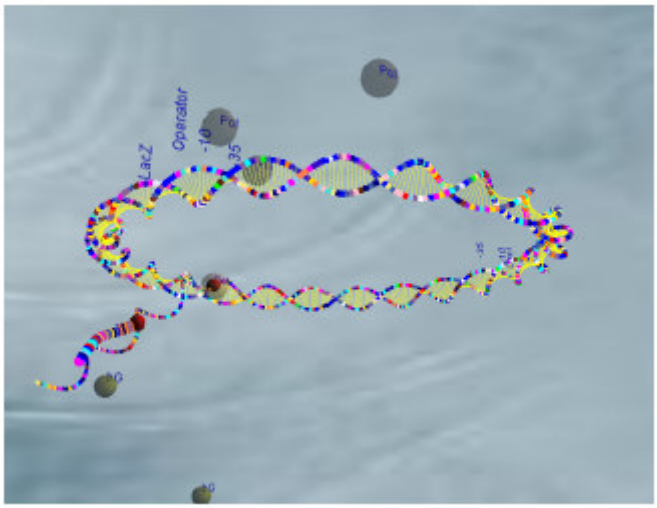
Joonis 8. Lihtne hübriidsüsteem kahe tööre•iimiga (Alur *et al*. 2002).

Iga agent sisaldab pidevat olekumuutujat x ja diskreetsete tööre•iimide kogumit Q (joonis 8). Iga reþiim sisaldab diferentsiaalvõrrandeid, millele vastavalt olekut x muudetakse, ja diferentsiaalvõrrandite kehtivuspiirkonda kirjeldavat infot. Vajadusel võib mõnes reþiimis diferentsiaalvõrrandite asemel kasutada kaa muid meetodeid, näiteks aine väga madalate kontsentratsioonide korral võivad parema tulemuse anda stohhastilised algoritmid. Tööre•iimivahetus toimub vastavalt eelnevalt kindlaksmääratud tingimustele sõltuvalt olekust x.



Joonis 9. Raku esitus multiagentsüsteemina
artiklis Katare and Venkatasubramanian (2001).

Artiklis Katare and Venkatasubramanian (2001) on eesmärgiks võetud mikroobide kohanemisvõime modelleerimine. Konkreetsemalt on uuritud nn. glükoosiefekti evolutsioonilist väljakujunemist (nii laktoosi kui glükoosi olemasolul keskkonnas tarbib bakter kõigepealt ära suurema toiteväärtusega glükoosi ja seejärel asub laktoosi kallale). Rakk on esitatud suhteliselt lihtsal viisil, kasutades ainult kolme liiki agente: tuum, organell ja (sise)keskkond (joonis 9). Tuum jagab organellidele ajaressursse T_1 ja T_2, keskkond toitaineid C_1 ja C_2. Organelle on palju ja igaüks neist on kas olekus A või B, omades ligipääsu vastavalt ressurssidele T_1 ja C_1 või T_2 ja C_2 ning tekitades ressursside olemasolu korral vastavalt oma tootlikkusele juurde biomassi nii endale kui rakule. Eluspüsimiseks kasutavad organellid osa oma toodetud biomassist ära, selle lõppedes hukkuvad. Sobivate tingimuste korral nad poolduvad, üldjuhul kaheks sama tüüpi (A või B), kuid mutatsiooni toimudes vastupidist tüüpi organelliks. Oma olekut võivad agendid ka ise muuta, näiteks on teatud arvul neist lubatud jäljendada parajasti kõige edukamaid organelle. Raku kui terviku, samuti kõigi teda moodustavate agentide eesmärk on elus püsida ning kasvada. Tulemusena saadud rakumudel käitus simulatsioonides "adekvaatselt". Glükoosiefekti evolutsioonilise arengu uurimisel ei kasutatud aga enam eelkirjeldatud rakumudelit, vaid vaadeldi tervet rakku ühe teatud toitumisstrateegiat omava agendina ning kasutades geneetilisi algoritme simuleeriti populatsiooni arengut.



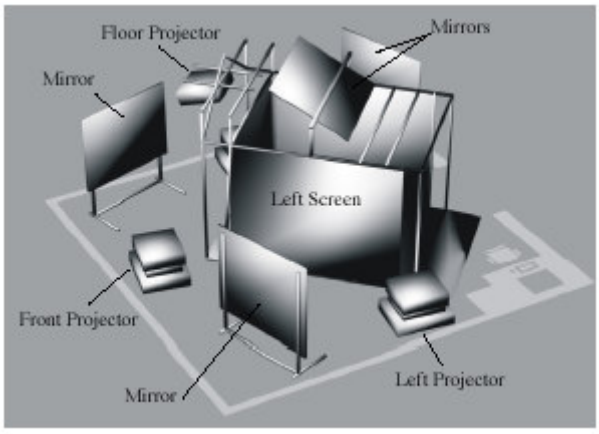
Joonis 10. *Lac*-operoniga seotud protsesside
3D visualisatsioon arvutiekraanil (Burleigh *et al*. 2003).

Allikas Burleigh *et al*. (2003) on kirjeldatud *lac*-operoniga seotud protsesside sülemipõhist *(swarm-based)* kolmemõõtmelist modelleerimist ja visualiseerimist (*lac*-operon on geenide süsteem (grupp), mis "vastutab" laktoosi glükoosiks muundamise eest). Iga simulatsioonis osalev element on iseseisev agent, mille käitumise määravad lihtsad interaktsioonireeglid. Mittepaiksed elemendid liiguvad virtuaalses rakus suvaliselt ringi ja teiste agentidega kohtudes käituvad vastavalt neile reeglitele. Pseudokoodis näide RNA polümeraasi juhtivatest reeglitest:

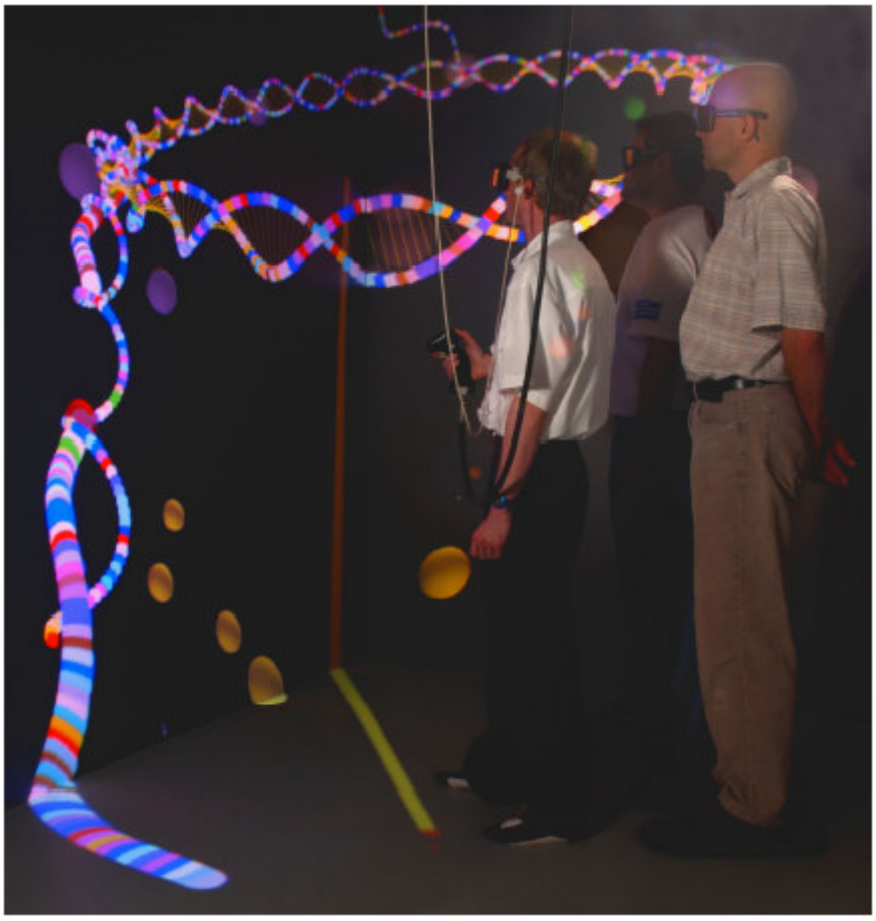
```
case state of

   FLOATING: /* initial state */
      if near DNA:
         attach self to nearest DNA codon
         state = DOCKED
      else:
         move randomly within the cell

   DOCKED:
      if promoter region is reached:
         state = READY_TO_TRANSCRIBE
      else:
         move along DNA to next codon
```

Jne.

Töö põhirõhk on rakus toimuvate protsesside kolmemõõtmelisel visualiseerimisel. Näide mudeli esitusest arvutiekraanil on toodud joonisel 10. Lisaks on olemas võimalus vaadelda simulatsiooni ka stereoskoopiliselt spetsiaalses süsteemis nimega *The CAVE Automated Virtual Environment* (joonised 11 ja 12).



Joonis 11. *The CAVE Automated Virtual Environment* (Burleigh *et al*. 2003).



Joonis 12. *Lac*-operoni mudeli 3D visualisatsioon Calgary ülikooli *CAVE*'s (Burleigh *et al*. 2003).

Artiklis Querrec *et al*. (2003) on reaktiivsete agentidena esitatud biokeemilised reaktsioonid. Agendi käitumistsükkel koosneb kolmest osast:

1. Agent loeb keskkonnast vastavas reaktsioonis osalevate ainete kontsentratsioonid.
2. Agent arvutab vastavalt kontsentratsioonidele reaktsiooni toimumise kiiruse ning selle alusel leiab, kui suur kogus reaktsioonis osalevaid aineid ära kaob ja juurde tekib.
3. Agent muudab vastavalt saadud tulemustele keskkonnas olevate ainete kontsentratsioone.

Järgmisena käivitatav agent-reaktsioon valitakse juhuslikult.

Khan *et al*. (2003) on agendi pannud vastavusse molekulide liigiga *(molecular-species-as-agent approach)*: agent sisaldab iga sellesse liiki kuuluva individuaalse molekuli (või arvutiressursside efektiivsema kasutamise nimel mitmest molekulist koosneva grupi) olekut. Samuti on agendil olemas nimekiri neist reaktsioonidest, milles tema poolt esitatav aine osaleda võib. Reaktsioone algatavad individuaalsed molekulid (või molekulide grupid) stohhastilise algoritmi alusel.

Artiklis Webb and White (*In Press*) on põhirõhk objektorienteerituse ja UML' i (*Unified Modeling Language)* kasutamisel biokeemias ja demonstreeritakse tarkvaralise vahendi *Rational Rose RealTime* rakendatavust raku modelleerimisel alates klassidiagrammide loomisest kuni käivitatava koodi genereerimiseni. Seejuures aga kasutatakse ka mõistet "aktiivne objekt" ning tõmmatakse paralleele eelkirjeldatud agentsüsteemiga Cellulat, kus agentide seosed on samuti eelnevalt jäigalt kindlaks määratud.

7. DnaA titratsiooni mudelil põhinev multiagentsüsteem

Järgnev kirjeldab minu poolt loodud agentmudelit. Kõigepealt on seletatud, miks üldse sellist mudelit oli mõtet valmistada, ja seejärel on esitatud modelleeritava bioloogilise protsessi kirjeldus, kasutatud tarkvaralised vahendid, mudeli enda kirjeldus, simulatsioonide tulemused ja lõpuks võimalikud suunad edasiseks arendustööks.

7.1. Milleks see mudel

Küsimused käesolevas töös loodud mudeli otstarbekuse kohta võib jagada kolme gruppi:

1. Miks simuleerida bakterit?
2. Miks simuleerida ühte bakterit multiagentsüsteemina?
3. Kas selliseid mudeleid juba teiste poolt tehtud ei ole?

Bakterite simuleerimine arvutiga on oluline mitmel põhjusel. Esiteks ei ole kõik neis toimuvad bioloogilised protsessid veel täpselt läbi uuritud, mistõttu tuleb jätkuvalt teha eksperimente. *In silico* katsetused on aga enamasti märksa lihtsamalt läbiviidavad kui eksperimendid bioloogiliste mikroorganismide endaga, pakkudes võimalust hüpoteeside esialgseks kontrollimiseks ja ideid reaalsete katsete edukamaks sooritamiseks. Teiseks võimaldab arvutimudeli koostamine ning kasutamine luua modelleeritavast objektist paremat ettekujutust: kõigepealt nõuab simulatsiooni koostamine bakteris toimuvate protsesside põhjalikku läbimõtlemist ning seejärel pakub teatud juhtudel võimalust toimuvat visualiseerida, vaadelda erinevate väärtuste muutumist ajas ning objektide liikumist bakteris. Mõlemad eelnimetatud põhjused on olulised nii teadusele kui tööstusele (eriti ravimi- ning toiduainetööstusele) ja samuti ka haridusasutustele.

Bakteri raku simuleerimine multiagentsüsteemina pakub eelkõige simulatsiooni "loomulikkust" ja paindlikkust. Loomulikkus seisneb võimaluses panna bioloogilistele objektidele või süsteemidele vastavusse sarnaselt käituvad (vajadusel proaktiivsed) tarkvaralised agendid, näiteks bioloogilisele DNA' le tarkvaraline DNA. Võrreldes diferentsiaalvõrrandite, temporaalloogika või muude (puht)matemaatiliste vahendite abil kirjeldatud rakuga on sellise agentmudeli struktuur inimesele kergemini mõistetav ning ka modelleerimise tulemus võib olla paremini tegelikkusele vastav. Paindlikkuse annab (headele) agentmudelitele võimalus suhteliselt vähese vaevaga agente lisada, modifitseerida ja eemaldada (s.h. süsteemi töö ajal), paralleeltöötluse tugi (s.t. agendid on jaotatud mitme arvuti vahel ja suhtlevad üle võrgu) ning võimalus erinevaid modelleeritavaid objekte esitada erineva detailsusega (samas simulatsioonis võib näiteks üks agent kujutada mõnda konkreetset molekuli ja teine kogu rakumembraani). Agentmudelis on tänu interaktsioonidel ja paralleeltöötlusel põhinemisele suhteliselt kerge saavutada ka ilmneva käitumise teket: süsteemi sellist käitumist, mis ei ole süsteemi kirjelduses ilmutatult esitatud ja seetõttu pakub modelleerijale suurt huvi (kui simulatsioonis toimuvad ainult sellised protsessid, mis on teadlikult ja ilmutatult ette kirjutatud, siis on modelleerijale selle virtuaalse süsteemi toimimine põhimõtteliselt juba ette teada ega paku muud huvi kui ehk arvutuste sooritamise ja süsteemi visualiseerimise seisukohast).

Küsimusele bakteri raku agentmudelite olemasolu kohta on põhjalikumalt vastatud eelnevas peatükis. Siinkohal võib lisada, et magistritööd tegema hakates ei olnud ma teadlik ühestki analoogsest mudelist ja eelkirjeldatud relevantsed tööd on leitud hiljem, intensiivse teadusajakirjade andmebaaside läbiotsimise tulemusena. Samuti kasutavad kõik nimetatud tööd erinevaid lähenemisviise ega oma suurt ühisosa ei omavahel ega

minu loodud mudeliga. Pealegi on mitmetes eelkirjeldatud töödes agendid omavahel väga jäigalt seotud või töötavad vaheldumisi, mis ei ole vastavuses agentorienteerituse oluliste märksõnadega nagu autonoomsus ja paralleelsus.

7.2. Modelleeritava bioloogilise protsessi kirjeldus

Bakteri sisemust saab olenevalt eesmärgist modelleerida mitmel erineval viisil ja keerukusastmel, alates mõne konkreetse protsessi isoleeritud vaatlemisest kuni raku käsitlemiseni tervikuna koos kõige tema sees toimuvaga. Käesoleva töö suund on raku kui terviku modelleerimine. Samas, kogu teadaoleva info agentmudelina kirjeldamine vajaks suurt töögruppi ning aastatepikkust intensiivset tööd. Seetõttu on siin valitud kompromissvariant: näidata agentmudeli kasutatavust esialgu küll peamiselt ainult ühte protsessi modelleerides (ja sedagi tugevalt lihtsustatult), kuid valida selline protsess, mis seob omavahel mikro- ja makrotaseme (bakteri suhtes) näitajad ning omab raku elus suurt tähtsust.

Üldisemas plaanis on kõigi bioloogiliste organismide põhieesmärk (või silmapaistvaim omadus, kui loodusele eesmärke mitte omistada) elus püsida, mis pikemas perspektiivis (indiviidi eluiga ületaval ajaskaalal) toimub paljunemise teel. Eriti aktuaalne on see teema mikroorganismide korral, sest võrreldes näiteks inimesega on nende kognitiivsed ja sotsiaalsed võimed üsna piiratud (kuigi mitte olematud, vt. näiteks Jacob *et al*. (2004)) ja ilmselt ei võimalda neil enda jaoks defineerida teistsuguseid elueesmärke nagu inimestel tavaks.

Bakterid paljunevad pooldumise teel. Raku pooldumine on edukas siis, kui mõlemad tütarrakud saavad kaasa liiki kirjeldava DNA ning piisavalt biomassi (koosnevana eluprotsessi ülalhoidvatest ainetest ja struktuuridest), et elus püsida ja olla jätkuvalt kasvamis- ja paljunemisvõimelised. Raku pooldumiseni viiva protsesside ahela alguseks, või vähemalt väga olulist rolli mängivaks lüliks, võib pidada DNA replikatsiooni (kopeerimise) alustamist. Just see algus ning tema seos raku massiga ongi valitud käesoleva töö modelleerimisobjektiks. Kuigi DNA replikatsiooni regulatsiooni molekulaarsed mehhanismid ei ole veel päris täpselt kindlaks määratud, peetakse kõige tõenäolisemaks, et põhiregulaatoriks on valk nimega DnaA. Sellel põhinevat prokarüootsete (eeltuumsete) rakkude kohta käivat teooriat ongi järgnevalt allikate Herrick *et al*. (1996) ja Abner (2003) alusel lühidalt ja tugevalt lihtsustatult kirjeldatud, lähtudes bioloogia mudelorganismide hulka kuuluvast bakterist *Escherichia Coli*.

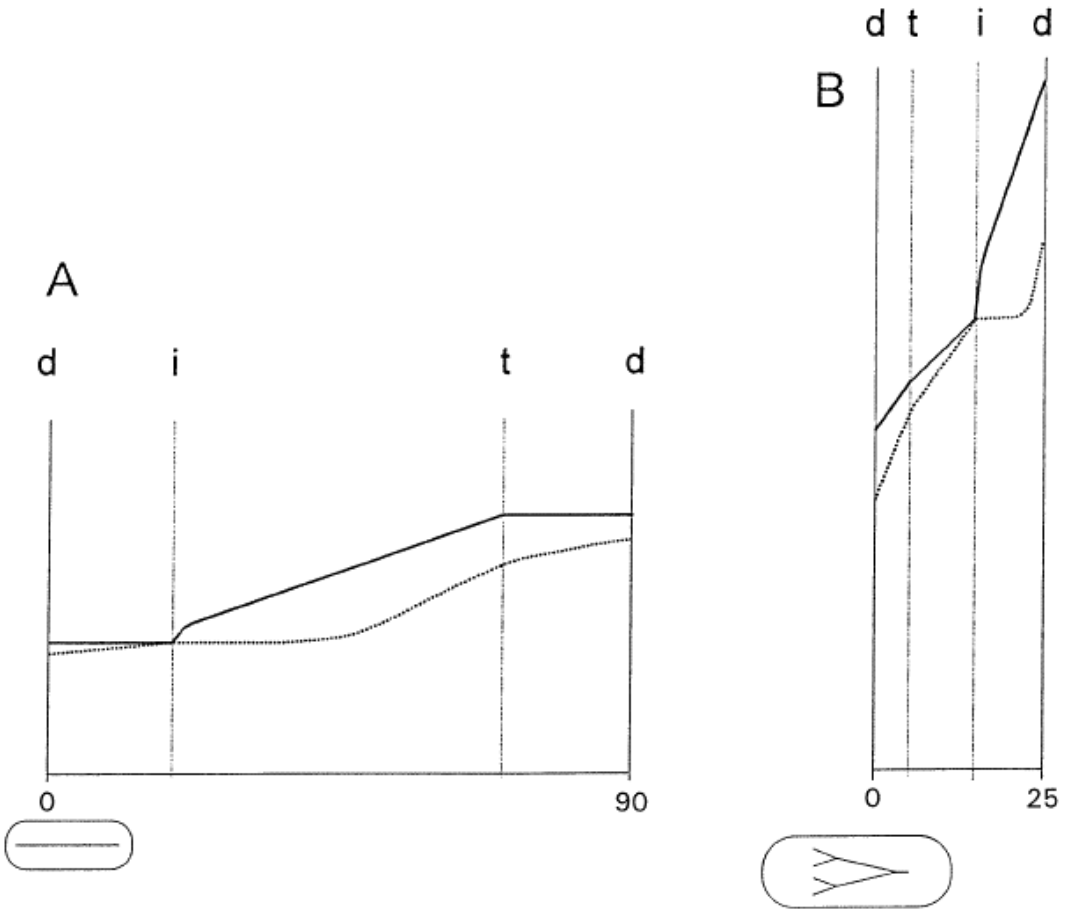
1960.-tel aastatel ja hiljem läbi viidud eksperimendid näitasid, et:

- DNA replikatsiooni aeg (mida tavaliselt tähistatakse C-ga) ja aeg replikatsiooni lõpust raku jagunemiseni (D) on konstantsed, *E. Coli* puhul vastavalt ~40 ja ~20 minutit.
- DNA replikatsiooni initsieerimine toimub ainult teatud rakumassi, nn. initsiatsioonimassi korral, mille suhe DNA replikatsiooni alguspunktide arvuga on konstantne. (Igal DNA ahelal on üks nn. oriC *(origin)* piirkond, millest kopeerimine peale hakkab. Kopeerimise tulemusena tekib kohe juurde teine oriC, mis asub uuel loodaval DNA' l. Kui tegu on kiiresti poolduva rakuga ja järgmine DNA kopeerimine käivitatakse enne eelnevast replikatsioonist tingitud raku pooldumist, siis on rakus initsiatsiooni hetkel korraga mitu oriC' d ja initsiatsioonimass on vastav arv kordi suurem.)
- Raku kasv on määratud kasvutingimustega ega sõltu DNA replikatsioonist.

