Lints, T. (2004). ACE – raamprogramm hajussüsteemide valmistamiseks. *A&A*, (6): 14–25. In Estonian, published by TTŰ Kirjastus.

T. Lints, "ACE – raamprogramm hajussüsteemide valmistamiseks," *A&A*, pp. 14–25, December 2004. In Estonian, published by TTŰ Kirjastus.

```
@article{Lints04_ACE,
  author = {Taivo Lints},
  title = {{ACE} -- raamprogramm hajuss\"{u}steemide
valmistamiseks},
  year = {2004},
  journal = {A\&A},
  number = {6},
  pages = {14-25},
  month = {December},
  note = {In Estonian, published by TT\"{U} Kirjastus}
}
```

ACE – raamprogramm hajussüsteemide valmistamiseks

1. Sissejuhatus

Paralleel- ja hajussüsteemide valmistamine nii "tavalistest" arvutitest kui ka kõikvõimalikust spetsiifilisemast riistvarast (sh. sardsüsteemidest *(embedded systems)*) on hoolimata infotehnoloogia kiirest arengust ikkagi üsna keeruline. Lihtsama arvutiklastri saab tekitada küll vähese vaevaga – piisab, kui võrku ühendatud arvutid käivitada vabalt kättesaadava klastritarkvaraga (nt. *Bootable Cluster CD* [1]), kuid kasutajarakenduste programmeerimine sellele klastrile on sellegipoolest vaevanõudev. Käesolevas artiklis vaadeldakse mõningaid võimalusi hajustarkvara arendamisele kuluva aja ja vaeva vähendamiseks: mustreid *(patterns)*, raamprogramme *(frameworks)*, vahetarkvara *(middleware),* ja konkreetset objektorienteeritud programmeerimisvahendit *ACE (The Adaptive Communication Environment)*, mis neil võimalustel põhineb.

2. Mustrid, raamprogrammid, vahetarkvara

2.1. Probleemid tarkvara loomisel

Tarkvaraarendus on alati raske olnud, kuid varasematel aegadel on siiski olnud võimalik enamik programme valmis kirjutada nö. nullist alustades – põhinedes ainult konkreetsete programmeerijate isiklikel kogemustel. Arvutimaailm aga muutub üha keerukamaks ning nõudmised suuremaks: soovitakse saada ennustatava käitumisega, usaldatavat, skaleeruvat, turvalist tarkvara, mis oleks võimeline töötama paljudel erinevatel riistvaraplatvormidel ja operatsioonisüsteemidel ning mitmesugustes arvutivõrkudes [2]. Veelgi enam, selliseid süsteeme tuleb luua üha kasvava globaalse konkurentsi tingimustes, kus toodete valmimise aeg, aga ka eelarve, on üha rohkem piiratud. Eriti suur on huvi ja nõudluse kasv just hajussüsteemide valdkonnas, põhjuseks hajussüsteemide arvukad positiivsed omadused: koostöö lihtsustumine üle suurte vahemaade, jõudluse kasv tänu paralleeltöötlusele, töökindlus tänu ressursside liiasusele, skaleeritavus tänu modulaarsusele, dünaamilise (re)konfigureerimise võimalused ning tasuvus tänu ressursside jagamisele ja avatud süsteemide ideoloogiale [3]. Samas on hajussüsteemid üks kõige keerulisemaid arvutisüsteemide liike üldse [4].

Hajustarkvara kirjutamisel tekitavad probleeme mitmed asjaolud [3]:

- Esiteks on vaja lahendada keerukaid ülesandeid, mis on hajussüsteemidele olemuslikud, näiteks võrgus ja arvutites tekkivate vigade avastamine ning nendest taastumine, infovahetuse viidete mõju minimeerimine, optimaalse tarkvarakomponentide ja töökoormuse jaotuse leidmine.
- Teine probleemide allikas on tarkvaraarenduse tööriistade ning tehnikate piiratus. Näiteks paljude standartsete võrgumehhanismide (nt. *sockets* ja *TLI (Transport Layer Interface)*) ja taaskasutatavate teekide (nt. *X windows* ja *Sun RPC*) rakendusliidesed *(APIs)* on problemaatilised: osaliselt või täielikult puuduvad muutujate tüübikontrollid, porditavus, taassisenetavus *(re-entrance)*, laiendatavus.
- Kolmas raskuste põhjustaja on algoritmilise dekompositsiooni kasutamine, mille tulemuseks on mittelaiendatavad ja taaskasutamiseks kõlbmatud tarkvarasüsteemid.

Nendest raskustest jagu saamiseks – kvaliteetse tarkvara loomiseks mõistliku ressursikuluga – on vaja edukate mudelite, disainide ja teostuste süstemaatilist taaskasutust [2]. Kui süsteemipäratu-oportunistliku taaskasutuse korral arendajad lihtsalt kopeerivad

olemasolevatest programmidest koodilõike uutesse süsteemidesse, siis süstemaatiline taaskasutus on tahtlik ja organiseeritud jõupingutus taaskasutatavate tarkvarakomponentide loomiseks ning kasutamiseks. Edukalt toimiva taaskasutuse korral kasutab iga uus projekt ära varasemaid edukaks osutunud disaine ja teostusi, lisades ainult spetsiifilist konkreetse rakenduse jaoks vajalikku koodi. Olemasolevaid tarkvaraarhitektuure ning disaine muudetakse sel juhul ümber vaid siis, kui nad enam ei vasta muutunud nõuetele. Süstemaatiline taaskasutus võimaldab ära jätta tüüpiliste tarkvarakomponentide kuluka ning vigaderohke taasavastamise-leiutamise ning taastestimise.

Süstemaatilise taaskasutuse näideteks on mustrid ja raamprogrammid, ning vahetarkvara, mis (soovitatavalt) nendel kahel põhineb.

