UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Janez Golob

Možnosti porazdeljene izvedbe kontrolnega sistema

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentorica: doc. dr. Mojca Ciglarič

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00442/2009
Datum: 05.04.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JANEZ GOLOB**

Naslov: **MOŽNOSTI PORAZDELJENE IZVEDBE KONTROLNEGA SISTEMA**
DISTRIBUTED CONTROL SYSTEM: IMPLEMENTATION OPTIONS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Preučite tehnologijo in delovanje izbranega kontrolnega sistema. Analizirajte funkcionalne in ostale zahteve, ki jih mora podpirati vmesna programska rešitev, da bi bil na njej lahko realiziran kontrolni sistem. Identificirajte izvedbe vmesne programske opreme, ki bi lahko zadoščale tem zahtevam in zasnujte metodologijo za njihovo preizkušanje. Rešitve primerjajte teoretično in praktično v testnem okolju ter na podlagi rezultatov izberite najprimernejšo za implementacijo kontrolnega sistema. Izbiro utemeljite.

Mentor:
M. Ciglarič
doc. dr. Mojca Ciglarič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Franc Solina

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Janez Golob,

z vpisno številko 63040040,

sem avtor diplomskega dela z naslovom:

Možnosti porazdeljene izvedbe kontrolnega sistema

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  doc. dr. Mojce Ciglarič
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 20.9.2009 Podpis avtorja:

Zahvala

Na tem mestu bi se zahvalil vsem, ki so kakorkoli pripomogli k uspešni izvedbi diplomskega dela.

Urški Majcen

Kazalo

Seznam uporabljenih kratic in simbolov

API Application Programming Interface
ITER International Thermonuclear Experimental Reactor
CODAC COntrol, Data Access and Communication
PON Plant Operation Network
PCN Plant Commissioning Network
OPN Open Public Network
TCN Time Cummunication Network
SDN Synchronous Data-bus Network
EDN Event Distribution Network
QOS Quality Of Service
EPICS Experimental Physics and Industrial Control System
CA Channel Access
CORBA Common Object Request Broker Architecture
DDS Data Distribution Service
CAC Channel Access Client
CAS Channel Access Server
CCM CORBA Component Model
ORB Object Request Broker
OMG Object Management Group
GIOP General Inter-ORB Protocol
IIOP Internet Inter-ORB Protocol
ADAPTIVE Dynamically Assembled Protocol Transformation and Validation Environment
ACE ADAPTIVE Communication Environment
DCPS Data-centric Publish-Subscribe
DLRL Data Local Reconstruction Layer
RPC Remote Procedure Call
AMI Asynchronous Method Invocation
AMD Asynchronous Method Dispatch
ACS Alma Control System

Povzetek

Namen tega diplomskega dela je preučiti možnost izvedbe porazdeljenega kontrolnega sistema. Zaradi tega je bilo potrebno na konkretnem primeru pridobiti in analizirati tako funkcionalne kot nefunkcionalne zahteve, ki jih mora podpirati vmesna programska oprema. V nadaljevanju sem najprej s pomočjo dostopne literature, kontaktov izdelovalcev in s pregledom izvorne kode ugotovil, če in v kolikšni meri zadostuje vsaka izmed vmesnih programskih oprem nefunkcionalnim zahtevam. Nato sem za vsako vmesno programsko opremo razvil enostaven kontrolni sistem, ga preizkusil na testnem okolju ter izmeril zmogljivost.

S pomočjo rezultatov meritev ter študije izpolnjevanja zahtev sem nato primerjal vmesne programske opreme in jih razvrstil od najslabše do najboljše. Izkazalo se je, da nobena rešitev ne izpolnjuje vseh zahtev v celoti in da ima vsaka tako dobre kot slabe lastnosti. RTI DDS je naprimer najzmogljivejša vmesna programska oprema, vendar ne vsebuje orodij, potrebnih za gradnjo kontrolnih sistemov. Kot najboljši možni kompromis med zmogljivostjo, skalabilnostjo, številom orodij itd. se je izkazala vmesna programska oprema EPICS.

Ključne besede:

Vmesna programska oprema, kontrolni sistemi, ITER, EPICS, CORBA, DDS, ZeroC Ice, primerjava zmogljivosti.

Abstract

The purpose of this bachelor work is to examine the possibility of implementing a distributed control system. In this case it was necessary to obtain and analyze functional as well as non-functional requirements that must be supported by middleware. Firstly I determined if and to what extent were each of middleware software non-functional requirements sufficient, by searching through available literature, contacts with manufacturers and the source code reviews. Then I developed a simple control system for each middleware software, tested it on a test environment and measured performance.