Seda prokarüootse rakutsükli algoritmilise olemuse kirjeldust, mida nimetatakse tema peamiste loojate järgi Cooper-Helmstetter-Donachie teooriaks, peetakse üldiselt mõistlikuks ja adekvaatseks, kuid tema puuduseks on makrotaseme mõiste "initsiatsioonimass" empiirilisus: ei ole ära näidatud seost mikrotasemega, s.t. pole kirjeldatud initsiatsiooni tekitavaid molekulaarseid mehhanisme.

Edasised katsed näitasid, et initsiatsioonimassi suhe oriC' de arvu siiski päris konstantne ei ole ja seda saab mõjutada teatud valgu – DnaA – kontsentratsiooni (kunstliku) muutmisega rakus. See koos muu järjest lisanduva infoga raku ehituse ja toimimise kohta viis 1980.–' 90.-tel aastatel uue, nn. IT mudeli (*Initiator Titration Model*; initsiaatori tiitrimise mudel) loomiseni, mille kohaselt initsiatsioon algatatakse siis, kui väga aeglaselt akumuleeruv DnaA valk (nt. raku pooldumisaja 40 min. korral kulub 1 monomeeri lisandumiseks ~6 sekundit) saavutab kriitilise kontsentratsiooni (või koguse). DNA' l asuvad DnaA' d siduvad kohad ehk DnaA boksid, mis võivad olla kõrge või madala afiinsusega ("külgetõmbejõuga"). Kokku on neid üle 300, sealhulgas mõned kõrge ja mõned madala afiinsusega boksid oriC piirkonnas. Kui DnaA kontsentratsioon kasvab tasemele, mille juures toimub seondumine ka madala afiinsusega boksidel oriC piirkonnas, moodustub seal suur kompleks ~20 DnaA molekulist ning mitmetest teistest molekulidest, mis viib DNA kaksikheeliksi lahtihargnemiseni oriC' s ja DNA replikatsiooni alustamiseni. Kopeerimise käivitumine omakorda vabastab oriC piirkonnast hulga sinna seondunud DnaA' d, tõstes järsult DnaA kontsentratsiooni raku selles ruumipiirkonnas ning põhjustades nn. poolsünkroonset initsiatsiooni teistel läheduses asuvatel DNA' del (kui neid juhtub olema, s.t. kiiresti poolduvate rakkude korral). Lisaks põhjustab replikatsiooniprotsess oriC inhibeerumise mõneks ajaks, mis väldib momentaalse reinitsiatsiooni teket. Samuti inhibeerub teatud ajaks oriC lähedal asuv DnaA promootor ehk DNA piirkond, kus asub DnaA tootmiseks vajalik info. Selle tagajärjel DnaA tootmine mõnda aega seisab, rakus olemasolevad DnaA molekulid seonduvad uu(t)e loodava(te) DNA ahela(te) külge ning DnaA kontsentratsioon rakus väheneb. IT mudeli poolt väljapakutav rakus oleva DnaA valgu koguhulga ja DnaA bokside koguarvu muutumine bakteri elutsükli jooksul on toodud joonisel 13.

NB! Siinkohal on veelkord mõistlik tähelepanu juhtida sellele, et esiteks on eeltoodud kirjeldus tugevalt lihtsustatud variant tegelikust teooriast ning teiseks ei ole ka teooria ise veel lõplikult tõestatud.



Joonis 13. DnaA bokside koguarvu (tugevam joon) ja DnaA valgu koguhulga (punktiir) muutumine rakutsükli jooksul. *d = division* (raku pooldumine)*, i = initiation* (DNA replikatsiooni alustamine)*, t = termination* (DNA replikatsiooni lõpp). Vasakul pooldumisaeg > C + D (s.t. raku pooldumisaeg on suur, DNA kopeerimine ja sellele järgnev "ooteaeg" jäävad kõik sama rakutsükli sisse). Paremal pooldumisaeg << C + D (s.t. pidevalt toimub mitu DNA replikatsiooni protsessi korraga ja kui sellise bakteri DNA sirgeks tõmmata (bakteris on DNA ringina, see omakorda "puntraks" kokku keerdunud), siis tulemus meenutab puud: vt. paremal all toodud näidet, kus replikatsioonikahvlite liikumissuund on vasakult paremale). (Herrick *et al*. 1996)

7.3. Kasutatud tarkvaralised vahendid

7.3.1. JADE 3.3

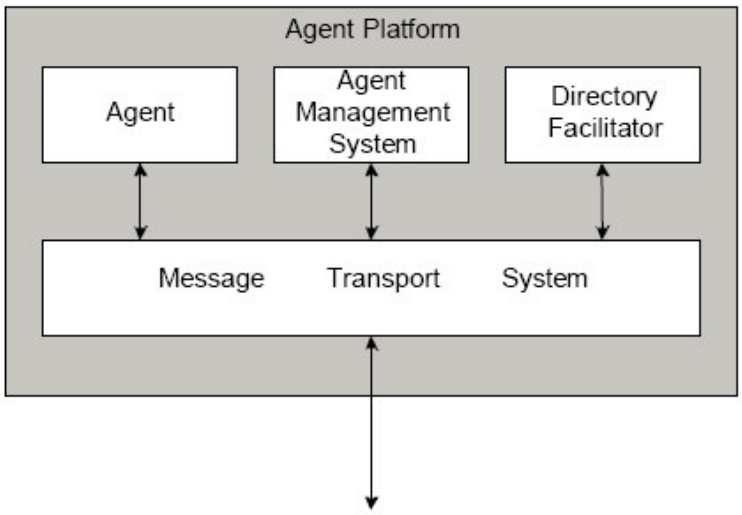
Agendiarenduskeskkonnana on käesolevas töös kasutatud vahendit JADE (*Java Agent DEvelopment framework*, http://jade.tilab.com/). Valiku peamine põhjus oli asjaolu, et seda on TTÜ Automaatikainstituudis ka varem kasutatud (Vadim Kimlaychuk'i poolt) ning on kindlaks tehtud vahendi sobivus sedalaadi keerukate agentsüsteemide valmistamiseks. Teiste tööriistade (vt. nimekirja agente tutvustava peatüki alapunktis "Agent-orienteeritud tarkvaratööriistad") sobivuse kontrollimine ja neist antud ülesande jaoks parima väljavalimine oleks olnud väga aeganõudev protsess ning ei olnud käesoleva töö eesmärgi saavutamiseks vajalik. Küll aga tuleb seda ilmselt teha juhul, kui tööd on kavas edasi arendada ja biotehnoloogidele praktikas kasutatavaks muuta. Järgnevas on antud JADE'i lühikirjeldus Caire ja Bellifemine *et al*. poolt kirjutatud JADE'i programmeerimisjuhendite põhjal.

JADE on vahetarkvara *(middleware)* multiagentsüsteemide loomise hõlbustamiseks, sisaldades:

- Käituskeskkonda *(runtime environment)*, kus JADE agendid "elavad".
- Teeki *(library)* agentide valmistamiseks.
- Graafilisi vahendeid käituskeskkonna ja töötavate agentide administreerimiseks ning jälgimiseks.

JADE'i võimalused ja omadused:

- Hajutatus: agendiplatvormi on võimalik jagada laiali üle mitme seadme. Igas seadmes piisab ühe Java rakenduse jooksutamisest ja seega ühest Java virtuaalmasinast: agendid on realiseeritud lõimedena *(threads)* ning elavad platvormi alla kuuluvates agendikonteinerites.
- Graafiline kasutajaliides, mille kaudu saab teostada ka kaughaldust, s.t. administreerida teistes seadmetes olevaid agente.
- Silumisvahendid *(debugging tools)*.
- Agentide mobiilsus platvormi piires.
- Agendi SEES toimuvate protsesside mõningase paralleliseerimise võimalus (tugi pseudoparalleelsusele, minimaalne tugi ka tõelisele paralleelsusele). Erinevate agendiEKSEMPLARIDE paralleelsele töötamisele tarkvara poolt piiranguid ei ole.
- Platvorm vastab FIPA *(Foundations of Intelligent Physical Agents)* standarditele, sisaldades agendihaldussüsteemi *(Agent Management System, AMS)*, ühte või mitut samaaegset kataloogiteenust *(Directory Facilitator, DF)* ja agendi-kommunikatsioonikanalit *(Agent Communication Channel, ACC)* (joonis 14).
- Efektiivne ACL *(Agent Communication Language)* sõnumite transport: platvormi piires Java objektidena, mitte stringidena. Väljapoole platvormi liikuvatele sõnumitele antakse automaatselt FIPA standarditele vastav süntaks, kodeering ja transpordiprotokoll.
- FIPA interaktsiooniprotokollide teek.
- Agentide automaatne registreerimine ja deregistreerimine AMS'is.
- FIPA standarditele vastav nimeteenus: agentidele antakse nende käivitumisel globaalselt unikaalne identifikaator *(Globally Unique Identifier, GUID)*.



Joonis 14. FIPA agendiplatvormi baasarhitektuur (Bellifemine *et al*.).

(JADE'i omaduste nimekirja jätk)

- Toetab rakenduse poolt defineeritud suhtlemiskeeli ja ontoloogiaid.
- *InProcess* liides, mis võimaldab platvormivälistel rakendustel käivitada autonoomseid agente.

Agentide funktsionaalsus on JADE' is esitatud "käitumistena" *(behaviours)*: programmeerija defineerib teegis olevale klassile *Behaviour* või selle alamklassidele (*TickerBehaviour, CyclicBehaviour* jt.) oma alamklassi(d) ja lisab neist tekitatud objektid agendi käivituskoodis või ka hiljem töökoodis käsuga *addBehaviour(Behaviour)* agendi käitumiste nimekirja, kust agendi plaanur *(scheduler)* neid käivitab. JADE' i poolt pakutavad käitumise baasklassid on näidatud joonisel 15 ning agendilõime tööpõhimõte joonisel 16.

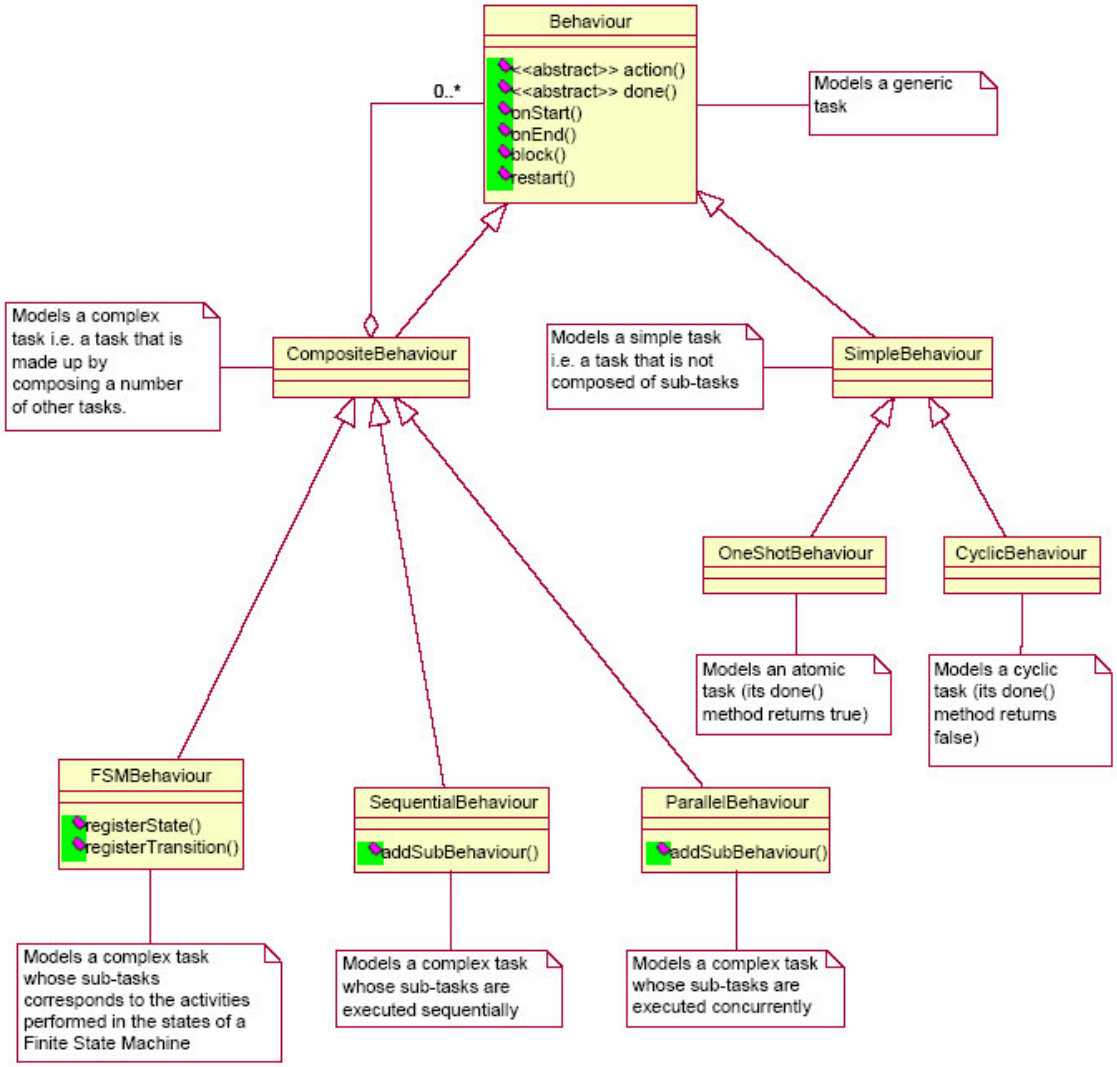
Agentide omavaheline suhtlemine toimub üksteisele sõnumite saatmise teel. Sõnumite saatmine on asünkroonne: saadetavad sõnumid asetatakse agendi sõnumijärjekorda ("postkasti"), kust agent siis neid soovi korral üles korjab ja töötleb (joonis 17).

7.3.2. Java 2 Platform, Standard Edition (J2SE)

Kuna JADE on Java-põhine, siis tuleb süsteemi jooksutamiseks kasutada *Java Runtime Environment*' i ja arendustööks *Java Software Development Kit*' i, mis on kättesaadavad aadressilt http://java.sun.com/j2se/index.jsp. Käesolevas töös olid kasutusel versioonid *j2re1.4.2_05-b04* ja *j2sdk1.4.2_05*.

7.3.3. JGraphT

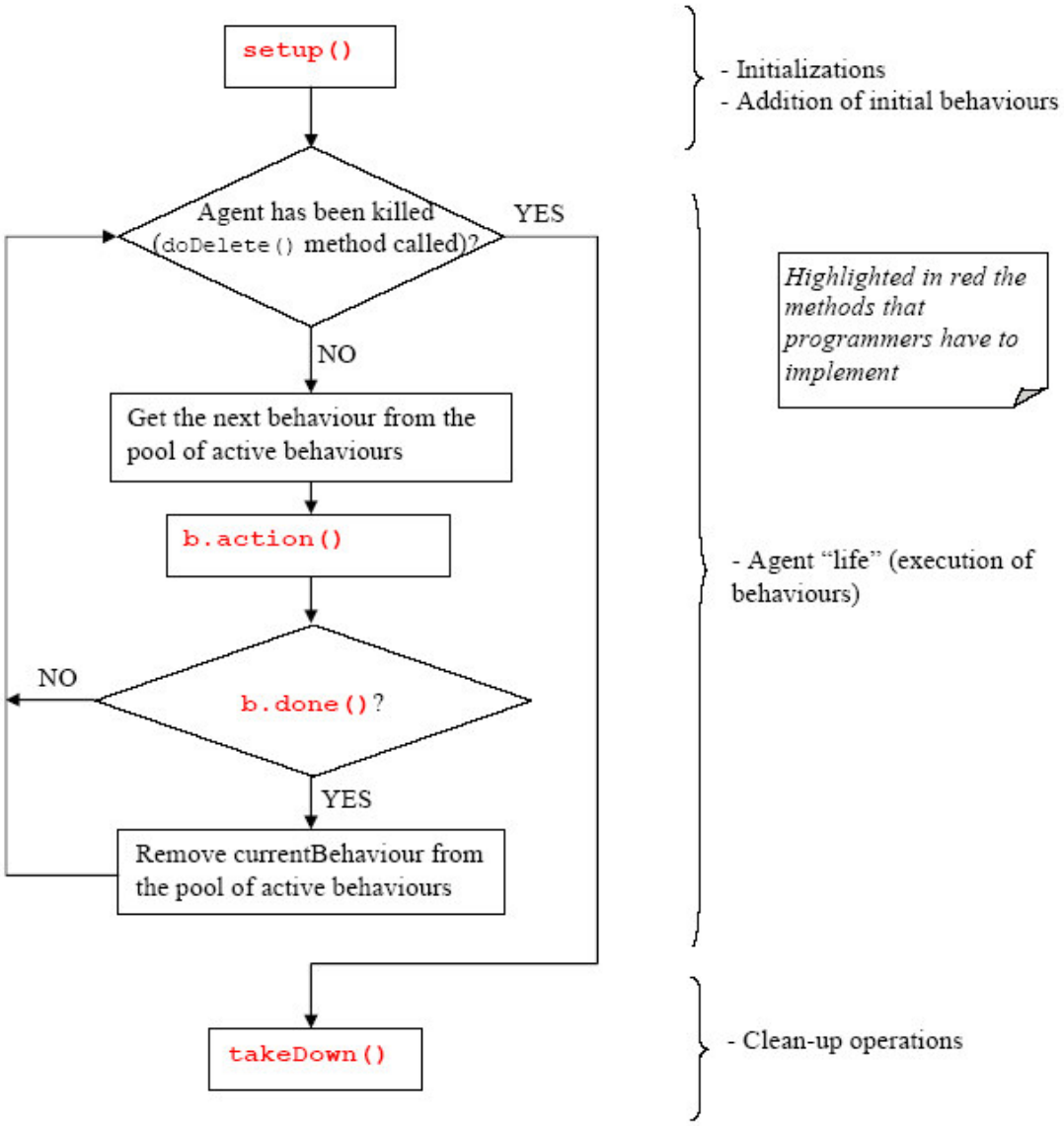
Replitseerimise ajal üksteisega seotud DNA' de haldamiseks kasutasin teeki *JGraphT*, mis on kättesaadav lehelt http://jgrapht.sourceforge.net/. *JGraphT* pakub kasutamiseks mitmesuguseid matemaatilise graafiteooria objekte ja algoritme. Käesolevas töös oli kasutusel versioon *jgrapht-0.5.3*. Kuna selgus, et teegil on probleeme serialiseeritavusega, siis tuli tema lähtekoodis teha paar väikest parandust, nimelt klassidele *DefaultEdge*, *DirectedEdge* ja *AbstractBaseGraph::Specifics* sai lisatud *implements Serializable*.



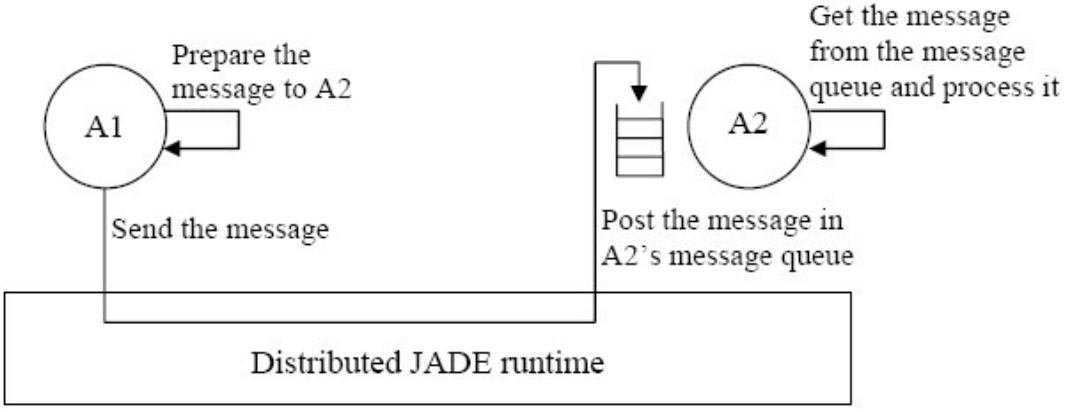
Joonis 15. JADE teegis asuvate agendi käitumist defineerida võimaldavate klasside hierarhia. Kõiki pakutavaid klasse pole siin siiski välja toodud, näiteks taimerite poolt käivitatavaid käitumisi. (Bellifemine *et al.*)

7.3.4. JChart2D

Süsteemi töö ajal agentide muutujate väärtuste visualiseerimiseks kasutasin teeki *JChart2D*, mis on kättesaadav lehelt http://jchart2d.sourceforge.net/. *JChart2D* võimaldab ilma suurema vaevata ekraanile kuvada joongraafikuid, mis andmepunktide lisamisel ise edasi kerivad ja vajadusel skaalasid muudavad, et info täpselt ära mahuks. Käesolevas töös oli kasutusel versioon *jchart2d-1.03*.



Joonis 16. JADE' i agendilõime tööskeem (Caire).



Joonis 17. Asünkroonne sõnumite saatmine JADE' is (Caire).

7.4. Loodud mudeli kirjeldus

Valminud agentmudel esitab tugevalt lihtsustatult bakteri *E. Coli* DNA replikatsiooni initsieerimist, replikatsiooni ennast ja nende seost raku massiga vastavalt alapeatükis "Modelleeritava bioloogilise protsessi kirjeldus" antud mudelile, kus replikatsiooni regulaatoriks on valk DnaA.

