

Tallinna Tehnikaülikool

Taivo Lints

990849LASM

Mustrid, raamprogrammid, vahetarkvara ja

The Adaptive Communication Environment

Referaat aines "Paralleelarhitektuurid"

Õppejõud: Kalle Tammemäe

Tallinn 2004

Sisukord

1. SISSEJUHATUS.....	1
2. MUSTRID, RAAMPROGRAMMID, VAHETARKVARA.....	2
2.1. PROBLEEMID TARKVARA LOOMISEL.....	2
2.2. MUSTRID (<i>PATTERNS</i>).....	3
2.3. RAAMPROGRAMMID.....	4
2.4. VAHETARKVARA	5
3. THE ADAPTIVE COMMUNICATION ENVIRONMENT.....	7
3.1. MIS ON <i>ACE</i> ?.....	7
3.2. <i>ACE</i> 'I STRUKTUUR JA FUNKTSIONAALSUS	7
3.2.1. <i>Operatsioonisüsteemide "adapterkiht"</i>	7
3.2.2. <i>C++ ümbrised (wrappers)</i>	8
3.2.3. <i>Raamprogrammi komponendid</i>	8
3.2.4. <i>Iseseisvad (self-contained) hajusteenuste komponendid</i>	9
3.2.5. <i>Kõrgema taseme vahetarkvara komponendid</i>	9
3.3. <i>ACE</i> 'I JA TAO JÕUDLUS NING "JALAJÄLG"	10
4. KOKKUVÕTE.....	12
5. KASUTATUD MATERJALID.....	13
6. LISA 1: ACE ARENDUSGRUPID JA PLATVORMID [13]	15
7. LISA 2: ACE SPONSORID [14]	16

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 lihtsalt – piisab, kui võrku ühendatud arvutid käivitada vabalt kättesaadava klastritarkvaraga (nt. *Bootable Cluster CD* [1]), kuid kasutajarakenduste programmeerimine sellele klastrile on sellegipoolest vaevanõudev. Käesolevas referaadis 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 baseerub.

2. Mustrid, raamprogrammid, vahetarkvara

2.1. Probleemid tarkvara loomisel

Tarkvaraarendus on alati raske olnud, kuid varasematel aegadel on siiski võimalik olnud enamik programme valmis kirjutada nõ. nullist alustades – baseerudes 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 (näiteks *sockets* ja *TLI (Transport Layer Interface)*) ja taaskasutatavate teekide (näiteks *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.

Nende raskustest jagu saamiseks – kvaliteetse tarkvara loomiseks mõistliku ressursikuluga – on vaja edukate mudelite, disainide ja teostuste süsteematilist taaskasutust [2]. Kui süsteemipäratu-oportunistliku taaskasutuse korral arendajad lihtsalt kopeerivad olemasolevatest programmide koodilõike uutesse süsteemidesse, siis süsteematiline 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. Süsteematiline taaskasutus võimaldab ära jätta tüüpiliste tarkvarakomponentide kuluka ning vigaderohke taasavastamise-leiutamise ning taastestimise.

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

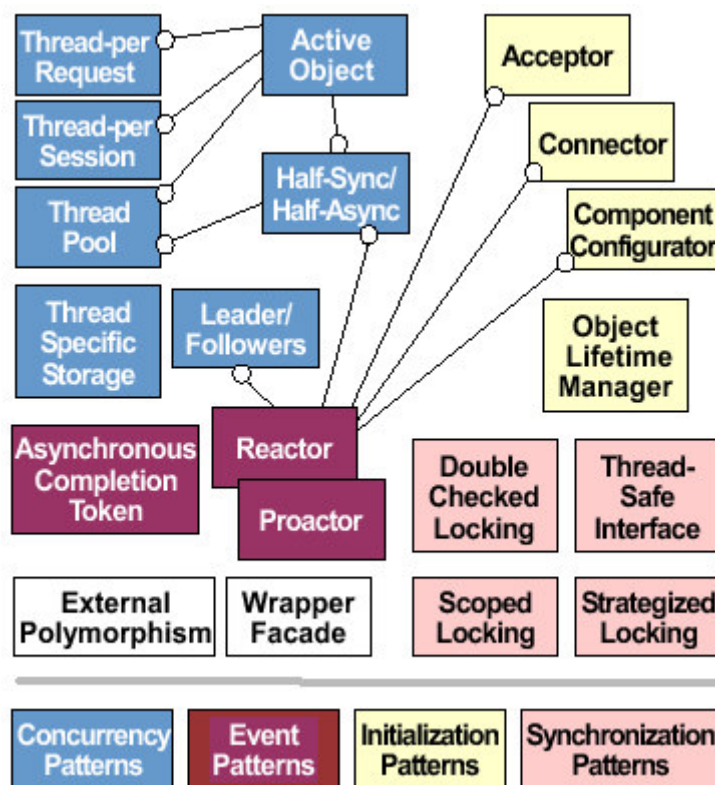
2.2. Mustrid (*Patterns*)