Using the results of measurements and requirements study, I compared the middleware softwares and categorized them from worst to best. It turned out that no of the investigated technologies is a perfect fit for the requirements and that each had advantages and drawbacks. RTI DDS for example is the most powerful middleware software, but does not offer any control system specific tools and services. As the best compromise between capacity, scalability, the number of tools and services, etc. has proven EPICS.

Key words:

Middleware, control system, ITER, EPICS, CORBA, DDS, Zeroc Ice, performance comparison.

Uvod

Veliki fizikalni projekti za delovanje in vodenje eksperimentov že dolgo uporabljajo takšne ali drugačne kontrolne sisteme. Prvi kontrolni sistemi so bili zgrajeni iz analognih komponent (žica, analogni senzorji, prikazovalniki, gumbi, ...). Analogni senzor je zaznal spremembo in ta se je kot analogni signal prenesla do prav tako analognega prikazovalnika. Operater je lahko s podobnim konceptom kontroliral naprave.

Z vse večjimi eksperimenti in razvojem tehnologije so se pričele stvari precej zapletati, koncept pa je ostal isti. Analogni senzorji so se počasi nadomestili z digitalnimi, prikazovalniki in števci so se nadomestili z monitorji, žico je zamenjalo takšno ali drugačno omrežje in vmesna programska oprema. Hkrati se je pričelo tudi razslojevanje odgovornosti posameznega dela kontrolnega sistema in abstrakcija posameznih delov.

Vsak kontrolni sistem deluje nad enim ali več vrstami omrežij (žica), ki zagotavljajo prenos podatkov. Vendar so ta omrežja gledana z vidika kontrolnega sistema praviloma skrita pod vmesno programsko opremo, ki zagotavlja skupno podlago za gradnjo višjenivojskih aplikacij. Z drugimi besedami: vmesna programska oprema je prenašalec sporočil med dvema točkama.

V diplomskem delu bom najprej opravil analizo zahtev kontrolnega sistema z vidika vmesne programske opreme. Nato bom podrobneje predstavil najbolj obetavne vmesne programske rešitve, ki se bodo v nadaljevanju ocenile z merili, določenimi v analizi zahtev. Prav tako bom predstavil prototip kontrolnega sistema oziroma testno okolje, na katerem sem testiral posamezno vmesno programsko opremo, in v nadaljevanju prikazal rezultate, pridobljene na testnem okolju.

Cilj diplomskega dela je prototipno preizkusiti na trgu prisotne vmesne programske opreme ter določiti najboljšo za gradnjo porazdeljenih kontrolnih sistemov.

Motivacija

Motivacija za to diplomsko nalogo je bilo študentsko delo na podjetju Cosylab, kjer sem se tudi prvič srečal s svetom velikih fizikalnih projektov. Cosylab je namreč vodilno svetovno podjetje na področju integracije in izgradnje kontrolnih sistemov. Ena od nalog, pri kateri sem sodeloval, je bila tudi študija vmesnih programskih oprem oziroma sistemov, zgrajenih na njihovi podlagi. Znanje, pridobljeno s tem delom, sem kasneje še razširil in tako pridobil znanje in ključne podatke za to diplomsko nalogo.