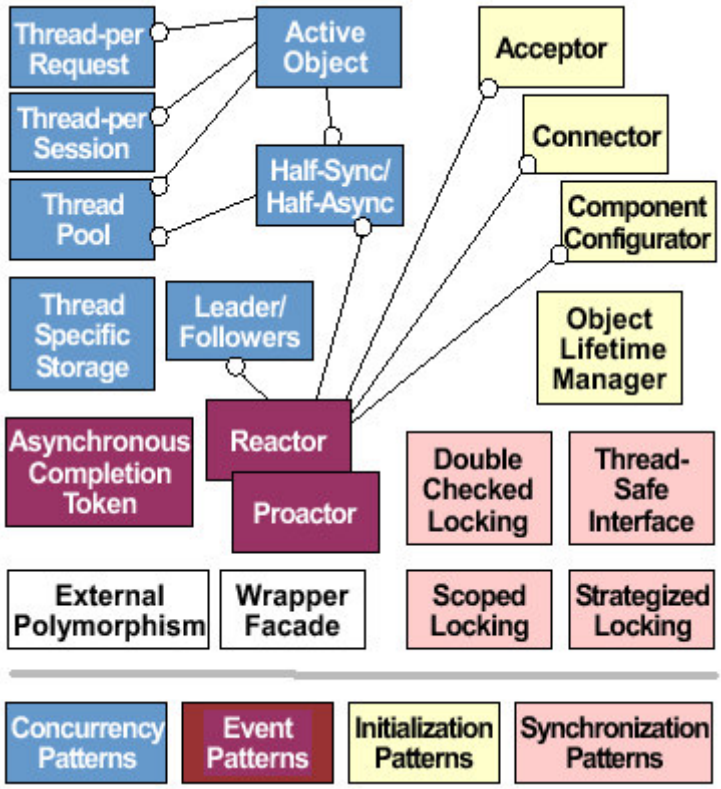
2.2. Mustrid *(Patterns)*

Paljud probleemid, mis tarkvara loomise käigus esile kerkivad (näiteks teenuste initsialiseerimine, ühenduste haldamine, vigadele reageerimine, usaldatavus), korduvad rakendusest rakendusse, valdkonnast valdkonda. Nende lahendamiseks vajalikud teadmised eksisteerisid kuni 1990-te keskpaigani peamiselt programmeerimisfolklooris, ekspertide ja arendajate peades või maetuna sügavale keerukasse lähtekoodi [2]. Need asukohad aga ei ole head, sest:

- Mustrite leidmine lähtekoodist on kallis ja aeganõudev, kuna on raske eristada disaini olemust tema teostusest tulenevatest detailidest.
- Kui projekteerijate kogemused jäävad dokumenteerimata, siis lähevad nad aja jooksul kaduma ja seega ei osale hilisemas tarkvara evolutsioonis.

Viimase kümnendi jooksul on kogenud tarkvaraarendajad ja -arhitektid aidanud neid probleeme lahendada, dokumenteerides taaskasutatavat teadmist järgmistel viisidel:

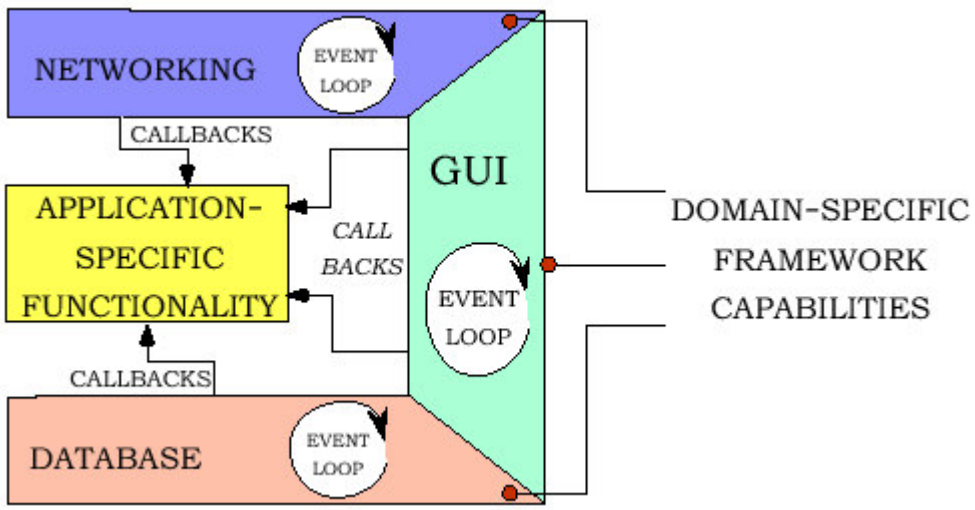
- Disainimustrid, mis kirjeldavad programmikomponentide sellist struktuuri ning omavahelisi suhteid, mis lahendab mõne konkreetse levinud disainiprobleemi.
- Arhitektuurimustrid, mis kirjeldavad tarkvarasüsteemi tervikstruktuuri ja pakuvad välja juhiseid selle terviku jaotamisel alamsüsteemideks.
- Mustrikeeled, mis toovad kokku hulga omavahel seotud mustreid, defineerides tarkvaraprobleemidest rääkimiseks sobiva sõnavara ja pakkudes välja protsessi nende probleemide lahendamiseks (näide mustrikeelest joonisel 1).



Joonis 1. Mustrikeel hajusprogrammeerimise jaoks [2]. Siinkohal on muidugi toodud vaid keele illustratsioon. Keel ise kirjeldab põhjalikult kõik mustrid ja seosed.

2.3. Raamprogrammid *(Frameworks)*

Raamprogramm on grupi omavahel seotud mustrite konkreetne realisatsioon [5]. Joonisel 2 on kujutatud raamprogrammi olemust:



Joonis 2. Raamprogrammi komponentide omavahelised suhted [2].