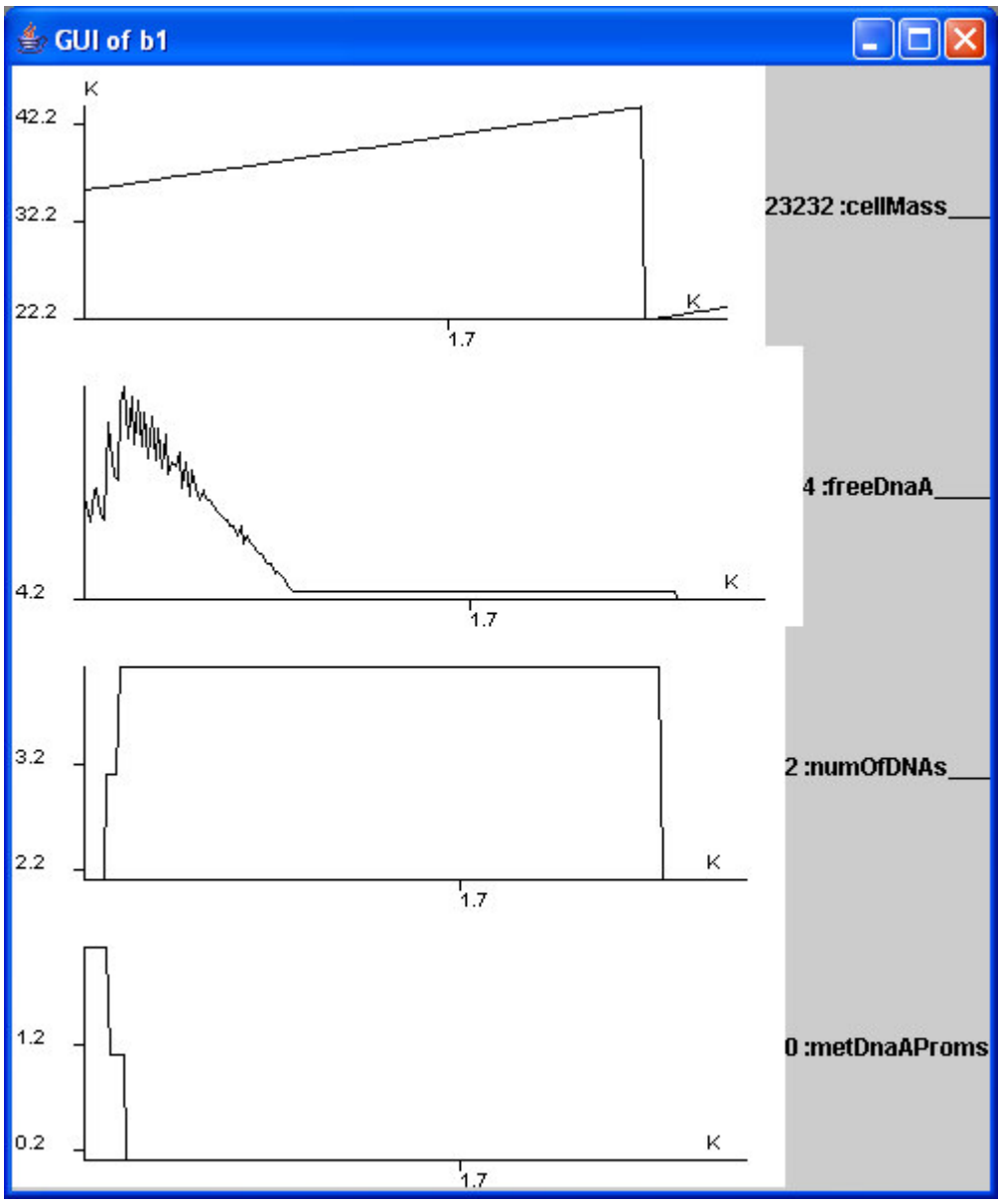
Mudel koosneb nelja liiki agentidest: Keskkond, Bakter, DnaA Tootja ja DNA. Järgnevas on antud nende agentide üldisem sõnaline kirjeldus, kus kõiki tehnilisi detaile eraldi välja ei ole toodud. Täpsemat infot on võimalik leida töö lisana antud lähtekoodist. Diagrammidena ei ole mudelit esitatud, kuna esiteks on tegu suhteliselt väikese ja ka sõnalise kirjelduse kaudu üsna kergesti mõistetava mudeliga ning teiseks ei sobi tavalised UML klassidiagrammid, mida on lihtne Java koodist genereerida, multiagentsüsteemide kujutamiseks.

Keskkond *(AgEnvironment)* koondab endas kogu bakterit ümbritseva keskkonna.

- Lihtsuse mõttes on kõik keskkonnatingimused (temperatuur, toitained, mürkained jms.) hetkel esitatud koos, üheainsa arvuna: mida suurem, seda parem bakteri jaoks.
- Kui Keskkonnale saata küsimus keskkonnatingimuste kohta, siis ta saadab küsijale tagasi sõnumi tingimuste hetkeväärtusega.
- Keskkonnatingimused ei muutu automaatselt, vaid neid (seda) saab muuta Keskkonnaagendile vastavat sõnumit saates. Hetkel töötab simulatsioon mõistlikult tingimuste vahemikus ~15..60 (suvalised ühikud). See piirang tuleneb muuhulgas reaalsest ajast sõltuvate taimerite kasutamisest agentide tegevuste käivitamiseks: liiga "heade" tingimuste korral toimuvad protsessid nii kiiresti, et ettemääratud andmetöötlushetkede vahel toimuvad muutused on liiga suured ja löövad süsteemi tasakaalust välja; liiga "halbades" tingimustes aga võib probleemseks osutuda näiteks arvutuste täpsus, kuna hetkel on lihtsuse mõttes paljudes kohtades kasutatud täisarve. Ilmselt tuleb mudelit edasi arendades asendada astronoomilise aja kasutamine arvutiajaga. See võimaldaks ka arvutiressursside efektiivsemat ärakasutamist – simulatsiooni nii kiiret tööd, kui arvuti parajasti suudab.

Bakter *(AgBacterium)* esindab tervet bakteri rakku, hallates raku osi kujutavaid agente (DNA(d), DnaA Tootja) ning simuleerides ülejäänud osade, mida ei ole agentidena esitatud, summaarset käitumist.

- Raku mass suureneb vastavalt keskkonnatingimustele. Praeguse seisuga kasutatakse väga lihtsat lineaarset kasvu: keskkonnatingimusi kujutav arv lihtsalt liidetakse iga natukese aja tagant (100 ms) bakteri massile.
- Bakter pollib (küsitleb) rakule kuuluvaid DNA' sid, et teada saada aktiivsete DnaA promootorite koguarvu, millest sõltub DnaA Tootja töökiirus (lihtsuse mõttes küsib DnaA Tootja praegu seda infot Bakteri käest, muidu peaks ta kõigepealt Bakterilt küsima sellele kuuluvate DNA' de nimed ja seejärel ise igaühe käest infot paluma).



Joonis 18. Bakteri infoaken.

- Bakter säilitab DnaA Tootjalt ja vahel ka DNA' delt sõnumitena sissetulevat vaba DnaA' d ning jagab seda, jällegi sõnumite vahendusel, küsijatele (DNA' d) sõltuvalt vaba DnaA koguhulgast: mida suurem koguhulk, seda suurema koguse küsija saab (et ära hoida olukorda, kus mõni DNA parajasti suudab siduda suure koguse DnaA' d ja rakus ka tõepoolest on palju vaba DnaA' d, aga nende ühekaupa saatmine võtab kaua aega ning venitab sidumisprotsessi ebarealistlikult pikaks). Vaba DnaA kaotsiminek protseduuri käigus on välditud sellega, et kui DNA saab vajatavast rohkem DnaA' d, siis ta saadab ülejäägi lihtsalt tagasi.
- Kui mõnelt DNA' lt tuleb sõnum, et DNA replikatsioon on lõppenud, siis käivitub raku pooldumiseelne taimer (konstantne, 20 sek.).
- Kui taimer on teavitanud replikatsioonijärgse ooteaja lõppemisest, siis tekitatakse juurde uus Bakter, antakse talle pool raku massist ja vabast DnaA' st ning ligikaudu pool DNA' dest (vastavalt sellele, millised DNA' d on ja millised ei ole omavahel ühendatud: kui tegu on kiire rakutsükliga bakteriga, kus ka pooldumise ajal mõnda DNA' d parajasti kopeeritakse, nii et seal uus DNA on vanemaga

ühendatud, siis neid mõistagi lahku eitõmmata). Kui süsteem on režiimis "singlecell", siis tekkinud tütarrakk hävitatakse, et hoida kokku arvutiressursse.

- Kui Bakterile saata küsimus raku massi, vaba DnaA hulga, DNA'de arvu ja nende omavaheliste seoste või aktiivsete DnaA promootorite koguarvu kohta, siis ta saadab küsijale tagasi sõnumi vastava muutuja hetkeväärtusega.

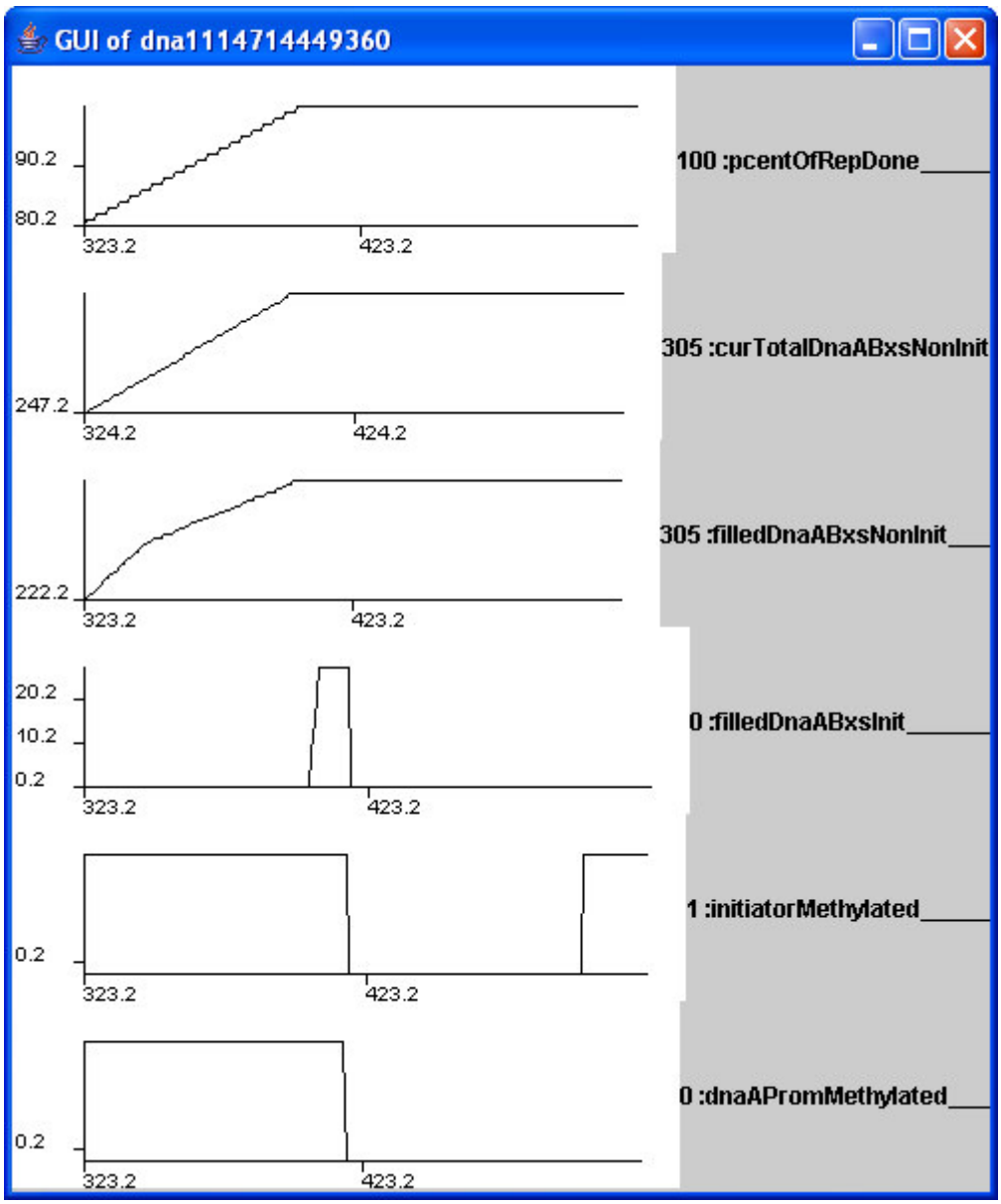
Bakteri oleku jälgimise lihtsustamiseks kuvatakse mõnede muutujate väärtused infoaknasse (joonis 18). Samuti logitakse osa infot tekstifaili.

DnaA Tootja *(AgDnaAFactory)* valmistab sõltuvalt keskkonnatingimustest ja rakus olevate aktiivsete DnaA promootorite arvust valku DnaA ning saadab selle agendile Bakter. Praeguse seisuga määratakse toodetava DnaA hulk järgnevalt: iga (6000 / keskkonnatingimused) millisekundi järel toodetakse (aktiivsete DnaA promootorite arv) DnaA valgu molekule. Igas Bakteris on üks DnaA Tootja.

DNA *(AgDNA)* kujutab kas valmisolevat või alles valmivat DNA topeltheeliksit.

- Kui DnaA boksid ei ole täis, siis küsib Bakterilt vaba DnaA'd.
- Täidab saadud DnaA'ga replikatsiooni initsiaatorpiirkonnast oriC väljaspool asuvaid bokse, mida on 305 tk.
- Kui need saavad täis, oriC ei ole inhibeeritud ja rakus on vaba DnaA'd üle 40 molekuli (sellel, nagu enamusel teistelgi arvudel selles mudelis, ei ole üks-ühest seost reaalse bakteri muutujate väärtustega), siis täidab initsiaatorpiirkonnas asuvaid bokse, mida on 30 tk. Nõue 40 vaba DnaA molekuli olemasolule on mõeldud selleks, et ligikaudselt jäljendada osade oriC's asuvate DnaA bokside madalat afiinsust, mis nõuab nende täitmiseks kõrgemat DnaA kontsentratsiooni.
- Kui initsiaator saab täis, alustatakse DNA replikatsiooni: initsiaatori DnaA visatakse tagasi Bakterisse, oriC ja DnaA promootor inhibeeritakse mõneks ajaks (oriC kuni ajani, mil on toimunud 20% replikatsioonist, DnaA promootor vastavalt 55% -ni; samad inhibeerituse ajad kehtivad ka kopeerimise käigus loodavale DNA'le) ning tekitatakse uus "mittevalmis" DNA agent.
- Replikatsiooniprotsess toimub kiirusega 0.25 protsendipunkti iga 100 ms järel, ja lõpeb 100% saavutamisel. Kuna praeguses lähenduses DNA pärilikkusinfot veel ei kanna, siis kopeerimine piirdubki peamiselt selle protsendi kasvatamisega ja veel kahe protsessiga: uue DNA väljaspool initsiaatorpiirkonda olevate DnaA bokside arv kasvab replikatsiooniprotsessi käigus nullist 305-ni; ja kopeeritava DNA need DnaA boksid, millest replikatsioonikahvel parajasti "üle sõidab", tühjendatakse ning vabanenud DnaA saadetakse tagasi Bakterile. Hetkel on eeldatud, et boksid on jaotunud ühtlaselt üle kogu DNA (reaalsuses päris ühtlaselt ei ole).
- Kui DNA'le saata küsimus teda omava Bakteri nime, tema vanemaks olnud DNA nime või DnaA promootori inhibeerituse kohta, siis ta saadab küsijale tagasi sõnumi vastava muutuja hetkeväärtusega.

Nagu Bakteri, nii ka DNA oleku jälgimise lihtsustamiseks kasutatakse infoakent, mille näidis on toodud joonisel 19.

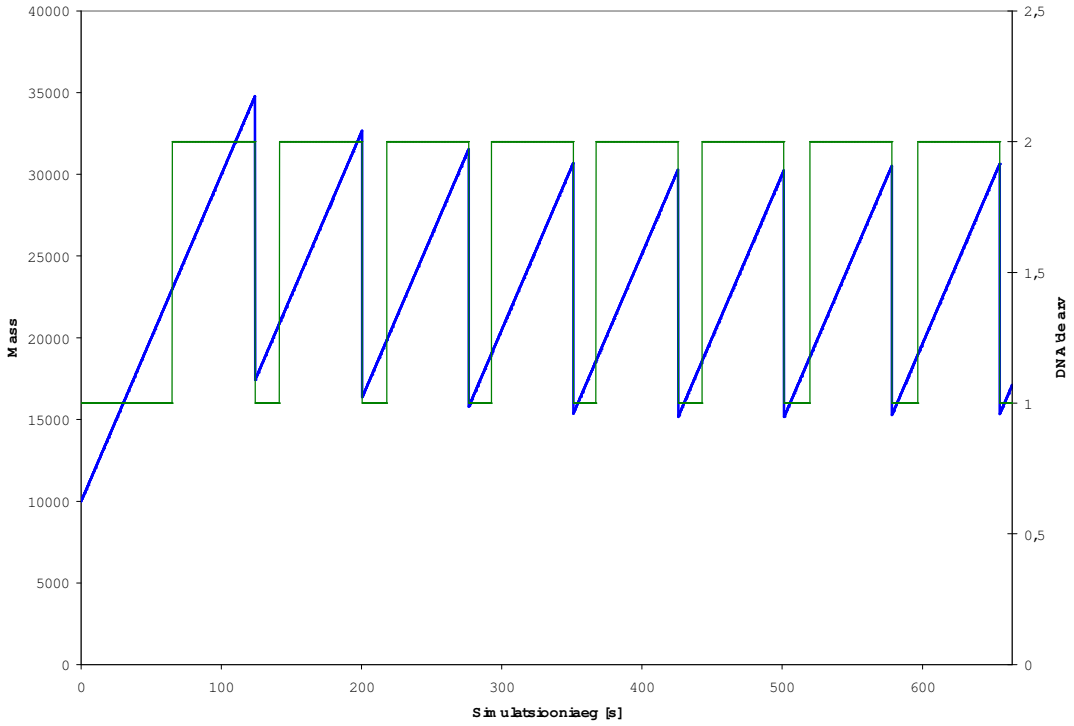


Joonis 19. DNA infoaken.

7.5. Tulemused

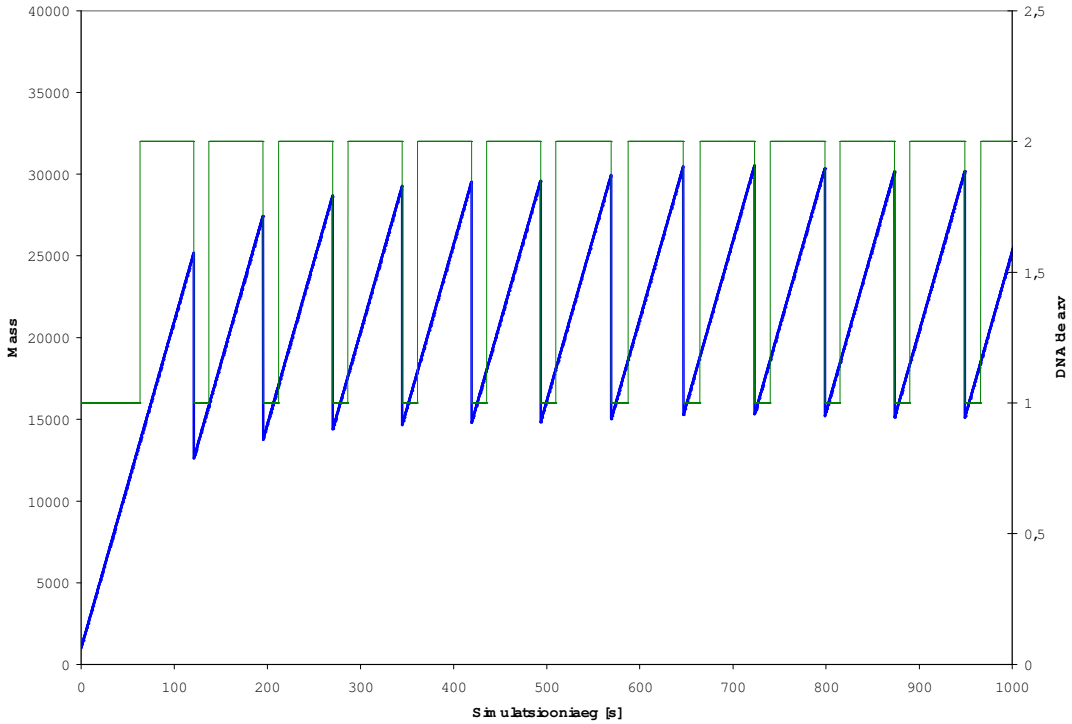
Eelkirjeldatud agentmudeli jooksutamisel arvutis oli võimalik välja selgitada mudeli käitumine ning järgnevas on võrreldud tulemusi reaalsete bakterite kohta teadaolevate andmetega.

Esimene küsimus käesoleva mudeli puhul on kindlasti: "Kas paljunemine üldse hakkab toimuma?". Nagu näha jooniselt 20, hakkab rakk tõepoolest poolduma, kusjuures stabiilsete keskkonnatingimuste korral kujuneb välja enam-vähem püsiv pooldumis-periood. Samuti stabiliseerub raku keskmine mass. DNA' de arv on sellel ja järgnevatel joonistel esitatud põhimõttel, et hüpe +1 toimub kohe uue DNA valmistamise algul, kui mudelis luuakse uus DNA agent (ehk siis bioloogilises mõttes on tegu pigem oriC' de arvuga rakus).

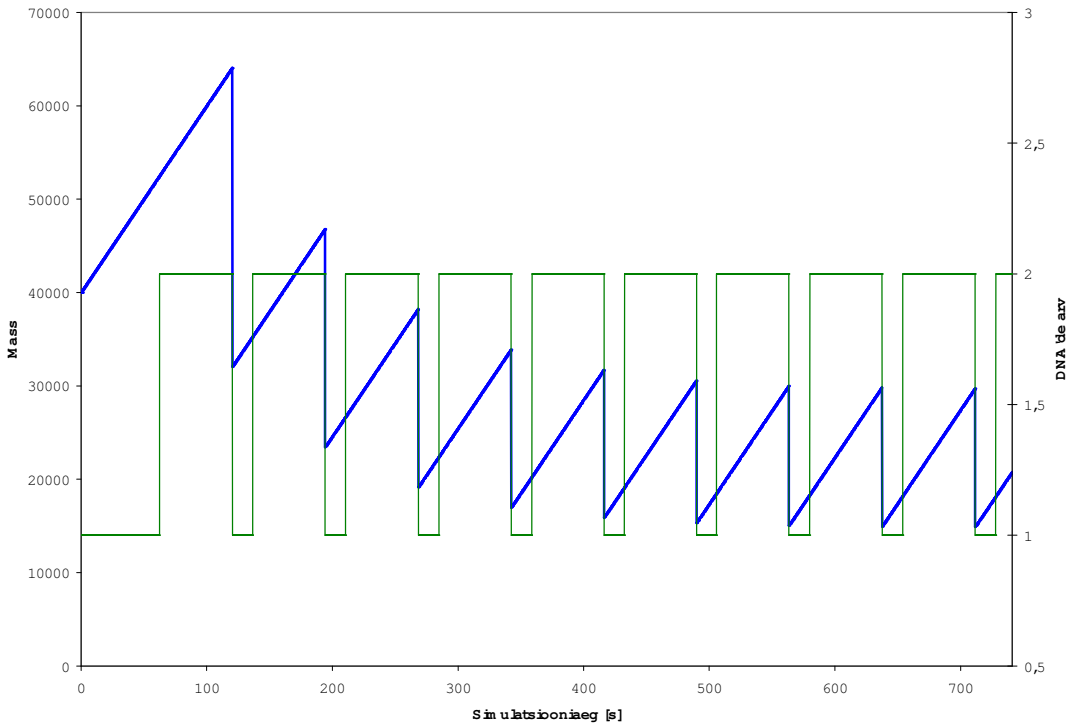


Joonis 20. Perioodiline pooldumine.
Stardimass 10 000, keskkonnatingimused 20.
Jämedam joon on raku mass, nõrgem DNA' de arv rakus.