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 projekterijate kogemused jäävad dokumenteerimata, siis lähevad nad aja jooksul kaduma ja seega ei osale hilisemas tarkvara evolutsioonis.

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

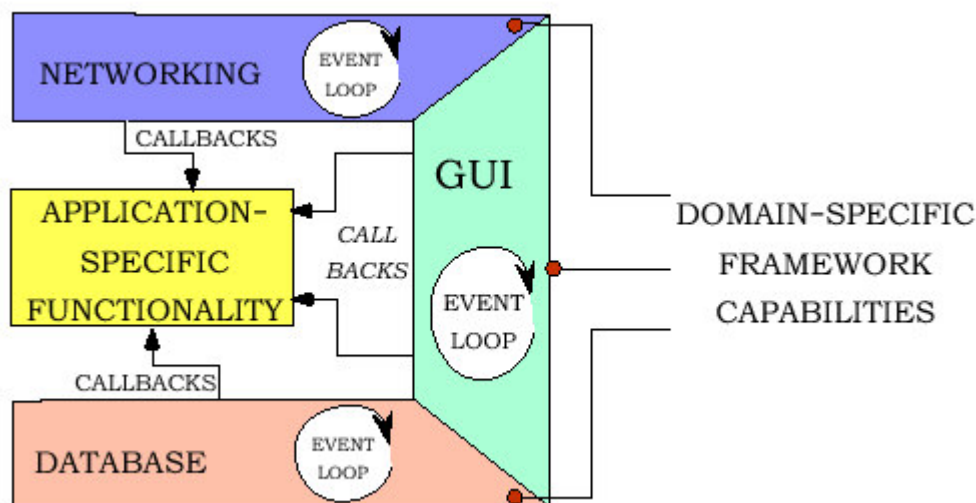
- Disainimustrid, mis kirjeldavad programmikomponentide sellist struktuuri ning omavahelisi suhteid, mis lahendab mõne konkreetse levinud disainiprobleemi.
- Arhitektuurimustrid, mis kirjeldavad tarkvarasüsteemi tervikstruktuuri ja pakuvad välja juhiseid selle terviku jaotamisel alamsüsteemideks.
- Mustrikeeled, 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. Mustrikeeled hajasprogrammeerimise jaoks [2]. Siinkohal on muidugi toodud vaid keele illustatsioon. Keele ise kirjeldab põhjalikult kõik mustrid ja seosed.

2.3. Raamprogrammid

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



Joonis 2. Raamprogrammi komponentide omavahelised suhted [2].

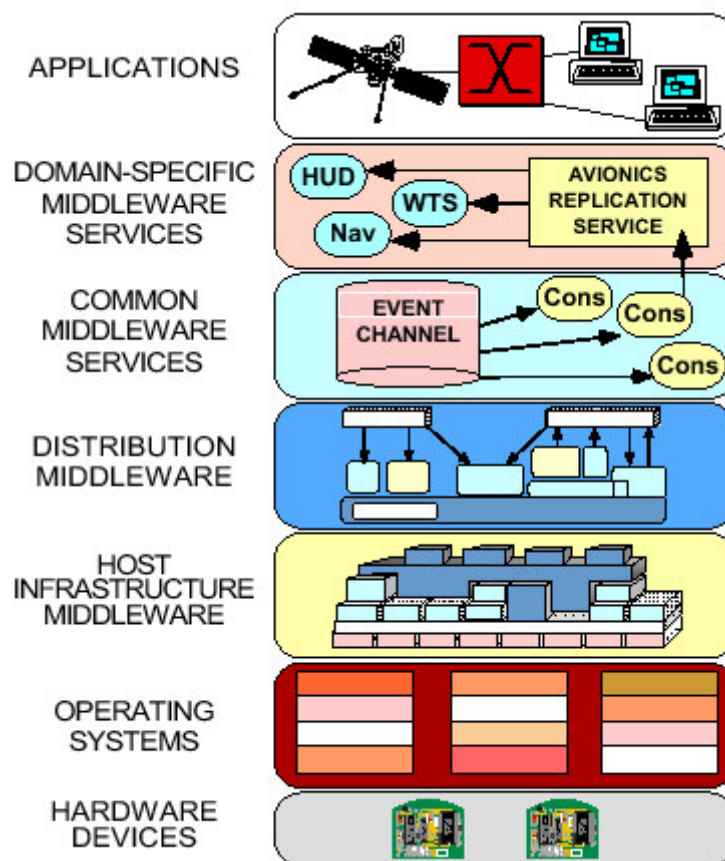
Sellisel raamprogrammil on järgnevad omadused [2]:

- "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 kutsus välja taaskasutatavaid teegi-funktsioone).
- 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 *Syngo* meditsiiniliste läbivalgustusaparaatide tarvis.

2.4. Vahetarkvara

Vahetarkvara ülesandeks on kaitsta programmeerijat hajussüsteemi madalama taseme probleemide eest, võimaldades enamuse ressursidest 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 "vertikaalseteks" kihtideks [2].

Vahetarkvara võib parema ülevaate ning mõistlikuma realiseerimise huvides jagada mitmeks kihiks (joonis 3) [2]:

- Arvuti infrastruktuuri vahetarkvara kapseldab ja parendab baasoperatsioonisüsteemi mehhanisme, luues näiteks taaskasutatavaid sündmuste demultipleksimise, protsessidevahelise suhtlemise, paralleelsuse-konkurentsuse ja sünkroniseerimisega tegelevaid objekte. Kapseldades konkreetsete operatsioonisüsteemide iseärasused, aitavad need objektid oluliselt vähendada tüütut ja vigadealdist madaltaseme programmeerimist ning suurendavad portitavust. 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 päringuvahen-

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

- Ühenduste loomine ja teenuste initsialiseerimine.
- Sündmuste demultipleksimine ja edastamine vastavate sündmuste teenindajatele.
- Protsessidevaheline suhtlemine ja jagatud mälu 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 ' 1 baseeruvaid kõrgema taseme vahetarkvarasid-komponente, nagu objektipäringuvahendajad (*Object Request Brokers*) ja veebiserverid.

ACE on avatud lähtekoodiga ning töötab paljudel erinevatel operatsioonisüsteemidel (vt. Lisa 1). Tema kasutajaskond on väga lai ning isegi sponsorite nimekiri on üsna aukartustäratav (vt. Lisa 2).

3.2. ACE 'i struktuur ja funktsionaalsus

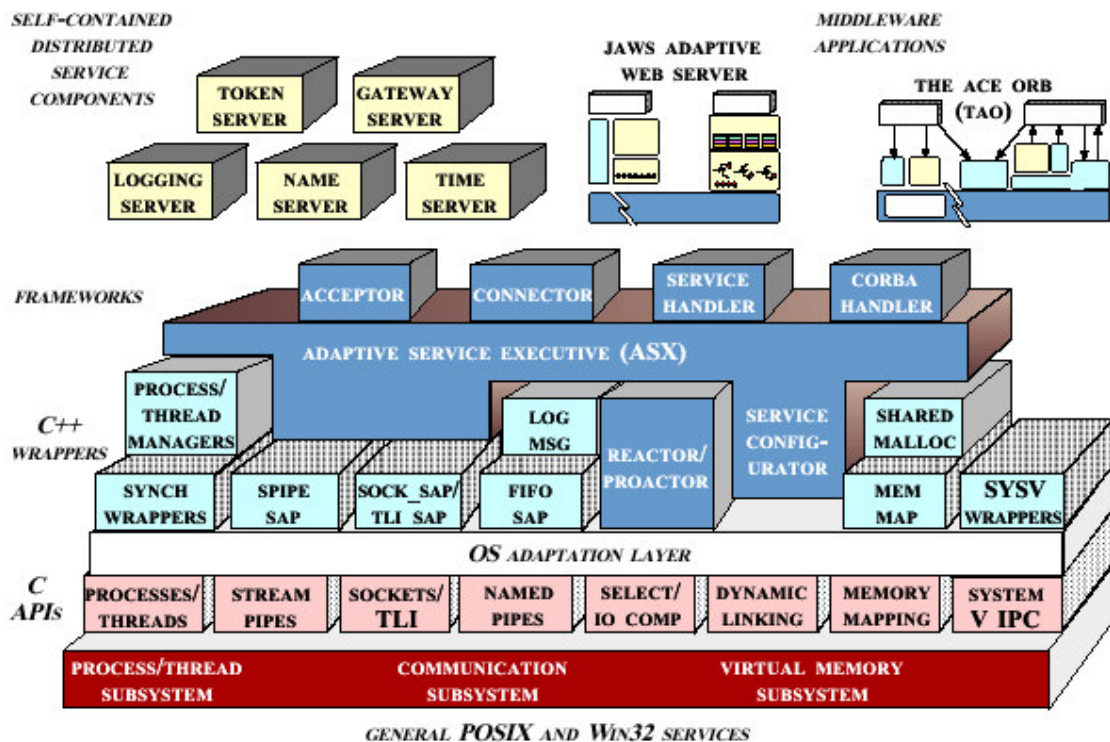
ACE raamprogramm sisaldab ~150 000 rida C++ koodi jagatuna ~450 klassiks. Erinevate probleemide üksteisest eraldamiseks ning keerukuse vähendamiseks on kasutatud kihulist arhitektuuri (joonis 4).