Sellisel raamprogrammil on järgnevad omadused [2]:

- "Juhtimise inversioon" *(inversion of control)* programmi töö ajal. See tähendab, et peaprogrammiks on raamprogramm, mis teatavate sündmuste avastamise korral

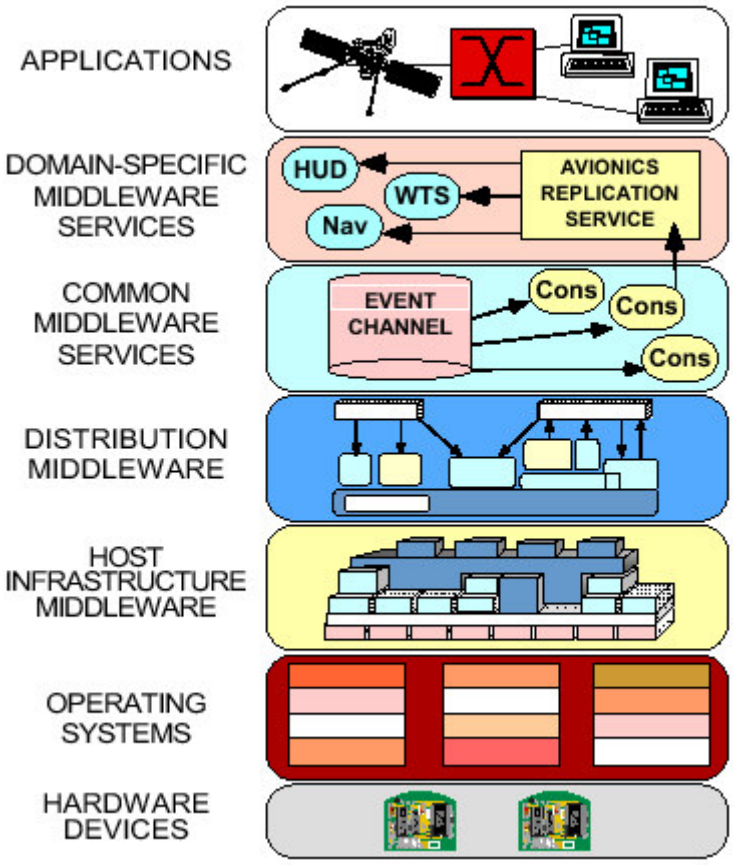
(näiteks hiireklikk või andmete saabumine võrgust) kutsub välja eelnevalt registreeritud rakendusspetsiifilisi komponente (vastupidiselt tavalisele teekide kasutamisele, kus kasutaja poolt kirjutatud rakendus kutsub välja taaskasutatavaid teegifunktsioone).

- Ühtseks integreeritud valdkonnaspetsiifilised struktuurid ja funktsionaalsus. Raamprogrammi taaskasutatavus sõltub suurel määral sellest, kui hästi ta modelleerib konkreetse valdkonna ühtsust ning muutlikkust, s.t. kas ta pakub lahendust sagedaselt esinevatele probleemidele ning samal ajal ei takista rakendusspetsiifiliste omaduste realiseerimist.
- Seega raamprogramm on poolvalmis rakendus, mida programmeerijad saavad kohandada oma vajadustele sobivaks.

Raamprogrammid pärinevad peamiselt graafiliste kasutajaliideste valdkonnast, kus neid on juba pikaajaliselt kasutatud (näiteks *MacApp, X-windows, Interviews, Java Swing, Microsoft Foundation Classes*). Viimasel ajal on aga sama ideoloogiat rakendatud ka paljudes teistes valdkondades, näiteks *ACE* ja *TAO*, millest tagapool juttu tuleb, on raamprogrammid hajussüsteemide vahetarkvara teatud kihtidele, *JBoss* ja *BEA's Weblogic Server* rakendusserveritele, *Open Grid Services Infrastructure* ja paljud teised keskenduvad veebiteenustele. On ka märksa spetsiifilisemaid raamprogramme, näiteks s*yngo* meditsiiniliste läbivalgustusaparaatide tarvis.

2.4. Vahetarkvara

Vahetarkvara ülesandeks on kaitsta programmeerijat hajussüsteemi madalama taseme probleemide eest, võimaldades enamuse ressurssidest kulutada konkreetse rakenduse spetsiifilistele probleemidele. See määratlus sarnaneb teataval määral raamprogrammide iseloomustusega ning tõepoolest – vahetarkvara ongi võimalik realiseerida raamprogrammi(de)na, saavutamaks eelnevates alampeatükkides nimetatud positiivseid omadusi.



Joonis 3. Hajussüsteem jaotatuna kihtideks [2].

Vahetarkvara võib parema ülevaate ning mõistlikuma realiseerimise huvides jagada mitmeks kihiks (4 keskmist kihti joonisel 3) [2]:

- Arvuti infrastruktuuri vahetarkvara kapseldab ja parendab baasoperatsioonisüsteemi mehhanisme, pakkudes näiteks taaskasutatavaid sündmuste demultipleksimise, protsessidevahelise suhtlemise, paralleelsuse-konkurentsuse ja sünkroniseerimisega tegelevaid klasse. Kapseldades konkreetsete operatsioonisüsteemide iseärasused, aitavad need klassid oluliselt vähendada tüütut ja vigadealdist madaltaseme programmeerimist ning suurendavad porditavust. Tuntud infrastruktuuri vahetarkvarad on näiteks *Sun Java Virtual Machine* ja *Microsoft's Common Language Runtime*. Sellesse kihti kuulub ka *The Adaptive Communication Environment (ACE)*.