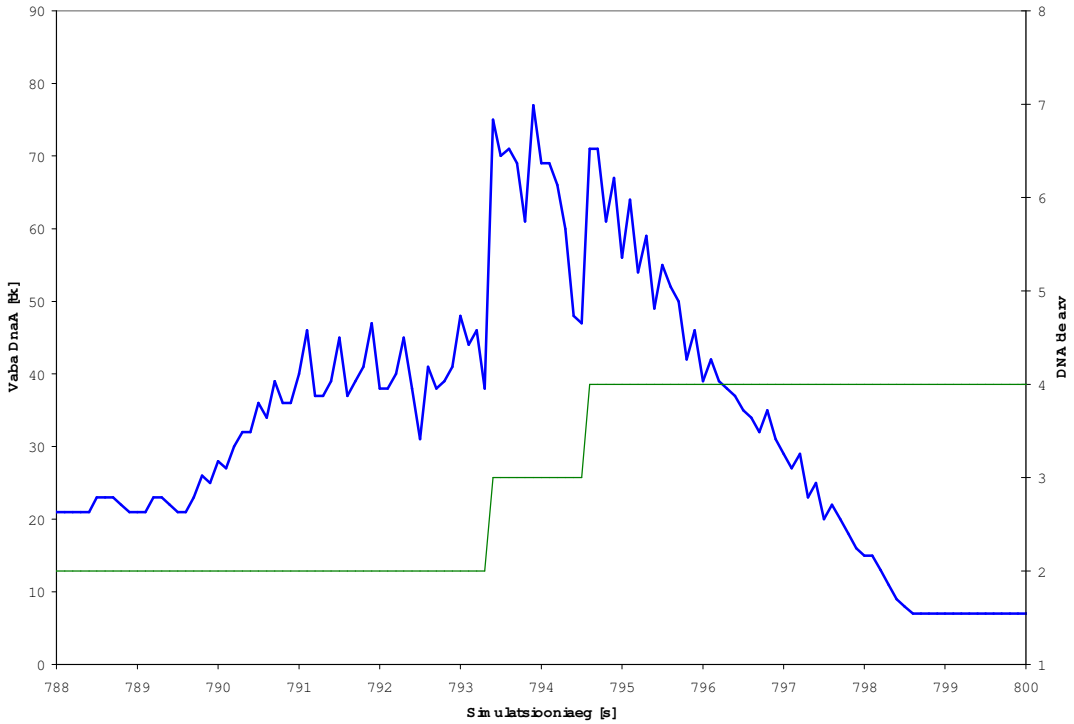
Programmeerija poolt (mõistlikes piirides) etteantud stardimassi mõju hiljem väljakujunevale stabiilsele (perioodiliselt võnkuvale) olekule võiks olla väike, kuna stabiilsete keskkonnatingimuste puhul peaks enamusel sama liiki bakteritel välja kujunema enam-vähem sarnane massi perioodilise kõikumise vahemik. Et see loodud mudelis tõepoolest nii toimub, on näidatud joonistel 21 ja 22, kus nii normaalsest oluliselt madalama kui ka kõrgema stardimassi korral stabiliseerutakse lõpuks samasse vahemikku nagu joonisel 20.



Joonis 21. Normaalsest väiksema stardimassi mõju.
Stardimass 1000, keskkonnatingimused 20.
Jämedam joon on raku mass, nõrgem DNA' de arv rakus.



Joonis 22. Normaalsest suurema stardimassi mõju.
Stardimass 40 000, keskkonnatingimused 20.
Jämedam joon on raku mass, nõrgem DNA' de arv rakus.



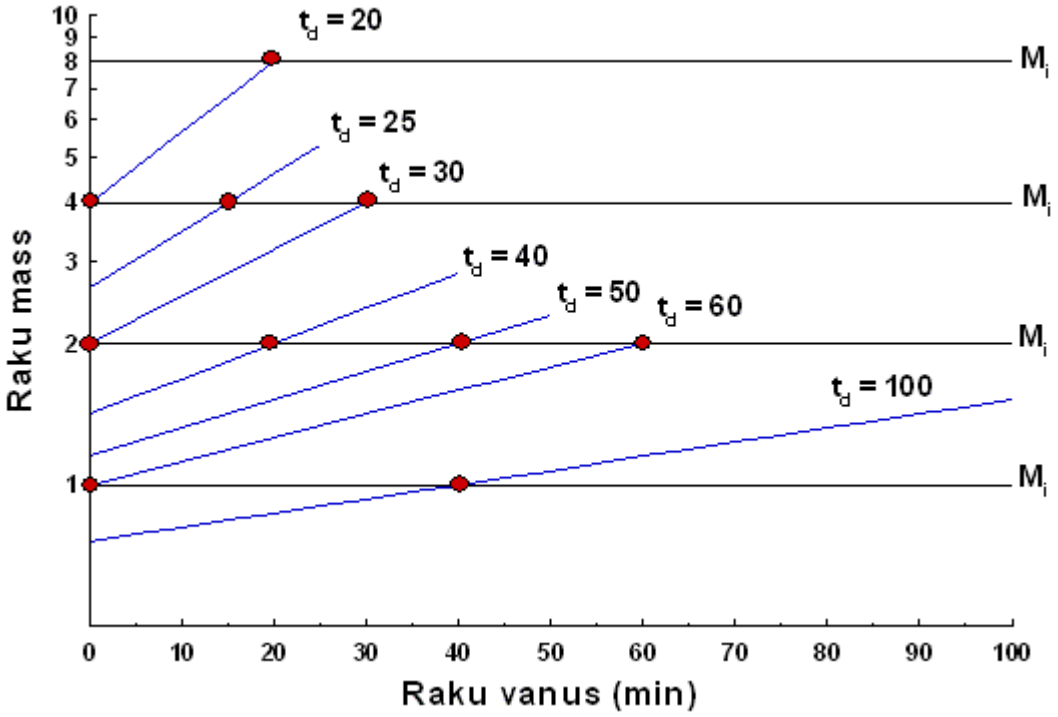
Joonis 23. DNA' de replikatsiooni poolsünkroonne initsiatsioon.
Jämedam joon on vabade DnaA molekulide, nõrgem DNA' de arv rakus.

Nagu bioloogilist protsessi kirjeldavas osas nimetatud, käivitub normaalsetel *E. Coli* bakteritel kiire kasvu korral DNA' de replikatsioon poolsünkroonselt ehk kõigil rakus asuvatel oriC' del toimub initsiatsioon peaaegu samaaegselt. Selle nähtuse toimumine loodud agentmudelis on näidatud joonisel 23. Seal toimuva seletus:

- 791. (simulatsiooni)sekundil ületab vabade DnaA molekulide arv rakus madala afiinsusega DnaA bokside täitumiseks vajaliku minimaalse taseme 40.
- 793. sekundil saavad ühe DNA oriC' l asuvad DnaA boksid täidetud ning seal toimub replikatsiooni initsiatsioon. See vabastab oriC piirkonnas olevad DnaA molekulid ja vaba DnaA hulk rakus tõuseb järsult. Selle DNA ja uue loodava DNA oriC inhibeeritakse, vältimaks võimalikku kohest reinitsiatsiooni. Samuti inhibeeritakse DnaA promootor, mistõttu DnaA valgu tootmine rakus aeglustub.
- Kõrge vaba DnaA kontsentratsioon kiirendab oluliselt teise, veidi "mahajäänud" DNA oriC piirkonna DnaA bokside täitumist.
- 794. sekundi teises pooles saavad ka selle teise DNA initsiaatorboksid täidetud ja toimub DNA replikatsiooni alustamine. Ka siin vabastatakse oriC piirkonnas olevad DnaA molekulid ja vaba DnaA hulk rakus tõuseb veelkord. Ka nende kahe DNA oriC ja DnaA promootor inhibeeritakse ning DnaA valgu tootmine rakus peatub.
- Uute DNA ahelate valmimise käigus tekib järjest juurde oriC piirkonnast väljaspool asuvaid tühje kõrge afiinsusega DnaA bokse, mis seovad rakus olevat vaba DnaA' d, nii et oriC' de inhibeeritusperioodi lõppedes on vaba DnaA tase langenud juba oluliselt alla läve 40, mis jällegi väldib kiiret reinitsiatsiooni.

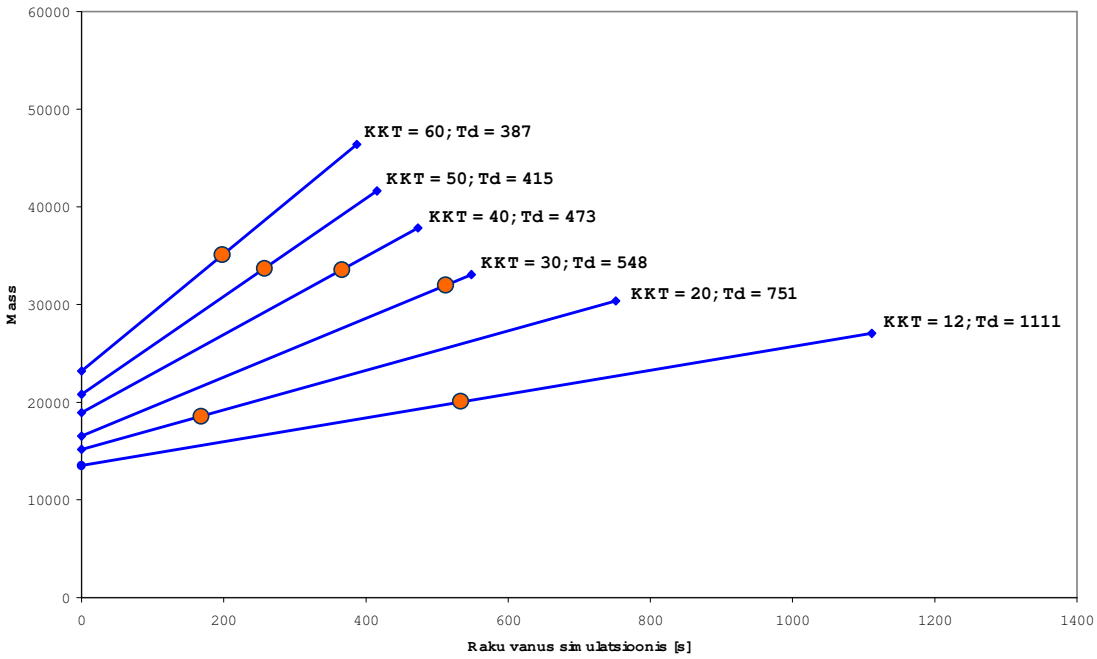
Kõige parem loodud agentmudeli adekvaatsuse näitaja oleks aga üldtunnustatud Cooper-Helmstetter-Donachie mudelis kirjeldatud initsiatsioonimassi ilmnemine. Nimelt, nagu varem selgitatud, on käesolev agentmudel valmistatud initsiaatori tiitrimise mudeli baasil, mis kirjeldab mikrotaseme käitumist, ja seetõttu ei ole agentmudelis initsiatsioonimassi kusagile ilmutatult sisse kirjutatud.

Joonisel 24 on selgitatud initsiatsioonimassi olemust vastavalt Donachie poolt väljapakutule. Nagu näha, varieerub raku vanus DNA replikatsiooni alustamise hetkel sõltuvalt raku kasvukiirusest, kuid raku mass initsiatsioonihetkel on konstantne teatud kasvukiiruste vahemikus. Raku mass on esitatud logaritmilises skaalas, kuna antud juhul on eeldatud, et raku mass suureneb rakutsükli jooksul eksponentsiaalselt. See aga ei tähenda, et agentmudelis kasutatud lineaarne kasv oleks väga ebareaalne. Nimelt ei ole bakteri raku täpset kasvufunktsiooni veel kindlaks määratud ja vähemalt esimeses lähenduses sobib selleks igati ka lineaarne kasv.



Joonis 24. Bakteri raku massi sõltuvus vanusest poollogaritmilises skaalas erinevatel kasvukiirustel (~1..3 poolestumist/h) Donachie järgi, kui raku mass kasvab rakutsükli kestel eksponentsiaalselt. Punased ringid tähistavad DNA replikatsiooni initsiatsiooni; t_d on rakutsükli pikkus. (Abner, 2003)

Joonis 25 on valmistatud agentsimulatsioonist kogutud info põhjal. Initsiatsioonimassi (ligikaudne) konstantsus teatud kasvukiiruste vahemikes tõepoolest ilmneb ja kasvukiiruse tõusul toimuv initsiatsioonimassi hüpe peaaegu kahekordistab selle massi analoogselt joonisel 24 tooduga. Seega võib loodud agentmudelit pidada õnnestunuks ning agendipõhine lähenemine bakteri raku simuleerimisele on täiesti võimalik (selles töös sai küll piirdutud peamiselt ühe protsessi modelleerimisega, kuid teised rakus toimuvad protsessid ei erine sellest niivõrd kardinaalselt, et muudaks agentorienteeritud lähenemise võimatuks).



Joonis 25. Raku massi sõltuvus vanusest erinevatel kasvukiirustel, vastavalt loodud agentmudelile. Raku mass kasvab rakutsükli kestel lineaarselt. Oranžid ringid tähistavad DNA replikatsiooni initsiatsiooni; KKT tähistab keskkonnatingimusi; Td on rakutsükli pikkus.

7.6. Kuidas tehtust edasi saab minna

Eelnevas sai demonstreeritud agentmudelite kasutatavus bakteri rakus toimuvate protsesside modelleerimiseks ja sellega käesoleva magistritöö eesmärk täidetud. Aga selleks, et luua reaalselt kasutatav agendipõhine rakkude modelleerimise vahend, tuleb veel kõvasti arendustööd teha. Järgnevas ongi välja pakutud mõned mõtted võimaliku edasise tegevuse kohta.

Agendiarenduskeskkonnaks sai valitud JADE, kuna oli teada, et temaga on võimalik sedalaadi agentsüsteeme valmistada (kuigi JADE on alles arendusjärgus keskkond ning ka loodud mudeli jooksutamisel tekkis vahepeal probleeme JADE' i töökindlusega). Praktilise modelleerimistööriista loomise puhul aga lihtsalt võimalikkusest ei piisa, arenduskeskkondadest tuleks välja valida kõige paremini sobiv.

Praeguse seisuga sõltub loodud simulatsioon reaalsest ajast, kuna kasutusel on taimerid. See toob aga endaga kaasa vähemalt ühe probleemi: kui kõiki taimereid käsitsi sobivalt ümber ei konfigureerita, siis väikese arvutusvõimsusega seadmetes simulatsioon ei tööta ning suure võimsusega arvutites ei kasutata kogu arvutusressurssi ära. Lisaks on karta sündmuste ajastamisega ning sõnumite värskusega seotud probleemide teket simulatsiooni hajutamisel üle mitme arvuti. Seetõttu tuleks ajakasutus võtta edaspidi põhjalikuma vaatluse alla ning leida sobivaim lahendus.

Bakteri täpsemal simuleerimisel tuleb kindlasti arvesse võtta ka rakus olevate objektide ruumiline asetus.

On tõenäoline, et esmalt modelleeritakse uue vahendiga kõige huvipakkuvamaid protsesse. Need aga ei pruugi olla koheselt tervikraku simulatsioonile lisatavad, kuna võivad vajada mõnede teiste, veel modelleerimata protsesside olemasolu simulatsioonis. Seetõttu oleks kasulik eelnevalt kindlaks määrata, millistele tingimustele peavad üksikprotsesside mudelid vastama, et neid saaks ka kunagi hiljem suurema vaevata üldmudelile lisada. Seejuures võiks uurida ka võimalust kasutada või vähemalt toetada laiemalt levinud bioloogiliste mudelite kirjeldamise keeli nagu SBML või CellML.

Üha keerukamaks muutuvate mudelite arusaadavuse säilitamisel on suur roll graafilistel kasutajaliidestel *(Graphical User Interface, GUI)*. Seega tuleks nii GUI' de loomine kui kasutamine teha võimalikult lihtsaks.

Kuna bakteri raku koostisosade arv on väga suur – ~10^9 molekuli, millest paljud on omavahel seotud, paljud osalevad mitmesugustes biokeemilistes protsessides jms. – siis oleks mõistlik lasta kasutajal vastavalt soovile ja olemasolevale arvutusvõimsusele muuta simulatsiooni abstraktsiooniastmeid. (Visuaalse) esituse abstraktsiooniaste määraks bakteri kujutamise detailsuse, võimaldades liigseid detaile peita või huvi korral neid täpsemalt välja tuua. Simulatsiooni enda abstraktsiooniastme muutmine tähendaks aga detailsete mudelite asendamist üldisematega (näiteks teatud tüüpi rakuplasmas ringi ujuvaid objekte esindavad agendid asendada ühe seda tüüpi objektide summaarset käitumist kirjeldava agendiga) või vastupidi, võimaldades reguleerida vajadust arvutusvõimsuse järele.

8. Kokkuvõte

Ühest küljest süsteemibioloogia areng ning teisalt agentmudelite populariseerumine põhjustavad suure tõenäosusega bioloogiliste organismide multiagentsüsteemidena modelleerimise laiema leviku (paljudest organismidest koosnevate populatsioonide agentsimulatsioonid on juba huviorbiiti tõusnud, kuid seal piirdutakse enamasti ühe agendiga indiviidi kohta).

Käesolev töö näitas, et bakteri rakus toimuvat on tõepoolest võimalik esitada multiagentsüsteemina ning saada simulatsioonist adekvaatseid tulemusi, kusjuures modelleerija ja kasutaja arusaama agentmudeli struktuurist ja vähemalt osaliselt ka käitumisest lihtsustab "loomulikkus" – mudeli elemente saab panna otseselt vastavusse bioloogilise süsteemi objektidega. Seega tuleks agentorienteeritud lähenemisviisiga raku modelleerimisele põhjalikumalt edasi töötada ning proovida luua bioloogide-biokeemikute jaoks praktiliselt kasutatavaid agendipõhiseid raku modelleerimise-simuleerimise vahendeid.

9. Viited

Internetist pärinevaid materjale on kasutatud sellistena, nagu nad olid kättesaadavad ajavahemikus 2004. a. lõpp – 2005. a. esimesed kuud (välja arvatud juhul, kui viite juurde on märgitud teisiti).

Abner, K. 2003, *Prokarüootse rakutsükli üherakumudel*, magistritöö, Tallinna Tehnikaülikool.

Alur, R., Belta, C., Kumar, V., Mintz, M., Pappas, G. J., Rubin, H. and Schug, J. 2002, "Modeling And Analyzing Biomolecular networks", *Computing in Science & Engineering*, vol. 4, no. 1, pp. 20-31, database: IEEE Xplore.

Anthes, G. H. 2003, "Agents of Change", *Computerworld*, ~4p, http://www.computerworld.com/printthis/2003/0,4814,77855,00.html

Bailer-Jones, D. M. 2002, "Scientists' Thoughts on Scientific Models", *Perspectives on Science*, vol. 10, no. 3, pp. 275-301, EBSCOhost database: Academic Search Premier.

Bellifemine, F., Caire, G., Trucco, T. and Rimassa, G. *Jade Programmer's Guide*, 52p, http://jade.tilab.com/doc/programmersguide.pdf

Burbeck, S. and Jordan, K. 2004, *An assessment of systems biology*, *IBM Life Sciences*, 32p, http://www-1.ibm.com/industries/healthcare/doc/content/bin/AssessmentofSys.pdf

Burleigh, I., Suen, G. and Jacob, C. 2003, "DNA in Action! A 3D Swarm-based Model of a Gene Regulatory System", *Proceedings of the First Australian Conference on Artificial Life. Lecture Notes in Computer Science.*, Springer-Verlag: Berlin, 15p. http://pages.cpsc.ucalgary.ca/~jacob/ESD/Research/LacOperon/3D/DNA-Action.pdf

Caire, G. *Jade Tutorial - Jade Programming For Beginners*, 22p, http://jade.tilab.com/doc/JADEProgramming-Tutorial-for-beginners.pdf

Chanchalani, D. 2003, *Agent Oriented Programming (student presentation)*, 21 slides, http://www.cse.sc.edu/~fredd/Fall03/CSCE740/individual/chanchalani.pdf

da Costa, N. and French, S. 2000, "Models, Theories, and Structures: Thirty Years On", *Philosophy of Science*, vol. 67, no. 3, pp. S116-S127, EBSCOhost database: Academic Search Premier.

Eikenberry, J. *GNU/Linux AI & Alife HOWTO: Agents*, ~14p, http://zhar.net/gnu-linux/howto/html/ai-6.html

Ferber, J. 1999, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman, Harlow, 528p.

Fisher, M. J., Malcolm, G. and Paton, R. C. 2000, "Spatio-logical processes in intracellular signalling", *Biosystems*, vol. 55, no. 1-3, pp. 83-92, database: ScienceDirect.

Fisher, M. J., Paton, R. C. and Matsuno, K. 1999, "Intracellular signalling proteins as 'smart' agents in parallel distributed processes", *Biosystems*, vol. 50, no. 3, pp. 159-171, database: ScienceDirect.

Garner, H. R. and Pertsemlidis, A. 2003, "Computational biology: biological insight from 1s and 0s", *BIOSILICO*, vol. 1, no. 1, pp. 27-35, database: ScienceDirect.

Goldin, D. and Wegner, P. 2004, "The Origins of the Turing Thesis Myth", *Brown Technical Report CS 04-13*, 9p, http://www.cse.uconn.edu/~dqg/papers/myth.pdf

Gombert, A. K. and Nielsen, J. 2000, "Mathematical modelling of metabolism", *Current Opinion in Biotechnology*, vol. 11, no. 2, pp. 180-186, database: ScienceDirect.