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, jagatud mälu.
- Sünkroonne ja asünkroonne sisendil/väljundil, taimeritel, signaalidel ja sünkronisatsioonil baseeruvate 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. Need 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 veaohlikke 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 rakenduspetsiifiliste komponentide väljakutsumisega.
- *Connector* ja *Acceptor*: eraldavad vastavalt aktiivse ja passiivse ühenduse initsialiseerimise rakenduspetsiifilistest 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. protokollikihtide (*protocol stack*)) realiseerimist.
- ORB adapter components: võimaldavad ACE ' i integreerida CORBA süsteemidesse.

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

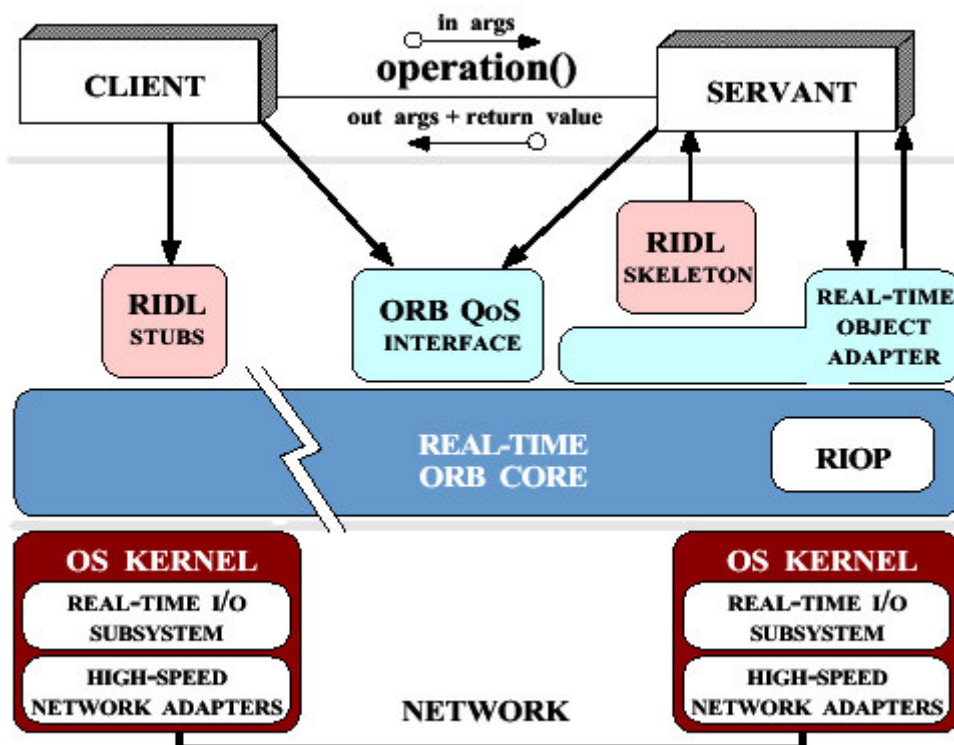
Need komponendid ei kuulu rangelt võttes ACE raamprogrammi, kuid sellegipoolest on neil kaks olulist rolli:

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

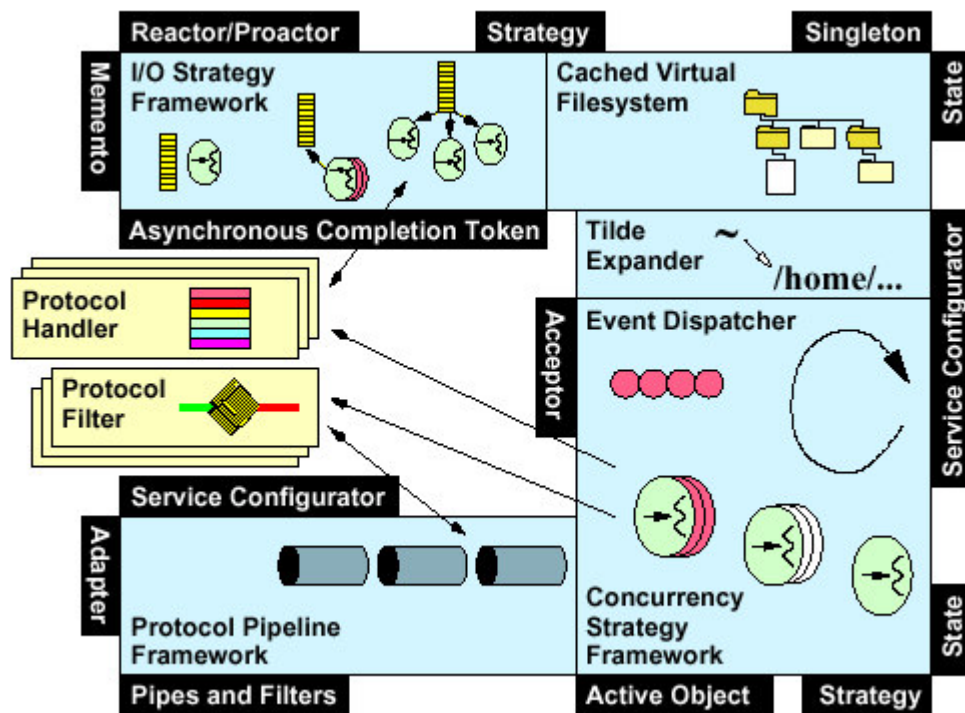
3.2.5. Kõrgema taseme vahetarkvara komponendid

Viivad abstraktsioonitaseme ACE raamprogrammist astme võrra kõrgemale. Nendeks komponentideks, mis ACE ' i lähtekoodiga alati kaasas käivad, on:

- The ACE ORB (TAO): standartse *OMG CORBA* mudeli realisatsioon koos täiustustega, mis erinevalt enamikust tavalistest ORB ' idest võimaldava TAO ' d kasutada ka suure jõudlusega ja reaalarakendustes (joonis 5).
- 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].

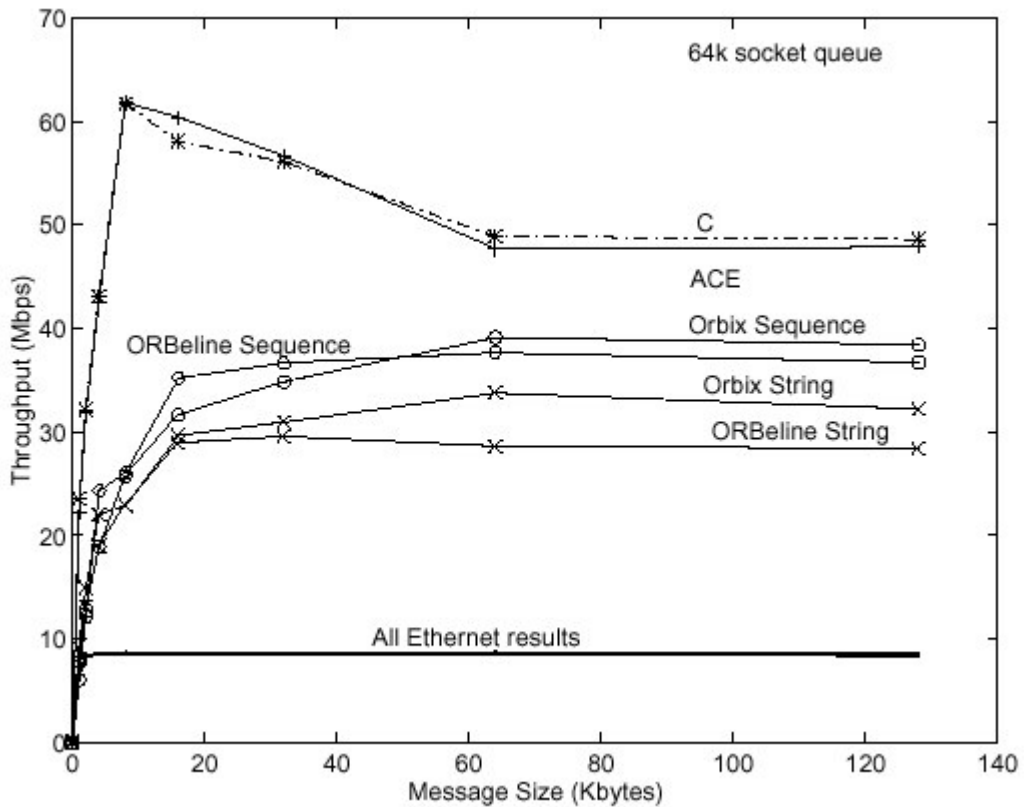
3.3. ACE'i ja TAO jõudlus ning "jalajälg"