- Hajususega tegelev vahetarkvara, mis võimaldab programmeerijal vaadelda süsteemi kui tervikut ning vabastab vajadusest eelnevalt teada objektide täpseid füüsilisi asukohti ja nende baasarvutite operatsioonisüsteeme, suhtlusprotokolle ja võrgutehnoloogiaid ning riistvara. Sellesse kihti kuuluvad eelkõige päringuvahendajad *(request brokers),* nagu näiteks *OMG's Common Object Request Broker Architecture (CORBA), Sun's Java Remote Method Invocation (RMI)* ja *Simple Object Access Protocol (SOAP)*, aga ka *The ACE ORB (TAO)*, mis on *The Adaptive Communication Environment*'i baasil valmistatud avatud lähtekoodiga reaalaja *CORBA ORB* [5].

- Üldised vahetarkvara teenused täiendavad hajutamisega tegelevat vahetarkvara kihti, pakkudes valdkonnast sõltumatuid kõrgema taseme taaskasutatavaid teenuseid. Näitena võib tuua *OMG's CORBA Common Object Services*: sündmustest

teavitamine, logimine, multimeedia striimimine, turvalisus, globaalaeg, veakindlus, transaktsioonid jms. Analoogselt võimaldavad ka *Sun' s Enterprise Java Beans* ning *Microsoft' s.NET* valmistada hajussüsteeme olemasolevate teenusekomponentide kokkulinkimise teel.

- Valdkonnaspetsiifilised vahetarkvara teenused, mis vastavad konkreetsete valdkondade nõudmistele ja soovidele, näiteks *ACE+TAO* 'l põhinevad *Siemens syngo* (platvorm kõikvõimalike meditsiiniliste läbivalgustusmasinate integreerimiseks ühtsesse arstidele mugavasse süsteemi [6]) ja *Boeing Bold Stroke* (lennuki pardaarvutussüsteemi arhitektuur, mis ühendab navigatsiooniseadmed, piloodi näidikud ja ekraanid, sensorid ning relvade sihtimise ja tulistamise mehhanismid [5]).

3. The Adaptive Communication Environment (Peatükk põhineb allikal [7])

3.1. Mis on *ACE*?

ACE on objektorienteeritud raamprogramm, mis realiseerib olulisemad paralleelsuse ja hajutamisega seotud mustrid. Joonisel 3 toodud jaotuses asub ta arvuti infrastruktuuri vahetarkvara kihis.

ACE sisaldab suure hulga taaskasutatavaid C++ "ümbriseid" *(wrappers)* ja raamprogrammi komponente, mis on suunatud suure jõudlusega reaalajarakenduste arendajatele. Need komponendid pakuvad järgmiste kommunikeeruvale tarkvarale iseloomulike ülesannete taaskasutatavaid realisatsioone:

- Ühenduste loomine ja teenuste initsialiseerimine.
- Sündmuste demultipleksimine ja edastamine vastavate sündmuste teenindajatele.
- Protsessidevaheline suhtlemine ja ühismälu *(shared memory)* haldamine.
- Kommunikatsiooniteenuste staatiline ja dünaamiline seadistamine.
- Hajutatud kommunikatsiooniteenused, näiteks arvutite "nimedega" tegelemine, sündmuste marsruutimine, logimine, aja sünkroniseerimine, ja lukustamine üle võrgu.

Lisaks on olemas *ACE* 'l põhinevaid kõrgema taseme vahetarkvarasid-komponente, nagu objektipäringuvahendajad *(Object Request Brokers)* ja veebiserverid.

ACE on avatud lähtekoodiga ning töötab paljudel erinevatel operatsioonisüsteemidel (tabel 1). Olemas on ka *Java* versioon *ACE'*st.

Grupp	**Platvormid**
DOC Group	Igapäevaselt testitakse ja kasutatakse järgnevatel platvormidel: *Solaris 2.6*, *7* ja *8* (paljud kompilaatorid, v.a. *SunC++ 4.x*), *Windows NT 4.0, 2000, XP (MSVC++ 6.x, 7.x, Borland C++ Builder 5.0), Linux/Intel* (paljud kompilaatorid), *Linux/IA64 (GCC).*
Riverace	Pakub tuge mitmesugustele platvormidele, sh. eelnimetatud, *HP-UX, AIX,* ja *Windows CE.*
OCI	Mõningad platvormid, mida neil parajasti oma *TAO* tarkvara ja teenuste pakkumiseks vaja on.
Remedy IT	Lisaks *DOC Group*'i poolt toetatavatele veel: *Borland C++ Builder 6, CBuilderX 1.0, VxWorks 5.5, OpenVMS 7.3-2, Linux 64bit Alpha,* ja *Tru64*.

ACE kasutajaskond	Kõikvõimalikud platvormid, mida eeltoodud grupid ei arenda, näiteks: *Windows 95/98 (Borland C++ Builder 4.0* ja hilisemad*, GNU g++ (MinGW, Cygwin)); Digital UNIX (Compaq Tru64) 4.0* ja *5.0; IRIX 6.x; UnixWare 7.1.0; SunOS 4.x* ja *Solaris (SunC++ 4.x); Linux on Alpha and PPC; OpenMVS; Tandem; SCO; FreeBSD; NetBSD; OpenBSD; Chorus; OS/9; PharLap TNT Embedded ToolSuite 9.1; QNX RTP* ja *Neutrino 2.0; VxWorks; LynxOS; RTEMS*.

Tabel 1. *ACE*' i arendusgrupid ja platvormid [13].

ACE' i kasutajaskond on väga lai ning isegi sponsorite nimekiri on üsna aukartustäratav [14]: *ATD, BBN, Boeing, CDI/GDIS, Cisco, Comverse, DARPA, Ericsson, Experian, Global MAINTECH, Hughes Network Systems, Kodak , Krones, Lockheed Martin, Lucent, Microsoft, Mitre, Motorola CGISS, Motorola Iridium, OCI, Oresis, OTI, Nokia, Nortel , NSF, QNX, Raytheon, Riverace, SAIC, Siemens ATD, Siemens MED, Siemens SCR, Siemens ZT, Sprint, Telcordia, USENIX*.