Gonzalez, P. P., Cardenas, M., Camacho, D., Franyuti, A., Rosas, O. and Lagunez-Otero, J. 2003, "Cellulat: an agent-based intracellular signalling model", *Biosystems*, vol. 68, no. 2-3, pp. 171-185, database: ScienceDirect.

Grand, S. 1997, "Three observations that changed my life", *IEEE Expert*, vol. 12, no. 6, pp. 13-17, database: IEEE Xplore. // Artikkel on kättesaadav ka aadressilt: http://fp.cyberlifersrch.plus.com/articles/ieee_2.htm

Green, W. 1995, "The Man From CHAOS", *Fast Company*, no. 1, ~2p, http://www.fastcompany.com/magazine/01/chaos.html

Gwinn, T. "The Rosen Modeling Relation", ~8p, http://www.panmere.com/rosen/faq_mr1.htm

Herrick, J., Kohiyama, M., Atlung, T. and Hansen, F. G. 1996, "The initiation mess?", *Molecular Microbiology*, vol. 19, no. 4, pp. 659-666, database: Blackwell Synergy.

Hertzfeld, A. "Bicycle", ~2p, http://www.folklore.org/StoryView.py?project=Macintosh&story=Bicycle.txt&sortOrder=Sort%20by%20Date&detail=medium

Jacob, E. B., Becker, I., Shapira, Y. and Levine, H. 2004, "Bacterial linguistic communication and social intelligence", *Trends in Microbiology*, vol. 12, no. 8, pp. 366-372, database: ScienceDirect.

Jennings, N. R. and Wooldridge, M. 1998, "Applications of Intelligent Agents", in Jennings, N. R.and Wooldridge, M. (eds) *Agent Technology: Foundations, Applications, and Markets*, Springer-Verlag, pp. 3-27, http://www.csc.liv.ac.uk/~mjw/pubs/applications.pdf

Katare, S. and Venkatasubramanian, V. 2001, "An agent-based learning framework for modeling microbial growth", *Engineering Applications of Artificial Intelligence*, vol. 14, no. 6, pp. 715-726, database: ScienceDirect.

Kell, D. B. 2004, "Metabolomics and systems biology: making sense of the soup", *Current Opinion in Microbiology*, vol. 7, no. 3, pp. 296-307, database: ScienceDirect.

Khan, S., Makkena, R., McGeary, F., Decker, K., Gillis, W. and Schmidt, C. 2003, "A MultiAgent System for the Quantitative Simulation of Biological Networks", *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 385-392, database: ACM Digital Library.

Livelock from FOLDOC, ~1p, http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?livelock

Mikulecky, D. C. "A Close Look at a New Science: Chaos as Science or Science in Chaos?", ~11p, http://www.people.vcu.edu/~mikuleck/chaos2.htm

Milner, R. and Stepney, S. 2003, "Nanotechnology: Computer Science opportunities and challenges", *Submission by the UK Computing Research Committee to the Nanotechnology Working Group of the Royal Society and the Royal Academy of Engineering*, ~5p, http://www.nanotec.org.uk/evidence/92aUKCRC.htm

Morris, R. W., Bean, C. A., Farber, G. K., Gallahan, D., Jakobsson, E., Liu, Y., Lyster, P. M., Peng, G. C. Y., Roberts, F. S., Twery, M., Whitmarsh, J. and Skinner, K. 2005, "Digital biology: an emerging and promising discipline", *Trends in Biotechnology*, vol. 23, no. 3, pp. 113-117, database: ScienceDirect.

Parunak, H. V. D. 1998, "Practical and Industrial Applications of Agent-Based Systems", 41p, http://www.erim.org/~vparunak/apps98.pdf

Paton, R. C. 1993, "Some Computational Models at the Cellular Level", *Biosystems*, vol. 29, no. 2-3, pp. 63-75, http://citeseer.ist.psu.edu/paton93some.html

Peck, S. L. 2004, "Simulation as experiment: a philosophical reassessment for biological modeling", *Trends in Ecology & Evolution*, vol. 19, no. 10, pp. 530-534, database: ScienceDirect.

Querrec, G., Rodin, V., Abgrall, J. F., Kerdelo, S. and Tisseau, J. 2003, "Uses of Multiagents Systems for simulation of MAPK pathway", *Third IEEE Symposium on Bioinformatics and Bioengineering, Proceedings*, pp. 421-425, database: IEEE Xplore.

Russell, S. and Norvig, P. 1995, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc. (Part I, chapter 2, pp. 31-52: http://www-ee.eng.hawaii.edu/~nreed/agent693/papers/RussellNorvigch02.pdf)

Seibel, F. and Kellam, L. 2003, "The Virtual World of Agent-Based Modeling: Procter & Gamble's Dynamic Supply Chain", *Cap Gemini Ernst & Young' s Focus E-Zine Technology*, no. 15, 6p, http://community.capgemini.com/focus/fcd/Technology15/en_index.jsp

Sorger, P. K. 2005, "A reductionist's systems biology: Opinion", *Current Opinion in Cell Biology*, vol. 17, no. 1, pp. 9-11, database: ScienceDirect.

Stepney, S., Braunstein, S. L., Clark J. A., Tyrrell, A., Adamatzky, A., Smith, R. E., Addis, T., Johnson, C., Timmis, J., Welch, P., Milner, R., Partridge, D. 2004, "Journeys in Non-Classical Computation. A Grand Challenge for Computing Research.", *Grand Challenges for Computing Research Draft Proposal, UK Computing Research Committee*, 30p, http://www.nesc.ac.uk/esi/events/Grand_Challenges/proposals/stepney.pdf

Stroustrup, B. 1988, "What is Object-Oriented Programming?", *IEEE Software*, vol. 5, no. 3, pp. 10-20, database: IEEE Xplore.

Tesfatsion, L. *General Software and Toolkits. Agent-Based Computational Economics (ACE) and Complex Adaptive Systems (CAS)*, ~15p, http://www.econ.iastate.edu/tesfatsi/acecode.htm

The Snake Robot, http://www.meganic.dk/old/snakerobot.html (vaadatud detsembris 2003, praegu pole enam kättesaadav)

Vidal, J. M. *MultiAgent systems: Software*, ~1p, http://www.multiagent.com/Software/index.html

Webb, K. and White, T. Article In Press, Corrected Proof, "UML as a cell and biochemistry modeling language", *Biosystems*, 20p, database: ScienceDirect.

Wegner, P. 1999, "Towards Empirical Computer Science", *The Monist*, vol. 82, no. 1 *(Issue on the Philosophy of Computation)*, 27p, http://citeseer.ist.psu.edu/156429.html

Wolkenhauer, O. 2002, "Simulating what cannot be simulated", *Dagstuhl Position Statement*, 9p, http://www.systemsbiology.umist.ac.uk/docs/dagstuhl_6_2002.pdf

Wolkenhauer, O. and Klingmüller, U. 2004, "Systems Biology: From a Buzzword to a Life Sciences Approach", *BioForum Europe*, no. 04/2004, pp. 22-23, http://www.systemsbiology.umist.ac.uk/docs/p_bioforum_aug_04.pdf

Wooldridge, M. 1998, "Agents and Software Engineering", *AI*IA Notizie*, vol. 3, pp. 31-37, http://www.csc.liv.ac.uk/~mjw/pubs/aiia98.pdf

Wooldridge, M. and Jennings, N. R. 1998, "Pitfalls of Agent-Oriented Development", in Sycara, K. P. and Wooldridge, M. (eds) *Agents ' 98: Proceedings of the Second International Conference on Autonomous Agents,* ACM Press, 7p, http://www.csc.liv.ac.uk/~mjw/pubs/agents98.pdf

10. Lisa: lähtekood

.java failidest on genereeritud kujundatud .html failid *"Colorer take5"* abil, mis on kättesaadav aadressilt http://colorer.sourceforge.net/

AgEnvironment

(2 faili)

```
// File: AgEnvironment.java
// Description: The environment where bacteria live.


package bactmodel;

import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAException;

// ======
// CLASS |
// ======
public class AgEnvironment extends Agent {

    // All conditions represented by a single number.
    // The higher, the better.
    int conditions = 50;

    // ----------------
    // FUNCTION: setup |
    // ----------------
    protected void setup() {
        // System.out.println("\nEntering " + getAID().getName() + " setup()");

        // Adding main behaviours
        addBehaviour(new BehEnvConditionServer(this));

        // Registering in DF
        addBehaviour(new BehAddServiceToDF(this,
                                           "environment-service",
                                           "environmentinfo"));

        // System.out.println("\nNew agent " + getAID().getName()
        //                    + " constructed");
    } //endof FUNCTION: setup

    // -------------------
    // FUNCTION: takeDown |
    // -------------------
    protected void takeDown() {
        // System.out.println("\nEntering " + getAID().getName()
        //                                   + " takeDown()");
        // Trying to deregister from DF
        try {
            DFService.deregister(this);
        } catch (FIPAException fe) {
              fe.printStackTrace();
          }

        // System.out.println("\nLeaving " + getAID().getName()
        //                                  + " takeDown()");
    } //endof FUNCTION: takeDown

} //endof CLASS
```

```
// File: BehEnvConditionServer.java
// Description: Gives information about the environment conditions
//     and allows to set a new value for the conditions.
// Usage: query-ref "Conditions?"
//            => inform "integer"
//        request ConversationId:"Set conditions" "integer"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import java.util.regex.*;

// ======
// CLASS |
// ======
public class BehEnvConditionServer extends CyclicBehaviour {

    AgEnvironment myAgent = null;
    MessageTemplate msgTemplAsk = null;
    MessageTemplate msgTemplChng = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehEnvConditionServer(AgEnvironment a) {
        super(a);
        myAgent = a;

        // Constructing templates for recognizing relevant messages.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                            ACLMessage.QUERY_REF);
        MessageTemplate msgT2 = MessageTemplate.MatchContent("Conditions?");
        msgTemplAsk = MessageTemplate.and(msgT1, msgT2);

        msgT1 = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        msgT2 = MessageTemplate.MatchConversationId("Set conditions");
        msgTemplChng = MessageTemplate.and(msgT1, msgT2);

        // System.out.println("\nBehEnvConditionServer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        // Answering queries about current conditions.
        ACLMessage msg = myAgent.receive(msgTemplAsk);

        if(msg != null) {
            nothingfound = false;

            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(
                Integer.toString(myAgent.conditions));
            myAgent.send(reply);
            //System.out.println("Sent a reply: " + reply.getContent());
        }
```

```
        // Processing the request to change conditions.
        msg = myAgent.receive(msgTemplChng);
        if(msg != null) {
            nothingfound = false;
            String content = msg.getContent();

            try {
                myAgent.conditions = Integer.parseInt(content);
            } catch (NumberFormatException e) {
                  // System.out.println("ERROR: "
                  //                + "Can't change conditions to noninteger");
                  e.printStackTrace();
              }
            // System.out.println("Conditions: " + myAgent.conditions);
        }

        if(nothingfound)
            block();
    } //endof FUNCTION: action

} //endof CLASS
```