ACE' i loomisel on suurt tähelepanu pööratud jõudluskao minimeerimisele võrreldes madaltaseme programmeerimisega. See on suures osas ka saavutatud (nt. kasutades punktis 4.2.2. nimetatud *inlining'* ut), millest annavad tunnistust nii jõudlusteid (vt. joonis 7) kui äärmiselt laialdane kasutajate ringkond, sealhulgas äärmiselt nõudlikes valdkondades nagu lennundus ja sõjandus. Mõned näited (*Boeing, Siemens*) leidsid 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 süsteemi *DOTS (The Distributed Object-Oriented Threads System)*, programmeerimiskeskond ebaregulaarsete ja tugevalt andmetest sõltuvate algoritmide paralleliseerimiseks) kirjutamine ACE' le, samas kui tema eelkäija *DTS* baseerus *Parallel Virtual Machine'* l (*PVM*) [10]. Üleminek ACE' le andis järgmised eelised:

- ACE' teegid on lingitud, aga PVM käivitub eraldi (vähemalt 1998 aastal käivitus). Seega ACE' i kasutades jääb mittevajutatav funktsionaalsus välja.
- ACE on portitav 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 osutada. Nimelt, olles küllaltki universaalsed, omab neid kasutatav 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 osutada

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. Täpsem info testi kohta kättesaadav allikast [8]. NB! Testitud on *ACE* ' i, mitte *TAO* ' d (sest viimast ei olnud testi ajal veel valmis), 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 referaadis vaadeldud *The Adaptive Communication Environment (ACE)* ja tema lisakomponendid *The ACE ORB (TAO)* ning *JAWS* demonstreerivad selle väite õigsust, lihtsustades oluliselt mitmesuguste hajussüsteemide valmistamist ja omades seetõttu suurt kasutajaskonda üle kogu maailma.

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 | 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/~blochinger/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>

6. Lisa 1: ACE arendusgrupid ja platvormid [13]

Group	Platforms
<i>DOC Group</i>	Tested and used daily by the DOC group: Solaris 2.6, 7 and 8 (many compilers excluding SunC++ 4.x) Windows NT 4.0, 2000, XP (MSVC++ 6.x, and 7.x, and Borland C++ Builder 5.0) Linux/Intel (many compilers), Linux/IA64 (GCC).
<i>Riverace</i>	Offers support services for many platforms including those above, HP-UX, AIX, and Windows CE.
<i>OCI</i>	Maintains ACE on certain platforms required for their TAO software and service offerings.
<i>Remedy IT</i>	Maintains ACE on certain platforms required for their ACE and TAO service offerings.
<i>ACE user community</i>	Responsible for continued maintenance and testing of platforms to which ACE has been ported, but aren' t supported by the above groups. These include Windows 95/98 using Borland C++ Builder 4.0 and later, and GNU g++ on MinGW and Cygwin; Digital UNIX (Compaq Tru64) 4.0 and 5.0; IRIX 6.x; UnixWare 7.1.0; SunOS 4.x and Solaris with 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 and Neutrino 2.0; VxWorks; LynxOS; RTEMS

7. Lisa 2: ACE sponsorid [14]

The following companies, organizations, and funding agencies have sponsored our work on ACE and TAO over the past decade.

- * ATD
- * BBN
- * Boeing
- * CDI/GDIS
- * Cisco
- * Comverse
- * DARPA
- * Ericsson
- * Experian
- * Global MAINTECH
- * Hughes Network Systems
- * Kodak
- * Kronos
- * 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

We are very grateful for their support and collaboration.

Douglas C. Schmidt