3.2. *ACE* 'i struktuur ja funktsionaalsus

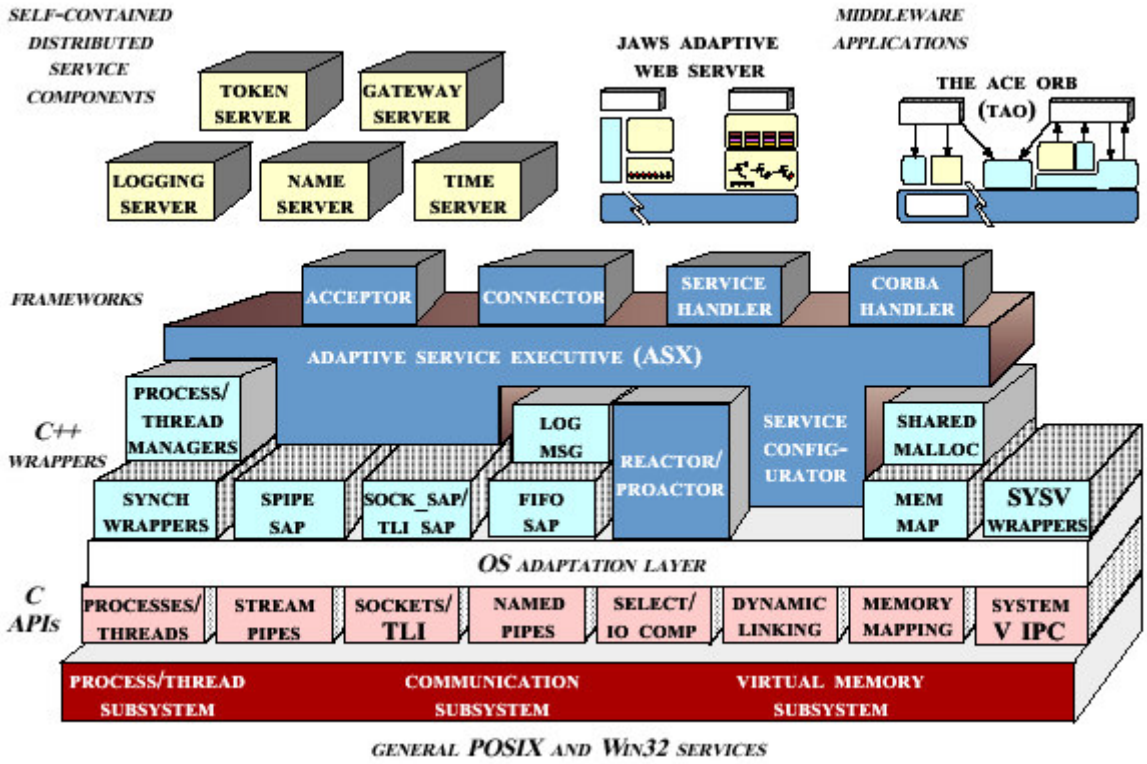
ACE raamprogramm sisaldab ~150 000 rida *C++* koodi jagatuna ~450 klassiks. Erinevate probleemide üksteisest eraldamiseks ning keerukuse vähendamiseks on kasutatud kihilist arhitektuuri (joonis 4).

3.2.1. Operatsioonisüsteemide "adapterkiht"

Moodustab umbes 13% kogu *ACE* ' i koodist. Kaitseb ülejäänud komponente platvormi-spetsiifilisuse eest, mis on seotud järgmiste operatsioonisüsteemi teenustega:

- Hargtöötlus *(multi-threading)*, multitöötlus *(multi-processing)* ja sünkroniseerimine.
- Protsessidevaheline kohalik ja kaugsuhtlus, ühismälu.
- Sünkroonne ja asünkroonne sisendil/väljundil, taimeritel, signaalidel ja sünkronisatsioonil põhinevate sündmuste demultipleksimine.
- Dünaamiline linkimine.
- Failide ja kataloogidega manipuleerimine.

Tänu OS adapteri poolt pakutavale abstraktsioonile on olnud võimalik *ACE* üles ehitada ühtse süsteemina, ilma iga platvormi jaoks eraldi versiooni loomata. Selline disain tõstab oluliselt *ACE* ' i porditavust ja hooldatavust.



Joonis 4. *ACE* 'i kihiline arhitektuur (all) ja lisakomponendid (üleval) [7].

3.2.2. C++ ümbrised *(wrappers)*

Põhimõtteliselt on võimalik kirjutada kergesti porditavaid rakendusi ka otse OS adapteri peale, aga enamus arendajaid eelistab siiski kasutada *C++* ümbriseid või veel kõrgemaid *ACE*'i kihte. *C++* ümbrised struktureerivad klassideks muidu eraldiseisvad *C* funktsioonid, pakuvad tugevat muutujate tüübikontrolli (kuid võimaldavad vajadusel ka sellest möödapääsemist), lihtsustavad mitmete funktsioonide kasutamist (nt. võimaldades programmeerijal mitte iga kord ette anda selliseid argumente, millede väärtused rakenduse töö käigus ei muutu) ja kõrvaldavad veaohtlikke olukordi (nt. sooritades klassi konstruktoris kõik vajalikud tegevused, mis tihti ununema kipuvad (nt. mõne olulise muutuja nullimine)) [8]. Vältimaks ümbriste kasutamisest tulenevat programmide jõudluskadu, on kasutatud mitmesuguseid optimeerimistehnikaid, näiteks *inlining*'ut (väikeste funktsioonide koodi kirjutamine otse kasutuskohta, vältimaks funktsiooni väljakutsest tulenevat lisakoormust).

C++ ümbrised moodustavad ~50% *ACE*'i koodist. Rakendused võivad neid klasse kasutada pärimise ja agregeerimise teel või neist lihtsalt objekte tekitada.