AgBacterium

(11 faili)

```
// File: AgBacterium.java
// Description: Represents the whole bacterium. Manages the agents
//     that represent parts of the bacterium (e.g. DNA). Simulates
//     the overall behaviour of all other parts that are not explicitly
//     represented as agents.
// NB! Currently creating a new AgBacterium creates a totally new cell
//     with one DNA & stuff like that. Agent cloning is used for division.
// Usage: If first argument is "singlecell", then executes in single-cell-only
//        mode, terminating all other cells that appear after divisions.
// Warning: Currently only "singlecell" model is tested!

package bactmodel;

import jade.core.AID;
import jade.core.behaviours.Behaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.wrapper.StaleProxyException;

import org._3pq.jgrapht.graph.SimpleDirectedGraph;

import jade.gui.GuiAgent;
import jade.gui.GuiEvent;

import java.io.File;

// ======
// CLASS |
// ======
public class AgBacterium extends GuiAgent {

    int freeDnaA = 0;
    int cellMass = 10000;
    // Will contain full names of DNAs (Strings).
    SimpleDirectedGraph dgMyDNAs = new SimpleDirectedGraph();
    // Number of active DnaA promoters.
    int methylatedDnaAPromos = 0;
    // AID of the environment agent.
    AID envAID = null;

    boolean justCloned = false;

    ACLMessage msgGetCondForGrowth = null;
    ACLMessage msgGetMet = null;

    // Update cell mass every x ms.
    int updPeriod = 100;

    // If true, then after division only parent cell survives
    // (therefore we can examine the one cell without wasting computing
    // resources to creating the whole colony).
    boolean singleCell = false;

    // This variable is used in takedown().
    String nameOfTheVeryFirstCell = null;

    // AgBacterium's graphical user interface.
    transient protected GuiBact gui;

    // Non-astronomical time for drawing graphs on GUI.
    int time = 0;
```

```
// For remembering the concrete GUI behaviour, because it must be
// removed in the clone.
Behaviour behGui = null;

// ----------------
// FUNCTION: setup |
// ----------------
protected void setup() {
    // System.out.println("\nEntering " + getAID().getName() + " setup()");

    // First of all, trying to find the environment agent.
    // If found, then activating cell growth, otherwise
    // terminating this agent (can't live without environment).
    hServiceFinder envF = new hServiceFinder(this,
                                             "environment-service",
                                             "environmentinfo");

    int nFound = envF.agentsWithService.length;

    if(nFound < 1) {
        // System.out.println("\nERROR: No environment agents found. "
        //                    + "Will terminate this AgBacterium.");
        this.doDelete();
    } else {
          /* if(1 == nFound) {
              System.out.println("\nSuccess: one environment agent found:");
          } else System.out.println("\nFound more than one environment "
                                    + "agent. Will use one of them:");
          */

          envAID = envF.agentsWithService[0];
          // System.out.println("   " + envAID.getName());
          addBehaviour(new BehBactGrowth(this, updPeriod));
      }

    nameOfTheVeryFirstCell = getName();

    // Adding other behaviours
    addBehaviour(new BehBactDnaAServer(this));
    addBehaviour(new BehBactDivOrganizer(this));
    addBehaviour(new BehBactInfoserver(this));

    // Starting DNA management
    addBehaviour(new BehBactDnaReg(this));

    // Creating one DNA.
    String newName = "dna" + System.currentTimeMillis();
    String[] argsDNA = {getName(), "null", "0"};
    try {
        getContainerController()
          .createNewAgent(newName, "bactmodel.AgDNA", argsDNA)
          .start();
    } catch(StaleProxyException e) {
          e.printStackTrace();
      }

    // Creating one DnaAFactory.
    newName = "dnaAFactory" + System.currentTimeMillis();
    String[] argsDnaF = {getName(), envAID.getName()};
    try {
        getContainerController()
          .createNewAgent(newName, "bactmodel.AgDnaAFactory", argsDnaF)
          .start();
    } catch(StaleProxyException e) {
          e.printStackTrace();
```

```
      }

    // Starting DnaA promoter activity poller.
    addBehaviour(new BehBactDnaAPromPoll(this, updPeriod));

    // Using command line arguments, if any.
    Object[] args = getArguments();
    if (args != null && args.length > 0) {
        String firstArg = (String) args[0];
        // System.out.println("\nFirst argument is " + firstArg);
        if(firstArg.equals("singlecell")) {
            singleCell = true;
            // Deleting the old logfile.
            File logfile = new File("_log.txt");
            logfile.delete();
            // System.out.println("\nSINGLE CELL ONLY mode ");
        }
    }

    // Registering in DF
    addBehaviour(new BehAddServiceToDF(this,
                                       "bacterium-service",
                                       "bacterium"));

    // Creating and showing the GUI
    gui = new GuiBact(this);
    gui.setVisible(true);
    behGui = new BehBactGui(this, updPeriod);
    addBehaviour(behGui);

    // System.out.println("\nLeaving " + getAID().getName() + " setup()");
} //endof FUNCTION: setup

// --------------------
// FUNCTION: takeDown |
// --------------------
protected void takeDown() {
    // System.out.println("\nEntering " + getAID().getName()
    //                                   + " takeDown()");

    if(!singleCell || getName().equals(nameOfTheVeryFirstCell)) {
        // Trying to deregister from DF
        try {
            DFService.deregister(this);
        } catch (FIPAException fe) {
              fe.printStackTrace();
          }
    }

    // Disposing the GUI
    if (gui!=null) {
      gui.dispose();
      gui.setVisible(false);
    }

    // System.out.println("\nLeaving " + getAID().getName()
    //                                  + " takeDown()");
} //endof FUNCTION: takeDown

// ----------------------
// FUNCTION: beforeClone |
// ----------------------
protected void beforeClone() {
    // System.out.println("\nEntering " + getAID().getName()
    //                                   + " beforeClone()");
    justCloned = true;
```

```
        // System.out.println("Leaving " + getAID().getName()
        //                                + " beforeClone()");
    } //endof FUNCTION: beforeClone

    // ---------------------
    // FUNCTION: afterClone |
    // ---------------------
    protected void afterClone() {
        // System.out.println("\nEntering " + getAID().getName()
        //                                  + " afterClone()");

        // Changing the sender of info requests to this agent.
        msgGetCondForGrowth.setSender(getAID());
        msgGetMet.setSender(getAID());

        if(!singleCell) {
            // Registering in DF
            addBehaviour(new BehAddServiceToDF(this,
                                               "bacterium-service",
                                               "bacterium"));

            // Creating one DnaAFactory.
            String newName = "dnaAFactory" + System.currentTimeMillis();
            String[] argsDnaF = {getName(), envAID.getName()};
            try {
                getContainerController()
                  .createNewAgent(newName, "bactmodel.AgDnaAFactory", argsDnaF)
                  .start();
            } catch(StaleProxyException e) {
                  e.printStackTrace();
              }
        }

        // Removing GUI behaviour, because currently the clone has no GUI.
        removeBehaviour(behGui);

        // System.out.println("\nLeaving " + getAID().getName()
        //                                 + " afterClone()");
    } //endof FUNCTION: afterClone

    // ------------------
    // FUNCTION: display |
    // ------------------
    // Gives new info to GUI
    void display() {
        gui.displayCellMass(time, cellMass);
        gui.displayFreeDnaA(time, freeDnaA);
        gui.displayNumOfDNAs(time, dgMyDNAs.vertexSet().size());
        gui.displayMethylatedDnaAPromos(time, methylatedDnaAPromos);
    } //endof FUNCTION: display

    // ---------------------
    // FUNCTION: onGuiEvent |
    // ---------------------
    // Currently non-interactive gui
    protected void onGuiEvent(GuiEvent ev) {}
    //endof FUNCTION: onGuiEvent

} //endof CLASS
```

```java
// File: BehBactDivider.java
// Description: Divides the bacterium in two.

package bactmodel;

import java.util.*;

import jade.core.AID;
import jade.core.behaviours.Behaviour;
import jade.core.ContainerID;
import jade.lang.acl.ACLMessage;
import jade.wrapper.ControllerException;

import org._3pq.jgrapht.graph.SimpleDirectedGraph;
import org._3pq.jgrapht.alg.ConnectivityInspector;

// ======
// CLASS |
// ======
public class BehBactDivider extends Behaviour {

    AgBacterium myAgent = null;
    int step = 0;
    ArrayList lConDnaSets = new ArrayList();
    String cloneName = null;
    boolean cloneFlagNotRaised = true;
    boolean done = false;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactDivider(AgBacterium a) {
        super(a);
        myAgent = a;
        // System.out.println("\nBehBactDivider Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        if(cloneFlagNotRaised) {
            // Everything here will be executed before cloning, because
            // doClone() only raises a flag and doesn't perform cloning
            // immediately.
            // System.out.println("\n" + myAgent.getName() + " before div:");
            // System.out.println("  cellMass " + myAgent.cellMass);
            // System.out.println("  freeDnaA " + myAgent.freeDnaA);
            // System.out.println("  dgMyDNAs " + myAgent.dgMyDNAs.toString());

            ConnectivityInspector c
                = new ConnectivityInspector(myAgent.dgMyDNAs);
            lConDnaSets.addAll(c.connectedSets());

            cloneName = "b" + System.currentTimeMillis();
            ContainerID cID = new ContainerID();
            try {
                cID.setName(myAgent
                            .getContainerController().getContainerName());
                myAgent.doClone(cID, cloneName);
                cloneFlagNotRaised = false;
            } catch (ControllerException e) {
                  e.printStackTrace();
              }
```

```
}

if(myAgent.justCloned) {
    // This will be executed after cloning, in both parent and clone,
    // to remove the resources that should have been left to the other
    // cell.

    // Different actions for parent and clone:
    if(cloneName.equals(myAgent.getLocalName())) {
        // This will be executed in clone.

        // Removing the DNA that stayed in the parent cell.
        int s = lConDnaSets.size();
        for(int i = 0; i < s/2; i++) {
            Iterator iter = ((Set)lConDnaSets.get(i)).iterator();
            while(iter.hasNext())
                myAgent.dgMyDNAs.removeVertex(iter.next());
        }

        // Changing the owner info of the acquired DNA.
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.setConversationId("Set new owner");

        for(int i = s/2; i < s; i++) {
            Iterator iter = ((Set)lConDnaSets.get(i)).iterator();
            while(iter.hasNext()) {
                String rname = (String)iter.next();
                AID receiverAID = new AID(rname, AID.ISGUID);
                msg.addReceiver(receiverAID);
            }
        }

        try {
            myAgent.send(msg);
        } catch(Exception e) {
              e.printStackTrace();
          }

        // Different actions depending on the mode (single-cell
        // or colony)
        if(myAgent.singleCell) {
            // In single-cell-mode clone must be terminated.

            // Requesting the DNA to terminate.
            // If, however, these messages happen to arrive _before_
            // those owner-change messages, then DNAs will try to
            // deregister themselves from the parent cell, not clone,
            // and that cell will notify about errors (but no real
            // errors, only notifications).
            msg = new ACLMessage(ACLMessage.REQUEST);
            msg.setConversationId("Kill");

            for(int i = s/2; i < s; i++) {
                Iterator iter = ((Set)lConDnaSets.get(i)).iterator();
                while(iter.hasNext()) {
                    String rname = (String)iter.next();
                    AID receiverAID = new AID(rname, AID.ISGUID);
                    msg.addReceiver(receiverAID);
                }
            }

            try {
                myAgent.send(msg);
            } catch(Exception e) {
                  e.printStackTrace();
              }
```

```
                    // System.out.println("\nClone " + myAgent.getName()
                    //                                 + " will terminate");
                    myAgent.doDelete();
                    done = true;

                } else {
                      // In normal mode clone must be trimmed to be the result
                      // of _division_ , not cloning.

                      myAgent.freeDnaA /= 2;
                      myAgent.cellMass /= 2;

                      // System.out.println("\nClone " + myAgent.getName()
                      //                                 + " after div:");
                      // System.out.println("  cellMass " + myAgent.cellMass);
                      // System.out.println("  freeDnaA " + myAgent.freeDnaA);
                      // System.out.println("  dgMyDNAs " + myAgent.dgMyDNAs
                      //                                          .toString());

                      done = true;
                }
            } else {
                  // This will be executed in parent.

                  // Remainder (% is the modulus operation) is added to avoid
                  // the loss of material.
                  myAgent.cellMass = myAgent.cellMass/2 + myAgent.cellMass % 2;
                  myAgent.freeDnaA = myAgent.freeDnaA/2 + myAgent.freeDnaA % 2;

                  // Removing the DNA that went to the new daughter cell
                  // (however, the new cell must change owner info in those
                  // DNA agents).
                  int s = lConDnaSets.size();
                  for(int i = s/2; i < s; i++) {
                      Iterator iter = ((Set)lConDnaSets.get(i)).iterator();
                      while(iter.hasNext())
                          myAgent.dgMyDNAs.removeVertex(iter.next());
                  }

                  // System.out.println("\nParent " + myAgent.getName()
                  //                                 + " after div:");
                  // System.out.println("  cellMass " + myAgent.cellMass);
                  // System.out.println("  freeDnaA " + myAgent.freeDnaA);
                  // System.out.println("  dgMyDNAs " + myAgent.dgMyDNAs
                  //                                          .toString());
                  done = true;
              }

              myAgent.justCloned = false;

          }

    } //endof FUNCTION: action

    // -----------------
    // FUNCTION: done |
    // -----------------
    public boolean done() {
        return done;
    } //endof FUNCTION: done

} //endof CLASS
```

```
// File: BehBactDivOrganizer.java
// Description: Organizes cell division.
// Usage: inform "DNA replication completed"
//        inform "Postreplication delay period ended"

package bactmodel;

import java.util.*;

import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import org._3pq.jgrapht.Edge;
import org._3pq.jgrapht.graph.SimpleDirectedGraph;

// ======
// CLASS |
// ======
public class BehBactDivOrganizer extends CyclicBehaviour {

    AgBacterium myAgent = null;
    MessageTemplate msgTemplDna = null;
    MessageTemplate msgTemplWaker = null;

    // Delay (in ms) from DNA replication to cell division.
    int postRepDelay = 20000;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactDivOrganizer(AgBacterium a) {
        super(a);
        myAgent = a;

        // Constructing a template for recognizing the message from DNA.
        MessageTemplate msgT1= MessageTemplate.MatchPerformative(
                                    ACLMessage.INFORM);
        MessageTemplate msgT2= MessageTemplate.MatchContent(
                                    "DNA replication completed");
        msgTemplDna = MessageTemplate.and(msgT1, msgT2);

        // Constructing a template for recognizing the message from Waker.
        msgT2= MessageTemplate.MatchContent(
                                    "Postreplication delay period ended");
        msgTemplWaker = MessageTemplate.and(msgT1, msgT2);

        // System.out.println("\nBehBactDivOrganizer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        // If DNA replication completed, then update graph dgMyDNAs
        // and start counting down the time to cell division.
        ACLMessage msg = myAgent.receive(msgTemplDna);
        if(msg != null) {
            nothingfound = false;
            // System.out.println("Got msg from DNA. "
            //                    + "Will update dgMyDNAs and create waker");
```

```
                String dnaName = msg.getSender().getName();
                if(1 == myAgent.dgMyDNAs.outDegreeOf(dnaName)) {
                    ArrayList outEdges = new ArrayList(myAgent.dgMyDNAs
                                                       .outgoingEdgesOf(dnaName));
                    myAgent.dgMyDNAs.removeEdge((Edge)outEdges.get(0));
                } else {
                     // System.out.println("\nERROR in " + myAgent.getName()
                     //          + "\n  dgMyDNAs seems to contain incorrect info");
                  }

                myAgent.addBehaviour(new BehBactDivWaker(myAgent, postRepDelay));
            }

            // If delay ended, perform cell division.
            msg = myAgent.receive(msgTemplWaker);
            if(msg != null) {
                nothingfound = false;
                myAgent.addBehaviour(new BehBactDivider(myAgent));
            }

            if(nothingfound)
                block();

        } //endof FUNCTION: action

    } //endof CLASS
```

```
// File: BehBactDivWaker
// Description: Notifies BehBactDivOrganizer, when postreplication time delay
//              has ended and it is OK to perform cell division.

package bactmodel;

import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.core.behaviours.WakerBehaviour;

// ======
// CLASS |
// ======
public class BehBactDivWaker extends WakerBehaviour {

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactDivWaker(AgBacterium a, long timeout) {
        super(a, timeout);
        // System.out.println("\nBehBactDivWaker Constructed");
    } //endof FUNCTION: Constructor

    // ------------------------------
    // FUNCTION: handleElapsedTimeout |
    // ------------------------------
    protected void handleElapsedTimeout() {

        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(myAgent.getAID());
        msg.setContent("Postreplication delay period ended");
        try {
            myAgent.send(msg);
        } catch(Exception e) {
              // System.out.println("\nERROR: Sending threw an exception.");
          }

    } //endof FUNCTION: handleElapsedTimeout

} //endof CLASS
```

```
// File: BehBactDnaAPromPoll.java
// Description: Polls DNA's to calculate total number of active
//              DnaA promoters in the cell.
// NB! Currently it just considers all incoming messages as new info,
//     does NOT do any accounting about from whom the message was etc.
//     (Quite a high danger of getting the wrong number of promoters!)

package bactmodel;

import java.util.*;

import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.core.behaviours.TickerBehaviour;

// ======
// CLASS |
// ======
public class BehBactDnaAPromPoll extends TickerBehaviour {

    AgBacterium myAgent = null;
    MessageTemplate msgTemplMet = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactDnaAPromPoll(AgBacterium a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;

        // Constructing a message for asking if methylated or not.
        String convID = "general DNA info";
        myAgent.msgGetMet = new ACLMessage(ACLMessage.QUERY_REF);
        myAgent.msgGetMet.setConversationId(convID);
        myAgent.msgGetMet.setContent("dnaAPromoterMethylated");

        // Constructing a template for recognizing the reply.
        msgTemplMet = MessageTemplate.MatchConversationId(convID);

        // System.out.println("\nBehBactDnaAPromPoll Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        // Using all answers that have arrived since last tick.
        int numMet = 0;
        ACLMessage msg = null;
        while(null != (msg = myAgent.receive(msgTemplMet))) {
            String content = msg.getContent();
            if(content.equals("true"))
                numMet++;
        }
        myAgent.methylatedDnaAPromos = numMet;

        // Refreshing the receivers list.
        myAgent.msgGetMet.clearAllReceiver();

        ArrayList lDnaNames = new ArrayList();
        lDnaNames.addAll(myAgent.dgMyDNAs.vertexSet());
        int s = lDnaNames.size();
```

```
            AID rec = null;
            for(int i = 0; i < s; i++) {
                rec = new AID((String)lDnaNames.get(i), AID.ISGUID);
                myAgent.msgGetMet.addReceiver(rec);
            }

            // Sending the question again.
            try {
                myAgent.send(myAgent.msgGetMet);
            } catch(Exception e) {
                  e.printStackTrace();
              }

        } //endof FUNCTION: onTick

    } //endof CLASS
```

```
// File: BehBactDnaAServer.java
// Description: Stores incoming DnaA in bacterium.
//              Gives out DnaA when asked.
// Usage: request :ConversationId:"Store DnaA" "integer"
//        request :ConversationId:"Get DnaA"
//            => inform :ConversationId:"Get DnaA" "integer"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehBactDnaAServer extends CyclicBehaviour {

    AgBacterium myAgent = null;
    MessageTemplate msgTemplStore = null;
    MessageTemplate msgTemplGet = null;

    // How much DnaA per request is given out (it is a variable,
    // not constant!).
    int amountDnaA = 0;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactDnaAServer(AgBacterium a) {
        super(a);
        myAgent = a;

        // Constructing a template for recognizing incoming DnaA.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                        ACLMessage.REQUEST);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                        "Store DnaA");
        msgTemplStore = MessageTemplate.and(msgT1,msgT2);

        // Constructing a template for recognizing requests to get DnaA.
        msgT2 = MessageTemplate.MatchConversationId("Get DnaA");
        msgTemplGet = MessageTemplate.and(msgT1,msgT2);

        // System.out.println("\nBehBactDnaAServer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        ACLMessage msg = myAgent.receive(msgTemplStore);

        if(msg != null) {
            nothingfound = false;
            String content = msg.getContent();

            try {
                myAgent.freeDnaA += Integer.parseInt(content);
            } catch (NumberFormatException e) {
                  // System.out.println("ERROR: "
```

```
                    //              + "The number of DnaA's must be an integer");
                    e.printStackTrace();
                }
            // System.out.println("freeDnaA: " + myAgent.freeDnaA);
        }

        msg = myAgent.receive(msgTemplGet);

        if(msg != null) {
            nothingfound = false;

            amountDnaA = myAgent.freeDnaA/8; //!! Just took 8 out of nowhere :)

            // Due to the equation freeDnaA/smth the freeDnaA should not
            // be able to go negative. I hope. But when this equation is
            // changed, some control mechanism should be applied...
            if(0 < amountDnaA) {
                ACLMessage reply = msg.createReply();
                reply.setPerformative(ACLMessage.INFORM);
                reply.setContent(Integer.toString(amountDnaA));
                myAgent.send(reply);

                myAgent.freeDnaA -= amountDnaA;
            }

            // System.out.println("freeDnaA: " + myAgent.freeDnaA);
        }

        if(nothingfound)
            block();

    } //endof FUNCTION: action

} //endof CLASS
```

```
// File: BehBactDnaReg.java
// Description: Processes registration and deregistration messages from DNAs.
// Usage: inform :ConversationId:"DNA registration" "parentAID"
//            (this message should come from the new DNA agent)
//        inform :ConversationId:"DNA deregistration"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehBactDnaReg extends CyclicBehaviour {

    AgBacterium myAgent = null;
    MessageTemplate msgTemplReg = null;
    MessageTemplate msgTemplDeReg = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactDnaReg(AgBacterium a) {
        super(a);
        myAgent = a;

        // Constructing a template for recognizing the messages.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                    ACLMessage.INFORM);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                   "DNA registration");
        MessageTemplate msgT3 = MessageTemplate.MatchConversationId(
                                                   "DNA deregistration");
        msgTemplReg = MessageTemplate.and(msgT1, msgT2);
        msgTemplDeReg = MessageTemplate.and(msgT1, msgT3);

        // System.out.println("\nBehBactDnaReg Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        // Receiving the registration message.
        ACLMessage msg = myAgent.receive(msgTemplReg);
        if(msg != null) {
            nothingfound = false;

            String dnaName = msg.getSender().getName(),
                   parentName = null,
                   cont = msg.getContent();

            if("" == cont) {
                parentName = "null";
            } else {
                  AID a = new AID(msg.getContent(), AID.ISGUID);
                  parentName = a.getName();
              }
```

```
                // System.out.println("Got message from " + dnaName);
                // System.out.println("  that contains parentName: " + parentName);

                // Updating the graph of bacterium's DNA's.
                try {
                    myAgent.dgMyDNAs.addVertex(dnaName);
                } catch(NullPointerException e) {
                      e.printStackTrace();
                  }
                if(myAgent.dgMyDNAs.containsVertex(dnaName)
                     && myAgent.dgMyDNAs.containsVertex(parentName))
                    myAgent.dgMyDNAs.addEdge(dnaName, parentName);

                // System.out.println("\ndgMyDNAs: " + myAgent.dgMyDNAs.toString());

            }

            // Receiving the deregistration message.
            msg = myAgent.receive(msgTemplDeReg);
            if(msg != null) {
                nothingfound = false;

                String dnaName = msg.getSender().getName();

                // Updating the graph of bacterium's DNA's.
                if(!(myAgent.dgMyDNAs.removeVertex(dnaName)));
                    //System.out.println("Error: no such DNA in dgMyDNAs!");


                // System.out.println("\ndgMyDNAs: " + myAgent.dgMyDNAs.toString());
            }

            if(nothingfound)
                block();

        } //endof FUNCTION: action

    } //endof CLASS
```

```java
// File: BehBactGrowth.java
// Description: Increases the cell mass according to current
//              environment conditions.

package bactmodel;

import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.core.behaviours.TickerBehaviour;

//!!! Logging
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

// ======
// CLASS |
// ======
public class BehBactGrowth extends TickerBehaviour {

    AgBacterium myAgent = null;
    MessageTemplate msgTempl = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactGrowth(AgBacterium a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;

        String convID = "BehBactGrowth asks environment conditions";

        // Constructing a message for asking the environment conditions.
        myAgent.msgGetCondForGrowth = new ACLMessage(ACLMessage.QUERY_REF);
        myAgent.msgGetCondForGrowth.addReceiver(myAgent.envAID);
        myAgent.msgGetCondForGrowth.setConversationId(convID);
        myAgent.msgGetCondForGrowth.setContent("Conditions?");

        // Constructing a template for recognizing the reply.
        msgTempl = MessageTemplate.MatchConversationId(convID);

        // System.out.println("\nBehBactGrowth Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        // Asking for the environment conditions.
        try {
            myAgent.send(myAgent.msgGetCondForGrowth);
        } catch(Exception e) {
              e.printStackTrace();
          }

        // Using (all) the answer(s) to increase the cell mass.
        ACLMessage msg = null;
        while(null != (msg = myAgent.receive(msgTempl))) {
            String content = msg.getContent();
            try {
                int i = Integer.parseInt(content);
                myAgent.cellMass += i;
```

```
            } catch (NumberFormatException e) {
                e.printStackTrace();
              }
        }

        //!!! Logging
        try {
            BufferedWriter out = new BufferedWriter(
                                              new FileWriter("_log.txt", true));
            out.write(getTickCount() + " "
                    + myAgent.cellMass + " "
                    + myAgent.freeDnaA + " "
                    + myAgent.dgMyDNAs.vertexSet().size() + " "
                    + "\n");

            out.close();
        } catch (IOException e) {
            e.printStackTrace();
          }

        //System.out.println("cellMass: " + myAgent.cellMass);

    } //endof FUNCTION: onTick

} //endof CLASS
```

```java
// File: BehBactGui.java
// Description: After every x ms updates "time"
// and tells AgBacterium to refresh GUI.

package bactmodel;

import jade.core.behaviours.TickerBehaviour;

// ======
// CLASS |
// ======
public class BehBactGui extends TickerBehaviour {

    AgBacterium myAgent = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactGui(AgBacterium a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        myAgent.time++;
        myAgent.display();

    } //endof FUNCTION: onTick

} //endof CLASS
```

```
// File: BehBactInfoserver.java
// Description: Gives general information about bacterium variables.
// Usage: query-ref :ConversationId:"general bacterium info" "cellMass"
//                                                           "freeDnaA"
//                                                           "dgMyDNAs"
//                                                  "methylatedDnaAPromos"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehBactInfoserver extends CyclicBehaviour {

    AgBacterium myAgent = null;
    MessageTemplate msgTemplCellMass = null;
    MessageTemplate msgTemplFreeDnaA = null;
    MessageTemplate msgTemplDgMyDNAs = null;
    MessageTemplate msgTemplMetDnaAProms = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehBactInfoserver(AgBacterium a) {
        super(a);
        myAgent = a;

        // Constructing templates for recognizing relevant messages.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                    ACLMessage.QUERY_REF);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                "general bacterium info");
        MessageTemplate msgT3 = MessageTemplate.and(msgT1, msgT2);
        MessageTemplate msgT4 = MessageTemplate.MatchContent("cellMass");
        MessageTemplate msgT5 = MessageTemplate.MatchContent("freeDnaA");
        MessageTemplate msgT6 = MessageTemplate.MatchContent("dgMyDNAs");
        MessageTemplate msgT7 = MessageTemplate.MatchContent(
                                                  "methylatedDnaAPromos");
        msgTemplCellMass = MessageTemplate.and(msgT3, msgT4);
        msgTemplFreeDnaA = MessageTemplate.and(msgT3, msgT5);
        msgTemplDgMyDNAs = MessageTemplate.and(msgT3, msgT6);
        msgTemplMetDnaAProms = MessageTemplate.and(msgT3, msgT7);

        // System.out.println("\nBehBactInfoserver Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        ACLMessage msg = myAgent.receive(msgTemplCellMass);
        if(msg != null) {
            nothingfound = false;

            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(Integer.toString(myAgent.cellMass));
```

```
            myAgent.send(reply);
        }

        msg = myAgent.receive(msgTemplFreeDnaA);
        if(msg != null) {
            nothingfound = false;

            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(Integer.toString(myAgent.freeDnaA));
            myAgent.send(reply);
        }

        msg = myAgent.receive(msgTemplDgMyDNAs);
        if(msg != null) {
            nothingfound = false;

            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(myAgent.dgMyDNAs.toString());
            myAgent.send(reply);
        }

        msg = myAgent.receive(msgTemplMetDnaAProms);
        if(msg != null) {
            nothingfound = false;

            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(Integer.toString(myAgent.methylatedDnaAPromos));
            myAgent.send(reply);
        }

        if(nothingfound)
            block();

    } //endof FUNCTION: action

} //endof CLASS
```

```
// File: GuiBact.java
// Description: AgBacterium's graphical user interface.

package bactmodel;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.util.*;
import java.io.*;

import jade.core.*;

import java.awt.Color;
import aw.gui.chart.Chart2D;
import aw.gui.chart.ITrace2D;
import aw.gui.chart.Trace2DLtd;
import aw.util.Range;

// ======
// CLASS |
// ======
public class GuiBact extends JFrame {

    // Labels for displaying the numerical value of the variable
    private JLabel cellMassText;
    private JLabel freeDnaAText;
    private JLabel numOfDNAsText;
    private JLabel methylatedDnaAPromosText;

    private int GuiWidth = 500;
    private int GuiHeight = 600;

    //Charts and traces

    int numOfPoints = 200;

    Chart2D cellMassChart = new Chart2D();
    ITrace2D cellMassTrace = new Trace2DLtd(numOfPoints);

    Chart2D freeDnaAChart = new Chart2D();
    ITrace2D freeDnaATrace = new Trace2DLtd(numOfPoints);

    Chart2D numOfDNAsChart = new Chart2D();
    ITrace2D numOfDNAsTrace = new Trace2DLtd(numOfPoints);

    Chart2D methylatedDnaAPromosChart = new Chart2D();
    ITrace2D methylatedDnaAPromosTrace = new Trace2DLtd(numOfPoints);

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public GuiBact(AgBacterium a) {

        setTitle("GUI of " + a.getLocalName());

        cellMassChart.addTrace(cellMassTrace);
        freeDnaAChart.addTrace(freeDnaATrace);
        numOfDNAsChart.addTrace(numOfDNAsTrace);
        methylatedDnaAPromosChart.addTrace(methylatedDnaAPromosTrace);
```

```
        // - - - - -
        // Setting up the GUI layout
        // - - - - -
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));

        // panelCellMass
        JPanel panelCellMass = new JPanel();
        panelCellMass.setLayout(new BoxLayout(panelCellMass, BoxLayout.X_AXIS));
        JLabel cellMassLabel = new JLabel(" :cellMass___");
        cellMassText = new JLabel("x");
        panelCellMass.add(cellMassChart);
        panelCellMass.add(cellMassText);
        panelCellMass.add(cellMassLabel);
        mainPanel.add(panelCellMass);

        // panelFreeDnaA
        JPanel panelFreeDnaA = new JPanel();
        panelFreeDnaA.setLayout(new BoxLayout(panelFreeDnaA, BoxLayout.X_AXIS));
        JLabel freeDnaALabel = new JLabel(" :freeDnaA____");
        freeDnaAText = new JLabel("x");
        panelFreeDnaA.add(freeDnaAChart);
        panelFreeDnaA.add(freeDnaAText);
        panelFreeDnaA.add(freeDnaALabel);
        mainPanel.add(panelFreeDnaA);

        // panelNumOfDNAs
        JPanel panelNumOfDNAs = new JPanel();
        panelNumOfDNAs.setLayout(
                            new BoxLayout(panelNumOfDNAs, BoxLayout.X_AXIS));
        JLabel numOfDNAsLabel = new JLabel(" :numOfDNAs___");
        numOfDNAsText = new JLabel("x");
        panelNumOfDNAs.add(numOfDNAsChart);
        panelNumOfDNAs.add(numOfDNAsText);
        panelNumOfDNAs.add(numOfDNAsLabel);
        mainPanel.add(panelNumOfDNAs);

        // panelMethylatedDnaAPromos
        JPanel panelMethylatedDnaAPromos= new JPanel();
        panelMethylatedDnaAPromos.setLayout(
                  new BoxLayout(panelMethylatedDnaAPromos, BoxLayout.X_AXIS));
        JLabel methylatedDnaAPromosLabel = new JLabel(" :metDnaAProms");
        methylatedDnaAPromosText = new JLabel("x");
        panelMethylatedDnaAPromos.add(methylatedDnaAPromosChart);
        panelMethylatedDnaAPromos.add(methylatedDnaAPromosText);
        panelMethylatedDnaAPromos.add(methylatedDnaAPromosLabel);
        mainPanel.add(panelMethylatedDnaAPromos);

        // Add all this to the ContentPane and show
        getContentPane().add(mainPanel, BorderLayout.CENTER);
        showCorrect();

    } //endof FUNCTION: Constructor

    // --------------------------
    // FUNCTION: displayCellMass |
    // --------------------------
    void displayCellMass(int x, int cellMass) {
        cellMassTrace.addPoint(x, cellMass);
        cellMassText.setText(Integer.toString(cellMass));
    } //endof FUNCTION: displayCellMass

    // --------------------------
    // FUNCTION: displayFreeDnaA |
    // --------------------------
    void displayFreeDnaA(int x, int freeDnaA) {
```

```
        freeDnaATrace.addPoint(x, freeDnaA);
        freeDnaAText.setText(Integer.toString(freeDnaA));
    } //endof FUNCTION: displayfreeDnaA

    // ---------------------------
    // FUNCTION: displayNumOfDNAs |
    // ---------------------------
    void displayNumOfDNAs(int x, int numOfDNAs) {
        numOfDNAsTrace.addPoint(x, numOfDNAs);
        numOfDNAsText.setText(Integer.toString(numOfDNAs));
    } //endof FUNCTION: displayNumOfDNAs

    // ------------------------------
    // FUNCTION: displayMethylatedDnaAPromos |
    // ------------------------------
    void displayMethylatedDnaAPromos(int x, int methylatedDnaAPromos) {
        methylatedDnaAPromosTrace.addPoint(x, methylatedDnaAPromos);
        methylatedDnaAPromosText.setText(
                                    Integer.toString(methylatedDnaAPromos));
    } //endof FUNCTION: displayMethylatedDnaAPromos

    // ----------------------
    // FUNCTION: showCorrect |
    // ----------------------
    // Arranges and displays GUI window.
    void showCorrect() {
        pack();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int x = (screenSize.width - GuiWidth) / 2;
        int y = screenSize.height - GuiHeight - 30;
        setBounds(x, y, GuiWidth, GuiHeight);
        show();
    } //endof FUNCTION: showCorrect

} //endof CLASS
```

AgDnaAFactory

(2 faili)

```
// File: AgDnaAFactory
// Description: Produces DnaA protein (depending on the environment
//              conditions and the number of active DnaA promoters in cell).
// NB! When instantiating, you must supply two string arguments:
//     bacterium GUID and environment GUID (should be the same environment,
//     where bacterium is, of course :)

package bactmodel;

import jade.core.AID;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAException;

// ======
// CLASS |
// ======
public class AgDnaAFactory extends Agent {

    // In which bacterium this DnaAFactory is.
    AID bactAID = null;
    // In which environment this DnaAFactory is.
    AID envAID = null;
    // Execute production every x ms.
    int updPeriod = 120;

    // ----------------
    // FUNCTION: setup |
    // ----------------
    protected void setup() {
        // System.out.println("\nEntering " + getAID().getName() + " setup()");

        // Getting (command line) arguments
        Object[] args = getArguments();
        if (2 == args.length) {
            try {
                AID bAID = new AID((String)args[0], AID.ISGUID);
                bactAID = bAID;
                AID eAID = new AID((String)args[1], AID.ISGUID);
                envAID = eAID;
                // System.out.println("bactAID: " + bactAID.toString());
                // System.out.println("envAID: " + envAID.toString());
            } catch (Exception e) {
                  e.printStackTrace();
                  // System.out.println(
                  //    "\nERROR: Incorrect arguments. Will terminate.");
                  doDelete();
              }
        } else {
              // System.out.println(
              //    "\nERROR: Incorrect number of arguments. Will terminate.");
              doDelete();
        }

        // Adding main behaviours
        addBehaviour(new BehDnaAFactoryProducer(this, updPeriod));

        // System.out.println("\nLeaving " + getAID().getName() + " setup()");
    } //endof FUNCTION: setup
```

```
    // --------------------
    // FUNCTION: takeDown |
    // --------------------
    protected void takeDown() {
        // System.out.println("\nEntering " + getAID().getName()
        //                                  + " takeDown()");
        // System.out.println("\nLeaving " + getAID().getName()
        //                                 + " takeDown()");
    } //endof FUNCTION: takeDown

} //endof CLASS
```

```java
// File: BehDnaAFactoryProducer.java
// Description: Produces DnaA protein (depending on the environment
//              conditions and the number of active DnaA promoters in cell).

package bactmodel;

import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.core.behaviours.TickerBehaviour;

// ======
// CLASS |
// ======
public class BehDnaAFactoryProducer extends TickerBehaviour {

    AgDnaAFactory myAgent = null;

    ACLMessage msgGetCond = null;
    ACLMessage msgGetPromoters = null;
    ACLMessage msgSendProduction = null;

    MessageTemplate msgTemplCond = null;
    MessageTemplate msgTemplProm = null;

    int lastKnownConds = 0;
    int lastTimeConds = 0;
    int lastKnownProms = 0;
    int lastTimeProms = 0;

    // The maximal allowed age of a message (in ticks). If older,
    // then producing no DnaA due to lack of information.
    // NB! Here we have no idea about the time of *sending* those
    // messages, only about when they arrived here.
    int allowedAge = 2;

    // Conditions that were used the previous time (for comparing
    // with new ones: if changed, then resets ticking period).
    // !!! If bad conditions, then period long and does NOT respond
    // to new changes for a long time.
    int prevUsedConds = 0;
    boolean condsHaveChanged = false;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaAFactoryProducer(AgDnaAFactory a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;

        // Constructing a message for asking the environment conditions.
        String convID = "BehDnaAFactoryProducer asks environment conditions";
        msgGetCond = new ACLMessage(ACLMessage.QUERY_REF);
        msgGetCond.addReceiver(myAgent.envAID);
        msgGetCond.setConversationId(convID);
        msgGetCond.setContent("Conditions?");

        // Constructing a template for recognizing the reply.
        msgTemplCond = MessageTemplate.MatchConversationId(convID);

        // Constructing a message for asking about the number of active
        // DnaA promoters.
        convID = "general bacterium info";
        msgGetPromoters = new ACLMessage(ACLMessage.QUERY_REF);
```

```
        msgGetPromoters.addReceiver(myAgent.bactAID);
        msgGetPromoters.setConversationId(convID);
        msgGetPromoters.setContent("methylatedDnaAPromos");

        // Constructing a template for recognizing the reply.
        msgTemplProm = MessageTemplate.MatchConversationId(convID);

        // Constructing a message for sending produced DnaA to bacterium.
        convID = "Store DnaA";
        msgSendProduction = new ACLMessage(ACLMessage.REQUEST);
        msgSendProduction.addReceiver(myAgent.bactAID);
        msgSendProduction.setConversationId(convID);

        // System.out.println("\nBehDnaAFactoryProducer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        // Asking for the environment conditions.
        try {
            myAgent.send(msgGetCond);
        } catch(Exception e) {
              e.printStackTrace();
          }

        // Asking about the number of active DnaA promoters.
        try {
            myAgent.send(msgGetPromoters);
        } catch(Exception e) {
              e.printStackTrace();
          }

        int curTick = getTickCount();

        // If any answers have arrived since last tick,
        // then memorizing the last answer and arriving time.

        ACLMessage msg = null;
        while(null != (msg = myAgent.receive(msgTemplCond))) {
            String content = msg.getContent();
            try {
                lastKnownConds = Integer.parseInt(content);
                lastTimeConds = curTick;
            } catch (NumberFormatException e) {
                  e.printStackTrace();
              }
        }

        msg = null;
        while(null != (msg = myAgent.receive(msgTemplProm))) {
            String content = msg.getContent();
            try {
                lastKnownProms = Integer.parseInt(content);
                lastTimeProms = curTick;
            } catch (NumberFormatException e) {
                  e.printStackTrace();
              }
        }
```

```
        // If info is fresh enough and conditions are positive,
        // then producing DnaA.
        int newDnaAs = 0;
        if((curTick - lastTimeConds <= allowedAge) &&
           (curTick - lastTimeProms <= allowedAge) &&
           (0 < lastKnownConds)) {
               newDnaAs = lastKnownProms; // (Dependence on conditions is
                                          // regulated with ticking period)
               if(prevUsedConds != lastKnownConds)
                   condsHaveChanged = true;
               // System.out.println("Produced " + newDnaAs + " DnaAs");
        } /* else
              System.out.println("Produced no DnaA."
                                 + " No fresh info or bad conditions");
          */

        // Sending created DnaAs to the bacterium.
        if(0 < newDnaAs) {
            msgSendProduction.setContent(Integer.toString(newDnaAs));
            try {
                myAgent.send(msgSendProduction);
            } catch(Exception e) {
                  e.printStackTrace();
              }
        }

        // If conditions have changed, then resetting the ticking period
        // (this regulates the dependence of production on the conditions).
        if((condsHaveChanged) && (0 < lastKnownConds)) {
            prevUsedConds = lastKnownConds;
            long newUpdPeriod = 6000 / lastKnownConds;
            reset(newUpdPeriod);
        }

    } //endof FUNCTION: onTick

} //endof CLASS
```