3.2.3. Raamprogrammi komponendid

Moodustavad ülejäänud ~40% *ACE*'i koodist, integreerides ja täiendades *C++* ümbriseid. Raamprogrammi komponentideks on:

- *Reactor* ja *Proactor*: laiendatavad objektorienteeritud demultiplekserid, mis reageerivad mitmesugustele sisendil/väljundil, taimeritel, signaalidel ja sünkronisatsioonil põhinevatele sündmustele vastavate rakendusspetsiifiliste komponentide väljakutsumisega.
- *Connector* ja *Acceptor*: eraldavad vastavalt aktiivse ja passiivse ühenduse initsialiseerimise rakendusspetsiifilistest ülesannetest, mida täidetakse pärast ühenduse loomist.

- *Service Configurator*: toetab rakenduste loomist, mille teenuseid saab dünaamiliselt lisada ja eemaldada installeerimise ja/või töö käigus.
- *Streams components*: lihtsustavad hierarhiliste-kihiliste teenuste (nt. protokolli-kihtide *(protocol stack)*) realiseerimist.
- *ORB adapter components*: võimaldavad *ACE* 'i integreerida *CORBA* süsteemidesse.

3.2.4. Iseseisvad *(self-contained)* hajusteenuste komponendid

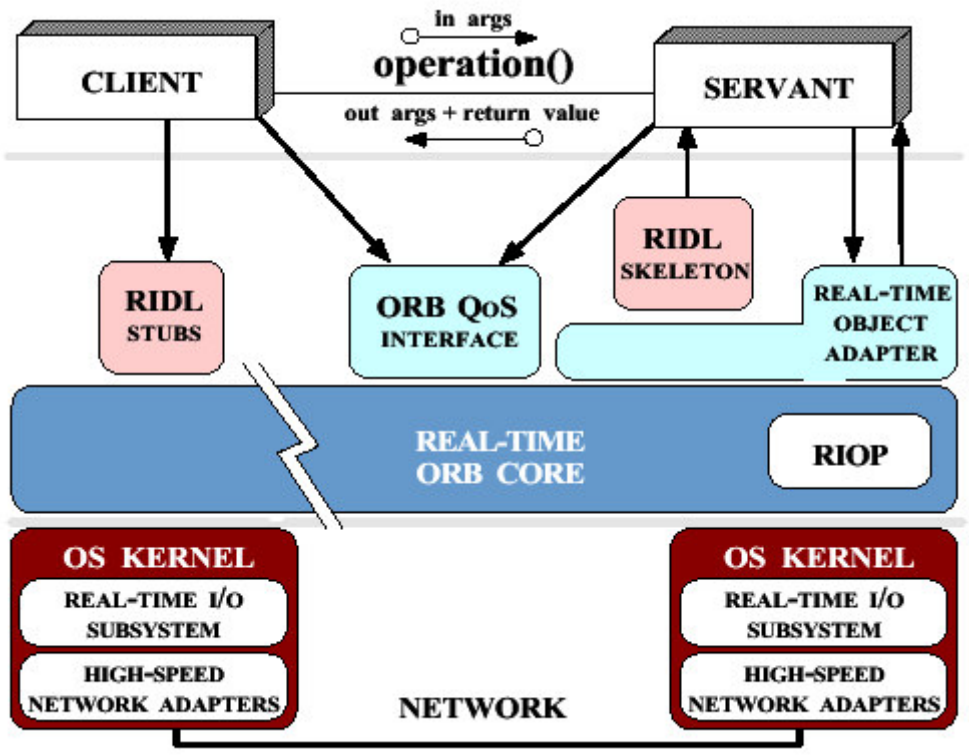
Need komponendid ei kuulu rangelt võttes *ACE* raamprogrammi, kuid sellegipoolest on neil kaks olulist rolli:

- Taaskasutatavad realisatsioonid tihtiesinevate ülesannete täitmiseks: arvutite "nimedega" tegelemine, sündmuste marsruutimine, logimine, aja sünkroniseerimine, ja lukustamine üle võrgu.
- *ACE*'i komponentide kasutamise demonstreerimine.

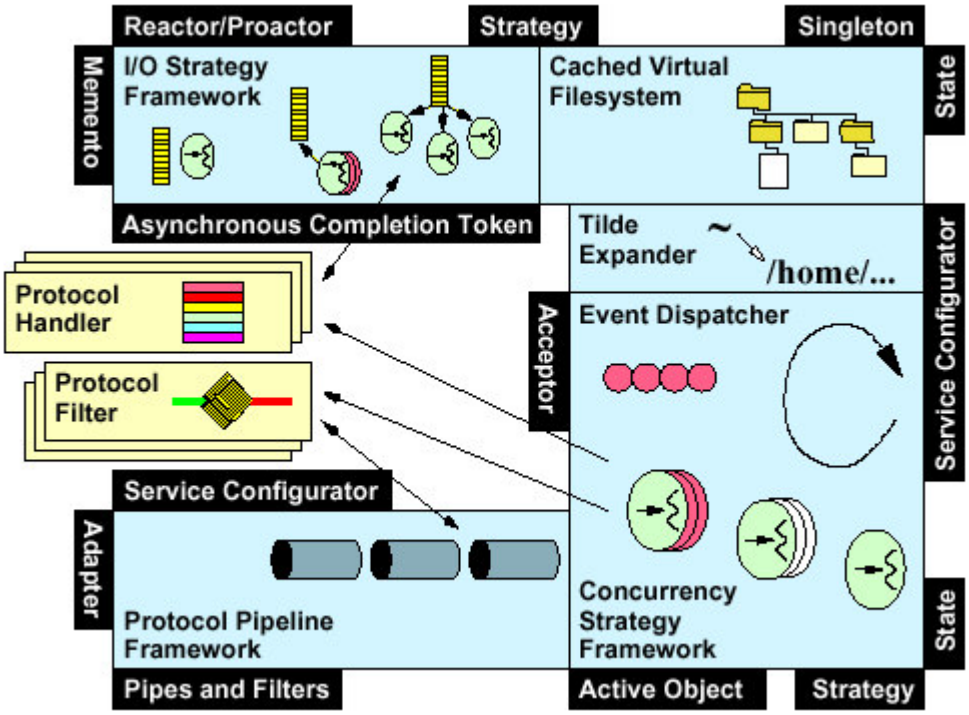
3.2.5. Kõrgema taseme vahetarkvara ja rakendused

Nendeks süsteemideks, mida üheskoos *ACE* 'ga arendatakse, on:

- *The ACE ORB (TAO)*: standartse *OMG CORBA* mudeli realisatsioon koos täiustustega, mis erinevalt enamikust tavalistest *ORB* 'idest võimaldavad *TAO* 'd kasutada ka suure jõudlusega ja reaalajarakendustes (joonis 5). Joonisel 3 asub *TAO* hajususega tegeleva vahetarkvara kihis.
- *JAWS*: suure jõudlusega adaptiivne veebiserver, mis on struktureeritud kui raamprogrammidest koosnev raamprogramm (joonis 6).



Joonis 5. *TAO* komponendid [7].



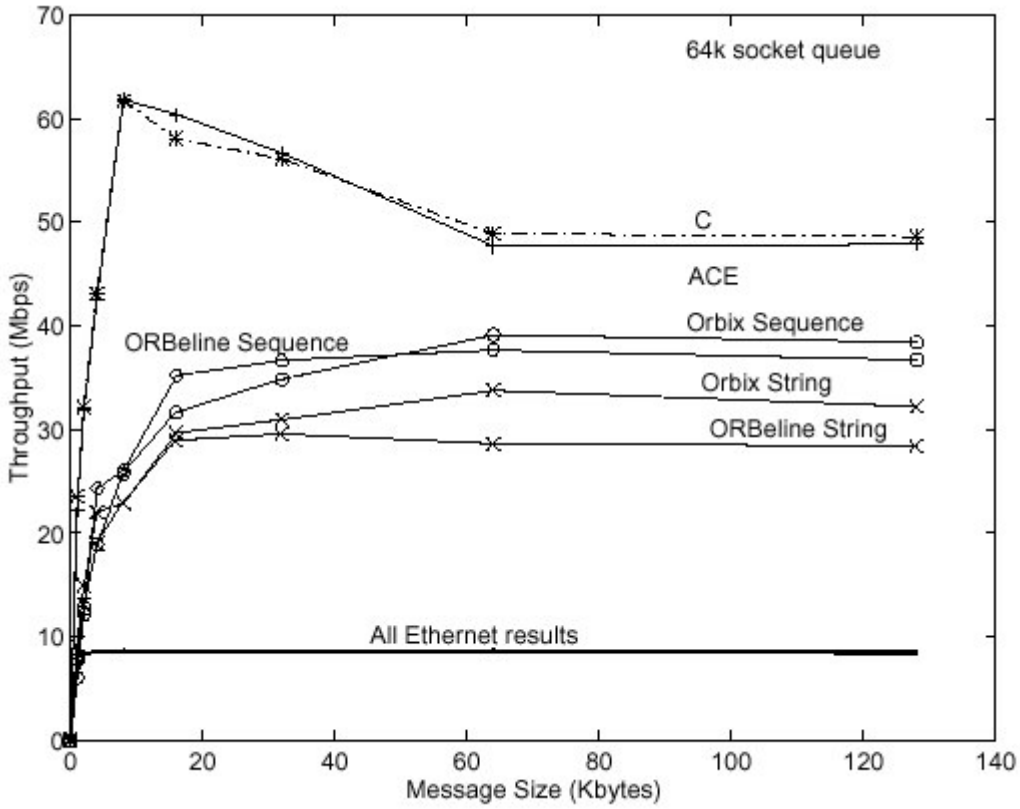
Joonis 6. Raamprogrammi *JAWS* arhitektuur [7].

3.3. *ACE*'i ja *TAO* jõudlus ning "jalajälg"

ACE'i loomisel on suurt tähelepanu pööratud sellele, et jõudluskadu võrreldes madaltaseme programmeerimisega oleks minimaalne. See on suures osas ka saavutatud (nt. kasutades punktis *3.2.2. "C++ ümbrised"* nimetatud *inlining*'ut), millest annavad tunnistust nii jõudlustestid (joonis 7) kui märkimisväärselt laialdane kasutajaskond, sealhulgas äärmiselt nõudlikes valdkondades nagu lennundus ja sõjandus. Mõned näited *(Boeing, Siemens)* leidusid juba eelnevas tekstis. Siia võiks lisada veel Türgi Mereväe poolt loodava lahinguhaldussüsteemi vahetarkvara, mille autorid vähemalt aastal 2000 kirjutasid, et esmane planeeritav samm süsteemi jõudluse tõstmiseks on selle portimine *The ACE ORB*'ile [9]. Samuti on positiivseks näiteks süsteem *DOTS* (*The Distributed Object-Oriented Threads System*, programmeerimisvahend ebaregulaarsete ja tugevalt andmetest sõltuvate algoritmide paralleliseerimiseks) kirjutamine *ACE*'le, samas kui tema eelkäija *DTS* töötas *PVM*'i (*Parallel Virtual Machine)* [10]. Üleminek *ACE*'le andis järgmised eelised:

- *ACE*'i teegid on lingitud, aga *PVM* käivitub tervikuna eraldi. Seega *ACE*'i kasutades jääb mittevajatav funktsionaalsus välja.
- *ACE* on porditav väga paljudele platvormidele.
- Kõrgem abstraktsioonitase, mis samas ei kahjusta oluliselt jõudlust.
- Tugevam muutujate tüübikontroll vähendab töö käigus tekkivate vigade arvu (s.t. paljud vead leitakse juba kompileerimisel) ning tõstab seega tarkvara kvaliteeti.