AgDNA

(11 faili)

```
// File: AgDNA.java
// Description: DNA double helix, either complete or under construction.
// NB! When instantiating, you must supply three arguments:
//     bacterium GUID and parentDNA GUID (or "null", if no parentDNA)
//     and position of the parent (if don't know, give 0).
//     If parent is given, then a just-starting-to-be-assembled DNA is created,
//     otherwise it will be a ready-made DNA.

package bactmodel;

import jade.core.AID;
import jade.core.Agent;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;

import jade.gui.GuiAgent;
import jade.gui.GuiEvent;

// ======
// CLASS |
// ======
public class AgDNA extends GuiAgent {

    // In which bacterium this DNA is.
    AID bactAID = null;
    // Which DNA this DNA was copied from.
    AID parDnaAID = null;

    // Update info every x ms
    int updPeriod = 100;
    // Percentage of replication done.
    float pcentOfRepDone = 0;
    // Speed of replication (adds x percent points every updPeriod)
    float speedOfRep = (float)0.25;
    // How many non-initiating DnaA boxes are in a mature DNA.
    int maxDnaABxsNonInit = 305;
    // How many initiating DnaA boxes are in a mature DNA.
    // Actually here it is the number of ALL DnaAs needed for initiation
    // complex (not all will be binded to boxes).
    int maxDnaABxsInit = 30;
    // How many non-initiating DnaA boxes currently exist.
    // (grows from 0 to max, depending on how much of the replication is done).
    int curTotalDnaABxsNonInit = 0;
    // How many initiating DnaA boxes currently exist (here max all the time,
    // because they are produced right at the beginning of replication).
    int curTotalDnaABxsInit = maxDnaABxsInit;
    // How many non-initiating DnaA boxes are filled with DnaA.
    int filledDnaABxsNonInit = 0;
    // How many initiating DnaA boxes are filled with DnaA.
    int filledDnaABxsInit = 0;
    // Initiator can be filled with DnaA only when it is methylated.
    boolean initiatorMethylated = false;
    // DnaA promoter can be used only when it is methylated.
    boolean dnaAPromoterMethylated = false;

    // AgDNA's graphical user interface.
    transient protected GuiDNA gui;
    // Non-astronomical time for drawing graphs on GUI.
    int time = 0;
    // For remembering the position, so that new DNA GUI
    // will be placed elsewhere.
    int guiPosition = 0;
```

```
// ----------------
// FUNCTION: setup |
// ----------------
protected void setup() {
    // System.out.println("\nEntering " + getAID().getName() + " setup()");

    // Getting (command line) arguments
    Object[] args = getArguments();
    if (3 == args.length) {
        try {
            AID bAID = new AID((String)args[0], AID.ISGUID);
            bactAID = bAID;
            // System.out.println("bactAID: " + bactAID.toString());

            if("null" == (String)args[1]) {
                pcentOfRepDone = 100;
                curTotalDnaABxsNonInit = maxDnaABxsNonInit;
                filledDnaABxsNonInit = maxDnaABxsNonInit / 2;
                initiatorMethylated = true;
                dnaAPromoterMethylated = true;
            } else {
                  AID pdAID = new AID((String)args[1], AID.ISGUID);
                  parDnaAID = pdAID;
                  addBehaviour(new BehDnaReplicator(this, updPeriod));
                  addBehaviour(
                              new BehDnaDnaAPromMethylator(this, updPeriod));
                  addBehaviour(new BehDnaInitMethylator(this, updPeriod));
              }

            guiPosition = Integer.parseInt((String)args[2]);

        } catch (Exception e) {
              e.printStackTrace();
              // System.out.println(
              //    "\nERROR: Incorrect arguments. Will terminate.");
              doDelete();
          }
    } else {
          // System.out.println(
          //    "\nERROR: Incorrect number of arguments. Will terminate.");
          doDelete();
    }

    // Register in bacterium.
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.addReceiver(bactAID);
    msg.setConversationId("DNA registration");
    if(null == parDnaAID) {
        msg.setContent("null");
        // System.out.println("parDnaAID: null");
    } else {
          msg.setContent(parDnaAID.getName());
          // System.out.println("parDnaAID: " + parDnaAID.toString());
      }

    try {
        send(msg);
    } catch(Exception e) {
          e.printStackTrace();
          // System.out.println("\nERROR: Sending threw an exception.");
      }

    // Adding main behaviours
    addBehaviour(new BehDnaOwnerServer(this));
    addBehaviour(new BehDnaMethylServer(this));
    addBehaviour(new BehDnaInfoserver(this));
```

```
        addBehaviour(new BehDnaDnaABoxFiller(this, updPeriod));
        addBehaviour(new BehDnaRemDnaAServer(this));

/*!!
        // Creating and showing the GUI
        gui = new GuiDNA(this);
        gui.setVisible(true);
        addBehaviour(new BehDnaGui(this, updPeriod));
*/
        // System.out.println("\nLeaving " + getAID().getName() + " setup()");
    } //endof FUNCTION: setup

    // -------------------
    // FUNCTION: takeDown |
    // -------------------
    protected void takeDown() {
        // System.out.println("\nEntering " + getAID().getName()
        //                                  + " takeDown()");

        //Deregister in bacterium.
        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(bactAID);
        msg.setConversationId("DNA deregistration");
        try {
            send(msg);
        } catch(Exception e) {
              e.printStackTrace();
          }

        // Disposing the GUI
        if (gui!=null) {
          gui.dispose();
          gui.setVisible(false);
        }

        // System.out.println("\nleaving " + getAID().getName()
        //                                  + " takeDown()");
    } //endof FUNCTION: takeDown

    // ------------------
    // FUNCTION: display |
    // ------------------
    // Gives new info to GUI
    void display() {
        gui.displayPcentOfRepDone(time, (int)pcentOfRepDone);
        gui.displayCurTotalDnaABxsNonInit(time, curTotalDnaABxsNonInit);
        gui.displayFilledDnaABxsNonInit(time, filledDnaABxsNonInit);
        gui.displayFilledDnaABxsInit(time, filledDnaABxsInit);

        int iMet = 0;
        if(initiatorMethylated) iMet = 1;
        gui.displayInitiatorMethylated(time, iMet);

        int dapMet = 0;
        if(dnaAPromoterMethylated) dapMet = 1;
        gui.displayDnaAPromoterMethylated(time, dapMet);
    } //endof FUNCTION: display

    // ---------------------
    // FUNCTION: onGuiEvent |
    // ---------------------
    // Currently non-interactive gui
    protected void onGuiEvent(GuiEvent ev) {}
    //endof FUNCTION: onGuiEvent

} //endof CLASS
```

```
// File: BehDnaDnaABoxFiller.java
// Description: Fills the DnaA boxes with free DnaA from bacterium.
//              Also, if everything is full, starts the replication
//              process (creates new starting-to-replicate DNA agent).

package bactmodel;

import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.wrapper.StaleProxyException;

// ======
// CLASS |
// ======
public class BehDnaDnaABoxFiller extends TickerBehaviour {

    AgDNA myAgent = null;
    ACLMessage msgGetDnaA = null;
    ACLMessage msgAskAboutFreeDnaA = null;
    ACLMessage msgSendDnaA = null;
    MessageTemplate msgTemplReceiveDnaA = null;
    MessageTemplate msgTemplAboutFreeDnaA = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaDnaABoxFiller(AgDNA a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;

        // Constructing a message for getting the DnaA.
        String convID = "Get DnaA";
        msgGetDnaA = new ACLMessage(ACLMessage.REQUEST);
        msgGetDnaA.addReceiver(myAgent.bactAID);
        msgGetDnaA.setConversationId(convID);

        // Constructing a template for recognizing the reply.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                        ACLMessage.INFORM);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(convID);
        msgTemplReceiveDnaA = MessageTemplate.and(msgT1, msgT2);

        // Constructing a message for asking about the total freeDnaA.
        convID = "General bacterium info";
        msgAskAboutFreeDnaA = new ACLMessage(ACLMessage.QUERY_REF);
        msgAskAboutFreeDnaA.addReceiver(myAgent.bactAID);
        msgAskAboutFreeDnaA.setConversationId(convID);
        msgAskAboutFreeDnaA.setContent("freeDnaA");

        // Constructing a template for recognizing the reply.
        msgT2 = MessageTemplate.MatchConversationId(convID);
        msgTemplAboutFreeDnaA = MessageTemplate.and(msgT1, msgT2);

        // Constructing a message for sending back unused DnaA.
        msgSendDnaA = new ACLMessage(ACLMessage.REQUEST);
        msgSendDnaA.addReceiver(myAgent.bactAID);
        msgSendDnaA.setConversationId("Store DnaA");

        // System.out.println("\nBehDnaDnaABoxFiller Constructed");
    } //endof FUNCTION: Constructor
```

```
    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        // Receiving information about the amount of free DnaA in bacterium.
        // If more than one msg, then using the last one.
        int freeDnaA = 0;
        ACLMessage msg = null;
        while(null != (msg = myAgent.receive(msgTemplAboutFreeDnaA))) {
            String content = msg.getContent();
            try {
                freeDnaA = Integer.parseInt(content);
            } catch (NumberFormatException e) {
                  e.printStackTrace();
              }
        }

        // Receiving DnaA. If more than one msg, then summing them together.
        int myDnaA = 0;
        msg = null;
        while(null != (msg = myAgent.receive(msgTemplReceiveDnaA))) {
            String content = msg.getContent();
            try {
                int i = Integer.parseInt(content);
                myDnaA += i;
            } catch (NumberFormatException e) {
                  e.printStackTrace();
              }
        }

        boolean needMoreDnaA = false;

        // If non-init boxes not full, then fills them.
        // Makes sure they don't get "overfilled".
        if(myAgent.filledDnaABxsNonInit < myAgent.curTotalDnaABxsNonInit) {
            myAgent.filledDnaABxsNonInit += myDnaA;
            myDnaA = 0;
            int dif = myAgent.filledDnaABxsNonInit
                        - myAgent.curTotalDnaABxsNonInit;
            if(dif > 0) {
                myAgent.filledDnaABxsNonInit -= dif;
                myDnaA += dif;
            }
            if(dif < 0)
                needMoreDnaA = true;
        }

        // If non-init boxes full, the asks about the amount
        // of free DnaA in bacterium.
        if(!needMoreDnaA) {
            try {
                myAgent.send(msgAskAboutFreeDnaA);
            } catch(Exception e) {
                  e.printStackTrace();
              }
        }

/* !!
        // If non-init boxes full, initiator methylated and more than 10
        // freeDnaA in bacterium, then fills init boxes.
        if(!needMoreDnaA && myAgent.initiatorMethylated && (freeDnaA > 10)) {
*/
```

```
        // If initiator methylated and more than 10 freeDnaA in bacterium,
        // then fills init boxes.
        if(myAgent.initiatorMethylated && (freeDnaA > 40)) {
            myAgent.filledDnaABxsInit += myDnaA;
            myDnaA = 0;
            int dif = myAgent.filledDnaABxsInit - myAgent.curTotalDnaABxsInit;
            if(dif > 0) {
                myAgent.filledDnaABxsInit -= dif;
                myDnaA += dif;
            }
            if(dif < 0)
                needMoreDnaA = true;
        }

        if(needMoreDnaA) {
            try {
                myAgent.send(msgGetDnaA);
            } catch(Exception e) {
                  e.printStackTrace();
              }
        }

        // If initiator is filled, then starting DNA replication:
        // empties initiator (this DnaA gets sent back to bacterium),
        // unmethylates (or more precisely hemimethylates?) initiator
        // and promoter, and creates a new starting-to-replicate DNA agent.
        if(myAgent.filledDnaABxsInit == myAgent.maxDnaABxsInit) {
            myDnaA += myAgent.filledDnaABxsInit;
            myAgent.filledDnaABxsInit = 0;
            myAgent.initiatorMethylated = false;
            myAgent.dnaAPromoterMethylated = false;
            // Creating new DNA.
            String newName = "dna" + System.currentTimeMillis();
            int newGuiPosition = myAgent.guiPosition + 1;
            if(4 < newGuiPosition)
                newGuiPosition = 0;
            String[] argsDNA = {myAgent.bactAID.getName(), myAgent.getName(),
                                Integer.toString(newGuiPosition)};
            try {
                myAgent
                  .getContainerController()
                  .createNewAgent(newName, "bactmodel.AgDNA", argsDNA)
                  .start();
            } catch(StaleProxyException e) {
                  e.printStackTrace();
              }
        }

        // Sending unused DnaA back.
        if(0 < myDnaA) {
            msgSendDnaA.setContent(Integer.toString(myDnaA));
            try {
                myAgent.send(msgSendDnaA);
            } catch(Exception e) {
                  e.printStackTrace();
              }
        }

    } //endof FUNCTION: onTick

} //endof CLASS
```

```java
// File: BehDnaDnaAPromMethylator.java
// Description: Methylates DnaA promoter at a certain moment.
//              Also requests parent DNA to do so. Then terminates.

package bactmodel;

import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

// ======
// CLASS |
// ======
public class BehDnaDnaAPromMethylator extends TickerBehaviour {

    AgDNA myAgent = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaDnaAPromMethylator(AgDNA a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;
        // System.out.println("\nBehDnaDnaAPromMethylator Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {
        if(myAgent.pcentOfRepDone > 55) {
            myAgent.dnaAPromoterMethylated = true;
            // System.out.println("MethylBeh metylated promoter");

            ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
            msg.addReceiver(myAgent.parDnaAID);
            msg.setConversationId("Methylate");
            msg.setContent("DnaA promoter");
            try {
                myAgent.send(msg);
            } catch(Exception e) {
                  e.printStackTrace();
              }

            stop();
        }
    } //endof FUNCTION: onTick

} //endof CLASS
```

```java
// File: BehDnaGui.java
// Description: After every x ms updates "time"
// and tells AgDNA to refresh GUI.

package bactmodel;

import jade.core.behaviours.TickerBehaviour;

// ======
// CLASS |
// ======
public class BehDnaGui extends TickerBehaviour {

    AgDNA myAgent = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaGui(AgDNA a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        myAgent.time++;
        myAgent.display();

    } //endof FUNCTION: onTick

} //endof CLASS
```

```
// File: BehDnaInfoserver.java
// Description: Gives general information about DNA variables.
// Usage: query-ref :ConversationId:"general DNA info" "bactName"
//                                                    "parDnaName"
//                                                    "dnaAPromoterMethylated"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehDnaInfoserver extends CyclicBehaviour {

    AgDNA myAgent = null;
    MessageTemplate msgTemplBactName = null;
    MessageTemplate msgTemplParDnaName = null;
    MessageTemplate msgTemplDnaAPromMet = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaInfoserver(AgDNA a) {
        super(a);
        myAgent = a;

        // Constructing templates for recognizing relevant messages.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                    ACLMessage.QUERY_REF);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                   "general DNA info");
        MessageTemplate msgT3 = MessageTemplate.and(msgT1, msgT2);
        MessageTemplate msgT4 = MessageTemplate.MatchContent("bactName");
        MessageTemplate msgT5 = MessageTemplate.MatchContent("parDnaName");
        MessageTemplate msgT6 = MessageTemplate.MatchContent(
                                                   "dnaAPromoterMethylated");
        msgTemplBactName = MessageTemplate.and(msgT3, msgT4);
        msgTemplParDnaName = MessageTemplate.and(msgT3, msgT5);
        msgTemplDnaAPromMet = MessageTemplate.and(msgT3, msgT6);

        // System.out.println("\nBehDnaInfoserver Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        ACLMessage msg = myAgent.receive(msgTemplBactName);
        if(msg != null) {
            nothingfound = false;

            ACLMessage reply = msg.createReply();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent(myAgent.bactAID.getName());
            myAgent.send(reply);
        }

        msg = myAgent.receive(msgTemplParDnaName);
```

```
            if(msg != null) {
                nothingfound = false;

                ACLMessage reply = msg.createReply();
                reply.setPerformative(ACLMessage.INFORM);
                String content = null;
                if(null == myAgent.parDnaAID) {
                    content = "null";
                } else
                      content = myAgent.parDnaAID.getName();
                reply.setContent(content);
                myAgent.send(reply);
            }

            msg = myAgent.receive(msgTemplDnaAPromMet);
            if(msg != null) {
                nothingfound = false;

                ACLMessage reply = msg.createReply();
                reply.setPerformative(ACLMessage.INFORM);
                reply.setContent(Boolean.toString(myAgent.dnaAPromoterMethylated));
                myAgent.send(reply);
            }

            if(nothingfound)
                block();

        } //endof FUNCTION: action

    } //endof CLASS
```

```java
// File: BehDnaInitMethylator.java
// Description: Methylates initiator (OriC) at a certain moment.
//              Also requests parent DNA to do so. Then terminates.

package bactmodel;

import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

// ======
// CLASS |
// ======
public class BehDnaInitMethylator extends TickerBehaviour {

    AgDNA myAgent = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaInitMethylator(AgDNA a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;
        // System.out.println("\nBehDnaInitMethylator Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {
        if(myAgent.pcentOfRepDone > 20) {
            myAgent.initiatorMethylated = true;
            // System.out.println("MethylBeh metylated initiator");

            ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
            msg.addReceiver(myAgent.parDnaAID);
            msg.setConversationId("Methylate");
            msg.setContent("Initiator");
            try {
                myAgent.send(msg);
            } catch(Exception e) {
                  e.printStackTrace();
              }

            stop();
        }
    } //endof FUNCTION: onTick

} //endof CLASS
```

```
// File: BehDnaMethylServer.java
// Description: Methylates DNA segments if requested.
// Usage: request :ConversationId:"Methylate" "DnaA promoter"
//                                            "Initiator"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehDnaMethylServer extends CyclicBehaviour {

    AgDNA myAgent = null;
    MessageTemplate msgTempl = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaMethylServer(AgDNA a) {
        super(a);
        myAgent = a;

        // Constructing a template for recognizing methylation requests.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                        ACLMessage.REQUEST);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                            "Methylate");
        msgTempl = MessageTemplate.and(msgT1, msgT2);

        // System.out.println("\nBehDnaMethylServer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        ACLMessage msg = myAgent.receive(msgTempl);
        if(msg != null) {
            String content = msg.getContent();

            if(content.equals("DnaA promoter")) {
                myAgent.dnaAPromoterMethylated = true;
                // System.out.println("MethylServer methylated promoter");
            }

            if(content.equals("Initiator")) {
                myAgent.initiatorMethylated = true;
                // System.out.println("MethylServer methylated initiator");
            }

        } else block();

    } //endof FUNCTION: action

} //endof CLASS
```

```
// File: BehDnaOwnerServer
// Description: Changes the owner information of DNA (owner = bacterium).
// Usage: request :ConversationId:"Set new owner"
//            (this message should come from the new owner)
//        request :ConversationId:"Kill"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehDnaOwnerServer extends CyclicBehaviour {

    AgDNA myAgent = null;
    MessageTemplate msgTemplSetNew = null;
    MessageTemplate msgTemplKill = null;
    MessageTemplate msgTemplSetAndKill = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaOwnerServer(AgDNA a) {
        super(a);
        myAgent = a;

        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                    ACLMessage.REQUEST);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                    "Set new owner");
        MessageTemplate msgT3 = MessageTemplate.MatchConversationId(
                                                    "Kill");
        msgTemplSetNew = MessageTemplate.and(msgT1, msgT2);
        msgTemplKill = MessageTemplate.and(msgT1, msgT3);
        // System.out.println("\nBehDnaOwnerServer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {
        boolean nothingfound = true;

        ACLMessage msg = myAgent.receive(msgTemplSetNew);
        if(msg != null) {
            nothingfound = false;
            myAgent.bactAID = msg.getSender();
            // System.out.println(myAgent.getAID().getName()
            //                    + " set its owner to "
            //                    + myAgent.bactAID.toString());
        }

        msg = myAgent.receive(msgTemplKill);
        if(msg != null) {
            nothingfound = false;
            myAgent.doDelete();
        }

        if(nothingfound)
            block();
```

```
    } //endof FUNCTION: action

} //endof CLASS
```

```
// File: BehDnaRemDnaAServer.java
// Description: If asked, removes some amount of non-init DnaA
//              and sends it back to the bacterium.
// Usage: request :ConversationId:"RemoveNonInitDnaA" "integer"

package bactmodel;

import jade.core.behaviours.CyclicBehaviour;
import jade.core.AID;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

// ======
// CLASS |
// ======
public class BehDnaRemDnaAServer extends CyclicBehaviour {

    AgDNA myAgent = null;
    MessageTemplate msgTemplRem = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaRemDnaAServer(AgDNA a) {
        super(a);
        myAgent = a;

        // Constructing a template for recognizing incoming request.
        MessageTemplate msgT1 = MessageTemplate.MatchPerformative(
                                                        ACLMessage.REQUEST);
        MessageTemplate msgT2 = MessageTemplate.MatchConversationId(
                                                    "RemoveNonInitDnaA");
        msgTemplRem = MessageTemplate.and(msgT1,msgT2);

        // System.out.println("\nBehDnaRemDnaAServer Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {

        // How many DnaA's to remove
        int howMany = 0;

        // Summing up removal requests
        ACLMessage msg = null;
        while(null != (msg = myAgent.receive(msgTemplRem))) {
            String content = msg.getContent();
            try {
                int i = Integer.parseInt(content);
                howMany += i;
            } catch (NumberFormatException e) {
                  e.printStackTrace();
              }
        }

        // Removing DnaA and sending back to bacterium
        myAgent.filledDnaABxsNonInit -= howMany;

        ACLMessage msgSendDnaA = new ACLMessage(ACLMessage.REQUEST);
        msgSendDnaA.addReceiver(myAgent.bactAID);
        msgSendDnaA.setConversationId("Store DnaA");
        msgSendDnaA.setContent(Integer.toString(howMany));
```

```
            try {
                myAgent.send(msgSendDnaA);
            } catch(Exception e) {
                  e.printStackTrace();
              }

            block();

        } //endof FUNCTION: action

    } //endof CLASS
```

```
// File: BehDnaReplicator.java
// Description: Performs replication of DNA
//              (currently only increases the value of pcentOfRepDone).

package bactmodel;

import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

import java.lang.Math;

// ======
// CLASS |
// ======
public class BehDnaReplicator extends TickerBehaviour {

    AgDNA myAgent = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehDnaReplicator(AgDNA a, long tickPeriod) {
        super(a, tickPeriod);
        myAgent = a;
        // System.out.println("\nBehDnaReplicator Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: onTick |
    // -----------------
    protected void onTick() {

        float prevPcentOfRepDone = myAgent.pcentOfRepDone;

        myAgent.pcentOfRepDone += myAgent.speedOfRep;
        if(100 < myAgent.pcentOfRepDone)
            myAgent.pcentOfRepDone = 100;
        // System.out.println(myAgent.getLocalName() + " pcent: "
        //                                           + myAgent.pcentOfRepDone);

        // Tells parent to free those DnaA boxes which were just under
        // replication (I guess the replicator pushes DnaA away from there?).
        int i = Math.round(myAgent.pcentOfRepDone - prevPcentOfRepDone);
        String howMany = Integer.toString(i);

        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(myAgent.parDnaAID);
        msg.setConversationId("RemoveNonInitDnaA");
        msg.setContent(howMany);
        try {
            myAgent.send(msg);
        } catch(Exception e) {
              e.printStackTrace();
          }

        // Linear dependence here. In reality not so.
        myAgent.curTotalDnaABxsNonInit
               = (int)(myAgent.pcentOfRepDone/100 * myAgent.maxDnaABxsNonInit);
        // System.out.println(myAgent.getLocalName()
        //                    + " noninit: "
        //                    + myAgent.curTotalDnaABxsNonInit);
```

```
            // When finished, notify bacterium and stop replication behaviour.
            if(100 == myAgent.pcentOfRepDone) {
                msg = new ACLMessage(ACLMessage.INFORM);
                msg.addReceiver(myAgent.bactAID);
                msg.setContent("DNA replication completed");
                try {
                    myAgent.send(msg);
                } catch(Exception e) {
                      // System.out.println("\nERROR: Sending threw an exception.");
                  }
                stop();
            }

        } //endof FUNCTION: onTick

    } //endof CLASS
```

```
// File: GuiDNA.java
// Description: AgDNA's graphical user interface.

package bactmodel;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import java.util.*;
import java.io.*;

import jade.core.*;

import java.awt.Color;
import aw.gui.chart.Chart2D;
import aw.gui.chart.ITrace2D;
import aw.gui.chart.Trace2DLtd;
import aw.util.Range;

// ======
// CLASS |
// ======
public class GuiDNA extends JFrame {

    AgDNA myAgent = null;

    // Labels for displaying the numerical value of the variable
    private JLabel pcentOfRepDoneText;
    private JLabel curTotalDnaABxsNonInitText;
    private JLabel filledDnaABxsNonInitText;
    private JLabel filledDnaABxsInitText;
    private JLabel initiatorMethylatedText;
    private JLabel dnaAPromoterMethylatedText;

    private int GuiWidth = 500;
    private int GuiHeight = 600;

    //Charts and traces

    int numOfPoints = 200;

    Chart2D pcentOfRepDoneChart = new Chart2D();
    ITrace2D pcentOfRepDoneTrace = new Trace2DLtd(numOfPoints);

    Chart2D curTotalDnaABxsNonInitChart = new Chart2D();
    ITrace2D curTotalDnaABxsNonInitTrace = new Trace2DLtd(numOfPoints);

    Chart2D filledDnaABxsNonInitChart = new Chart2D();
    ITrace2D filledDnaABxsNonInitTrace = new Trace2DLtd(numOfPoints);

    Chart2D filledDnaABxsInitChart = new Chart2D();
    ITrace2D filledDnaABxsInitTrace = new Trace2DLtd(numOfPoints);

    Chart2D initiatorMethylatedChart = new Chart2D();
    ITrace2D initiatorMethylatedTrace = new Trace2DLtd(numOfPoints);

    Chart2D dnaAPromoterMethylatedChart = new Chart2D();
    ITrace2D dnaAPromoterMethylatedTrace = new Trace2DLtd(numOfPoints);
```

```
// ----------------------
// FUNCTION: Constructor |
// ----------------------
public GuiDNA(AgDNA a) {

    myAgent =  a;
    setTitle("GUI of " + myAgent.getLocalName());

    pcentOfRepDoneChart.addTrace(pcentOfRepDoneTrace);
    curTotalDnaABxsNonInitChart.addTrace(curTotalDnaABxsNonInitTrace);
    filledDnaABxsNonInitChart.addTrace(filledDnaABxsNonInitTrace);
    filledDnaABxsInitChart.addTrace(filledDnaABxsInitTrace);
    initiatorMethylatedChart.addTrace(initiatorMethylatedTrace);
    dnaAPromoterMethylatedChart.addTrace(dnaAPromoterMethylatedTrace);

    // - - - - -
    // Setting up the GUI layout
    // - - - - -
    JPanel mainPanel = new JPanel();
    mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));

    // panelPcentOfRepDone
    JPanel panelPcentOfRepDone = new JPanel();
    panelPcentOfRepDone.setLayout(
                  new BoxLayout(panelPcentOfRepDone, BoxLayout.X_AXIS));
    JLabel pcentOfRepDoneLabel = new JLabel(" :pcentOfRepDone_____");
    pcentOfRepDoneText = new JLabel("x");
    panelPcentOfRepDone.add(pcentOfRepDoneChart);
    panelPcentOfRepDone.add(pcentOfRepDoneText);
    panelPcentOfRepDone.add(pcentOfRepDoneLabel);
    mainPanel.add(panelPcentOfRepDone);

    // panelCurTotalDnaABxsNonInit
    JPanel panelCurTotalDnaABxsNonInit = new JPanel();
    panelCurTotalDnaABxsNonInit.setLayout(
            new BoxLayout(panelCurTotalDnaABxsNonInit, BoxLayout.X_AXIS));
    JLabel curTotalDnaABxsNonInitLabel =
                                    new JLabel(" :curTotalDnaABxsNonInit");
    curTotalDnaABxsNonInitText = new JLabel("x");
    panelCurTotalDnaABxsNonInit.add(curTotalDnaABxsNonInitChart);
    panelCurTotalDnaABxsNonInit.add(curTotalDnaABxsNonInitText);
    panelCurTotalDnaABxsNonInit.add(curTotalDnaABxsNonInitLabel);
    mainPanel.add(panelCurTotalDnaABxsNonInit);

    // panelFilledDnaABxsNonInit
    JPanel panelFilledDnaABxsNonInit = new JPanel();
    panelFilledDnaABxsNonInit.setLayout(
              new BoxLayout(panelFilledDnaABxsNonInit, BoxLayout.X_AXIS));
    JLabel filledDnaABxsNonInitLabel =
                                   new JLabel(" :filledDnaABxsNonInit___");
    filledDnaABxsNonInitText = new JLabel("x");
    panelFilledDnaABxsNonInit.add(filledDnaABxsNonInitChart);
    panelFilledDnaABxsNonInit.add(filledDnaABxsNonInitText);
    panelFilledDnaABxsNonInit.add(filledDnaABxsNonInitLabel);
    mainPanel.add(panelFilledDnaABxsNonInit);

    // panelFilledDnaABxsInit
    JPanel panelFilledDnaABxsInit = new JPanel();
    panelFilledDnaABxsInit.setLayout(
                 new BoxLayout(panelFilledDnaABxsInit, BoxLayout.X_AXIS));
    JLabel filledDnaABxsInitLabel = new JLabel(" :filledDnaABxsInit_____");
    filledDnaABxsInitText = new JLabel("x");
    panelFilledDnaABxsInit.add(filledDnaABxsInitChart);
    panelFilledDnaABxsInit.add(filledDnaABxsInitText);
    panelFilledDnaABxsInit.add(filledDnaABxsInitLabel);
    mainPanel.add(panelFilledDnaABxsInit);
```

```
        // panelInitiatorMethylated
        JPanel panelInitiatorMethylated = new JPanel();
        panelInitiatorMethylated.setLayout(
                    new BoxLayout(panelInitiatorMethylated, BoxLayout.X_AXIS));
        JLabel initiatorMethylatedLabel =
                                        new JLabel(" :initiatorMethylated_____");
        initiatorMethylatedText = new JLabel("x");
        panelInitiatorMethylated.add(initiatorMethylatedChart);
        panelInitiatorMethylated.add(initiatorMethylatedText);
        panelInitiatorMethylated.add(initiatorMethylatedLabel);
        mainPanel.add(panelInitiatorMethylated);

        // panelDnaAPromoterMethylated
        JPanel panelDnaAPromoterMethylated = new JPanel();
        panelDnaAPromoterMethylated.setLayout(
                 new BoxLayout(panelDnaAPromoterMethylated, BoxLayout.X_AXIS));
        JLabel dnaAPromoterMethylatedLabel =
                                          new JLabel(" :dnaAPromMethylated___");
        dnaAPromoterMethylatedText = new JLabel("x");
        panelDnaAPromoterMethylated.add(dnaAPromoterMethylatedChart);
        panelDnaAPromoterMethylated.add(dnaAPromoterMethylatedText);
        panelDnaAPromoterMethylated.add(dnaAPromoterMethylatedLabel);
        mainPanel.add(panelDnaAPromoterMethylated);

        // Add all this to the ContentPane and show
        getContentPane().add(mainPanel, BorderLayout.CENTER);
        showCorrect();

    } //endof FUNCTION: Constructor

    // --------------------------------
    // FUNCTION: displayPcentOfRepDone |
    // --------------------------------
    void displayPcentOfRepDone(int x, int pcentOfRepDone) {
        pcentOfRepDoneTrace.addPoint(x, pcentOfRepDone);
        pcentOfRepDoneText.setText(Integer.toString(pcentOfRepDone));
    } //endof FUNCTION: displayPcentOfRepDone

    // ----------------------------------------
    // FUNCTION: displayCurTotalDnaABxsNonInit |
    // ----------------------------------------
    void displayCurTotalDnaABxsNonInit(int x, int curTotalDnaABxsNonInit) {
        curTotalDnaABxsNonInitTrace.addPoint(x, curTotalDnaABxsNonInit);
        curTotalDnaABxsNonInitText.setText(
                                       Integer.toString(curTotalDnaABxsNonInit));
    } //endof FUNCTION: displayCurTotalDnaABxsNonInit

    // -------------------------------------
    // FUNCTION: displayFilledDnaABxsNonInit |
    // -------------------------------------
    void displayFilledDnaABxsNonInit(int x, int filledDnaABxsNonInit) {
        filledDnaABxsNonInitTrace.addPoint(x, filledDnaABxsNonInit);
        filledDnaABxsNonInitText.setText(
                                        Integer.toString(filledDnaABxsNonInit));
    } //endof FUNCTION: displayFilledDnaABxsNonInit

    // ---------------------------------
    // FUNCTION: displayFilledDnaABxsInit |
    // ---------------------------------
    void displayFilledDnaABxsInit(int x, int filledDnaABxsInit) {
        filledDnaABxsInitTrace.addPoint(x, filledDnaABxsInit);
        filledDnaABxsInitText.setText(Integer.toString(filledDnaABxsInit));
    } //endof FUNCTION: displayFilledDnaABxsInit
```

```
    // ------------------------------------
    // FUNCTION: displayInitiatorMethylated |
    // ------------------------------------
    void displayInitiatorMethylated(int x, int initiatorMethylated) {
        initiatorMethylatedTrace.addPoint(x, initiatorMethylated);
        initiatorMethylatedText.setText(Integer.toString(initiatorMethylated));
    } //endof FUNCTION: displayInitiatorMethylated

    // ----------------------------------------
    // FUNCTION: displayDnaAPromoterMethylated |
    // ----------------------------------------
    void displayDnaAPromoterMethylated(int x, int dnaAPromoterMethylated) {
        dnaAPromoterMethylatedTrace.addPoint(x, dnaAPromoterMethylated);
        dnaAPromoterMethylatedText.setText(
                                    Integer.toString(dnaAPromoterMethylated));
    } //endof FUNCTION: displayDnaAPromoterMethylated

    // ----------------------
    // FUNCTION: showCorrect |
    // ----------------------
    // Arranges and displays GUI window.
    void showCorrect() {
        pack();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int x = 0;
        int y = 0;

        y = (myAgent.guiPosition / 2) * 50;
        if(0 != (myAgent.guiPosition % 2)) {
            x = screenSize.width - GuiWidth;
        }

        setBounds(x, y, GuiWidth, GuiHeight);
        show();
    } //endof FUNCTION: showCorrect

} //endof CLASS
```

General

(2 faili)

```java
// File: BehAddServiceToDF.java
// Description: Publishes a service in Directory Facilitator

package bactmodel;

import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;

// For using Directory Facilitator
import jade.core.AID;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;

// ======
// CLASS |
// ======
public class BehAddServiceToDF extends OneShotBehaviour {

    private String srvType;
    private String srvName;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public BehAddServiceToDF(Agent a,
                             String serviceType, String serviceName) {
        super(a);
        srvType = serviceType;
        srvName = serviceName;
        // System.out.println("\nBehAddServiceToDF Constructed");
    } //endof FUNCTION: Constructor

    // -----------------
    // FUNCTION: action |
    // -----------------
    public void action() {

        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(myAgent.getAID());

        ServiceDescription sd = new ServiceDescription();
        sd.setType(srvType);
        sd.setName(srvName);

        dfd.addServices(sd);
        try {
            DFService.register(myAgent, dfd);
        } catch (FIPAException fe) {
              fe.printStackTrace();
        }

        // System.out.println("\nService "
        //                    + sd.getType() + " " + sd.getName()
        //                    + " registered in DF");
    } //endof FUNCTION: Action

} //endof CLASS
```

```java
// File: ServiceFinder.java
// Description: Tries to find (from Directory Facilitator) the agents,
//              who offer given service. Stores found agents' AIDs.
//              NB! Does NOT update it's info, when something changes in DF.

package bactmodel;

import jade.core.AID;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;

// ======
// CLASS |
// ======
public class hServiceFinder {

    AID[] agentsWithService = null;

    // ----------------------
    // FUNCTION: Constructor |
    // ----------------------
    public hServiceFinder(Agent myAgent, String srvType, String srvName) {
        DFAgentDescription adTemplate = new DFAgentDescription();
        ServiceDescription sdForTemplate = new ServiceDescription();
        sdForTemplate.setType(srvType);
        sdForTemplate.setName(srvName);
        adTemplate.addServices(sdForTemplate);

        try {
            DFAgentDescription[] foundAgentsDescriptors
                = DFService.search(myAgent, adTemplate);
            agentsWithService = new AID[foundAgentsDescriptors.length];
            for(int i = 0; i < agentsWithService.length; ++i) {
                agentsWithService[i] = foundAgentsDescriptors[i].getName();
            }
        } catch (FIPAException fe) {
              fe.printStackTrace();
          }
    } //endof FUNCTION: Constructor

} //endof CLASS
```