Kuigi *ACE*'i ja *TAO*'d kasutatakse edukalt ka paljudes sardsüsteemides, võivad nad (eriti *TAO*) mõningatel juhtudel siiski ebasobivaks osutuda. Nimelt, olles küllaltki universaalsed, omab neid kasutav programm hoolimata *ACE*'i lingitud iseloomust (mis võimaldab üleliigse funktsionaalsuse lõpp-programmist välja jätta) siiski suhteliselt suurt mahtu – "jalajälge". Eriti nõudlike ja piiratud ressurssidega sardsüsteemide korral võib see osutuda tõsiseks probleemiks, muutes seal *ACE*'i ja *TAO* kasutamise (vähemalt normaalkujul, ilma oluliste modifikatsioonideta) võimatuks [11, 12].



Joonis 7. *C, ACE* 'i ja kahe *CORBA* versiooni *Orbix* ja *ORBeline* võrdlus 155 Mbps ATM võrgus ja 10 Mbps Ethernet võrgus aastal ~1995. Täpsem info testi kohta on kättesaadav allikast [8]. NB! Testitud on *ACE* 'i (konkreetsemalt *C++* ümbriseid), mitte *TAO* 'd (sest seda ei olnud testi ajal veel valmis), ja lisaks võivad vaadeldud *ORB*'id vahepeal edasi arenenud olla, seega on mõttekam peatähelepanu pöörata võrdlusele *ACE vs. C*.

4. Kokkuvõte

Mustrite, raamprogrammide ja nendel põhineva vahetarkvara kasutamine võib muuta tarkvaraarendust oluliselt lihtsamaks, kiiremaks ja odavamaks. Käesolevas artiklis vaadeldud *The Adaptive Communication Environment (ACE)* ja temal põhinevad *The ACE ORB (TAO)* ning *JAWS* kinnitavad selle väite paikapidavust, lihtsustades oluliselt mitmesuguste hajussüsteemide valmistamist. Olles lisaks veel avatud lähtekoodiga ja tasuta kättesaadavad, on *ACE* ning *TAO* kogunud üsna suure kasutajaskonna üle kogu maailma. Täpsemat infot *ACE*'i kohta on huvilistel võimalik leida internetiaadressilt http://www.cs.wustl.edu/~schmidt/ACE.html

5. Kasutatud materjalid

ACM artiklid on kättesaadavad http://portal.acm.org/ kaudu (vajalik kasutusõiguse olemasolu, ligipääs olemas näiteks TTÜ arvutivõrgust).

1. Bootable Cluster CD. Paul Gray.
http://bccd.cs.uni.edu/

2. Patterns, Frameworks, and Middleware: Their Synergistic Relationships.
Douglas C. Schmidt, Frank Buschmann. IEEE/ACM International Conference on

Software Engineering, Portland, Oregon, May 3-10, 2003.
http://www.cs.wustl.edu/~schmidt/PDF/ICSE-03.pdf

3. The ADAPTIVE Communication Environment. Douglas C. Schmidt.
http://www.cs.wustl.edu/~schmidt/PDF/SUG-94.pdf

4. Pitfalls of Agent-Oriented Development.
Michael Wooldridge and Nicholas R. Jennings. K. P. Sycara and M. Wooldridge, editors: Agents ' 98: Proceedings of the Second International Conference on Autonomous Agents, ACM Press, May 1998.
http://www.csc.liv.ac.uk/~mjw/pubs/agents98.pdf

5. Adaptive middleware: Middleware for real-time and embedded systems.
Douglas C. Schmidt. Communications of the ACM, Volume 45 Issue 6, June 2002.

6. Siemens I Syngo.
http://www.syngo.com/

7. An Architectural Overview of the ACE Framework. Douglas C. Schmidt.
(This article will appear in a special issue of USENIX Login, 1998)
http://www.cs.wustl.edu/~schmidt/PDF/login.pdf

8. Object-Oriented Components for High-speed Networks Programming.
Douglas C. Schmidt, Tim Harrison, Ehab Al-Shaer. Proceedings of the 1st Conference on Object-Oriented Technologies and Systems (COOTS), USENIX , Monterey, CA, June, 1995.
http://www.cs.wustl.edu/~schmidt/PDF/COOTS-95.pdf

9. A Combat Management System Middleware Based on CORBA.
Aykut Kutlusan, Nadir Altmidort, Tufan Oruk, Alpay Duman. Proceedings of the International Symposium on Distributed Objects and Applications, September 21 - 23, 2000, Antwerp, Belgium. (kättesaadav ACM ja IEEE kaudu)

10. Parallel Direct Volume Rendering on PC Networks.
Michael Meissner, Tobias Hüttner, Wolfgang Blochinger, Andreas Weber.
http://www-sr.informatik.uni-tuebingen.de/~bloching/papers/pdpta98.ps

11. Transport Layer Abstraction in Event Channels for Embedded Systems.
Ravi Pratap M, Ron K. Cytron, David Sharp, Edward Pla. ACM SIGPLAN Notices, Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems, Volume 38 Issue 7, June 2003.

12. Footprint and Feature Management using Aspect-Oriented Programming Techniques. Frank Hunleth, Ron K. Cytron. ACM SIGPLAN Notices, Proceedings of the joint conference on Languages, compilers and tools for embedded systems: software and compilers for embedded systems, Volume 37 Issue 7, June 2002.

13. Building and Installing ACE and Its Auxiliary Libraries and Services.
http://www.cs.wustl.edu/~schmidt/ACE_wrappers/ACE-INSTALL.html

14. ACE+TAO Sponsors.
http://www.cs.wustl.edu/~schmidt/ACE-sponsors.html

Artikli autor: Taivo Lints, TTÜ magistrant
Artikli pealkiri inglise keeles: "ACE – a Framework for Creating Distributed Systems"