Poglavje 1

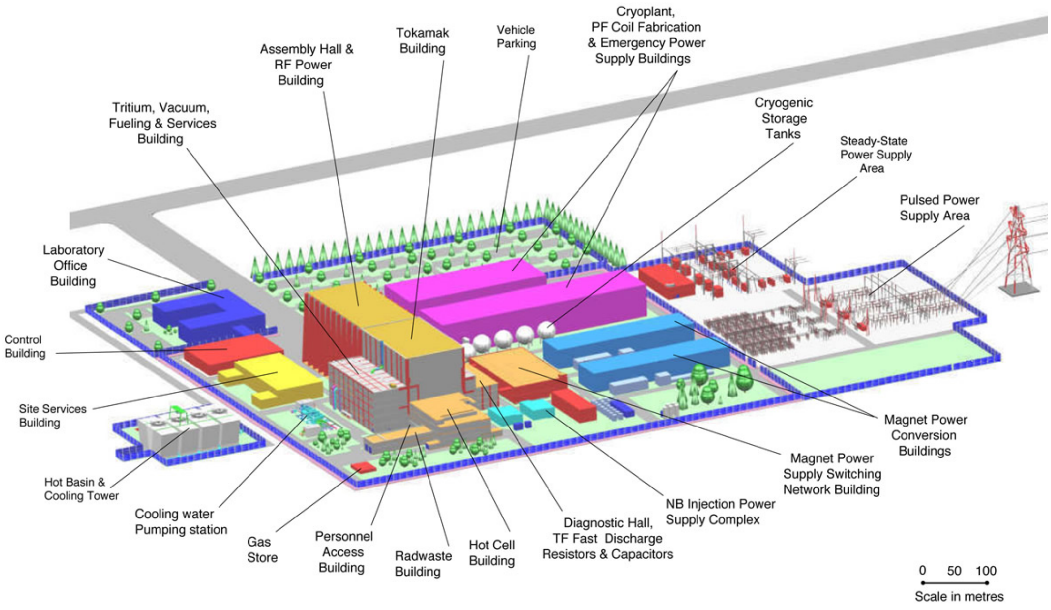
Kontrolni sistem

Če hočemo dobro razumeti naloge in pomen vmesne programske opreme, moramo vsaj površno razumeti zgradbo in namen kontrolnega sistema, ki ga bom v tem poglavju pregledno opisal na primeru ITER kontrolnega sistema. Kaj je kontrolni sistem? Vzemimo za primer osebo Urško, ki se vozi z osebnim avtomobilom iz Ljubljane v Maribor. Samoumevno je, da z gibi tako ali drugače upravlja avtomobil, torej ga kontrolira. Urška se niti ne zaveda, da ji vožnjo v vsakem trenutku olajšuje precejšnje število naprav.

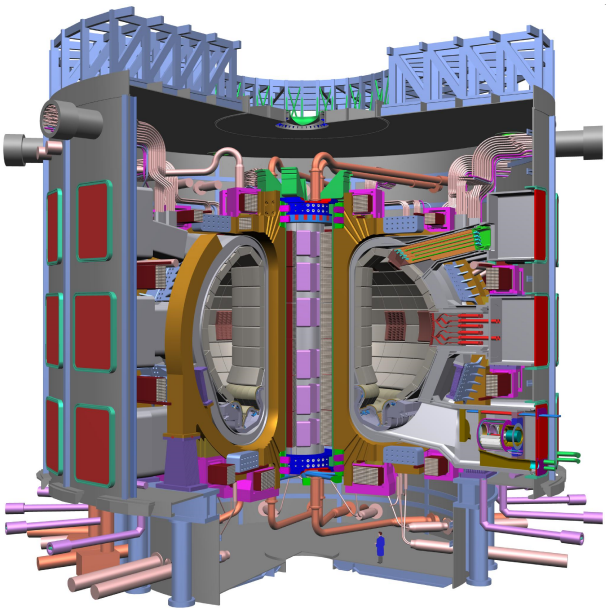
Kontrolni sistem bi lahko opredelili kot sestav naprav, senzorjev, računalnikov, merilcev, ki tako ali drugače zagotavljajo, da neka naprava opravi določeno nalogo. Ampak če hočemo resnično doumeti obsežnost velikih fizikalni projektov, si moramo zadevo malce pobližje ogledati – dober primer velikega fizikalnega projekta je vsekakor ITER. Gre za poizkus, s katerim hočejo dokazati tehnološko zmožnost izdelave fuzijskega jedrskega reaktorja. Sama zgradba se bo nahajala v Franciji 1.1. Predviden proračun za izgradnjo znaša 100 milijard evrov, vendar obstajajo špekulacije, da bo celoten znesek krepko presegel ceno mednarodne vesoljske postaje (znašala je 120 milijard evrov in je do sedaj najdražji znanstvenoraziskovalni projekt). Izgradnja ITER-ja naj bi trajala predvidoma 10 let, njegovo obratovanje pa še nadaljnjih 20 let.

Ker so to res nepredstavljive vsote, je morda najbolje, da pogledamo samo velikost. Slika 1.2 prikazuje samo glavni del objekta ITER, in sicer fuzijski reaktor. Njegov premer je približno 15 metrov, višina pa 20 metrov.

Namen gradnje objekta je poskus pridobivanja čiste električne energije. Sam reaktor bo vseboval ogromno senzorjev, naprav za dovajanje in odvzemanje različnih snovi, odjemalnikov toplote, veliko število magnetov, ki bodo držali 100 000 000 K vročo snov stran od zaščitnih odej, ... Vse to bo potrebno tudi izredno natančno krmiliti.
Kontrolni sistem, ki bo neprestano kontroliral celotno delovanje reaktorja in z njim povezanih naprav, se imenuje CODAC (ang. Command, Control, Data Acquisition and Communication) [1]. Nameščen bo med reaktorjem in kontrolno sobo in ga bo povezovalo več omrežij. Vendar je za visokonivojske aplikacije, ki morajo pošiljati sporočila od senzorja do kontrolne sobe ali obratno, nepomembno, kakšna je tehnološka rešitev omrežja. Zaradi tega se bo dodala vmesna programska oprema, ki bo namenjena izključno abstrak-



Slika 1.1: Prikaz objekta ITER iz severo-zahoda. (Vir: www.iter.org)

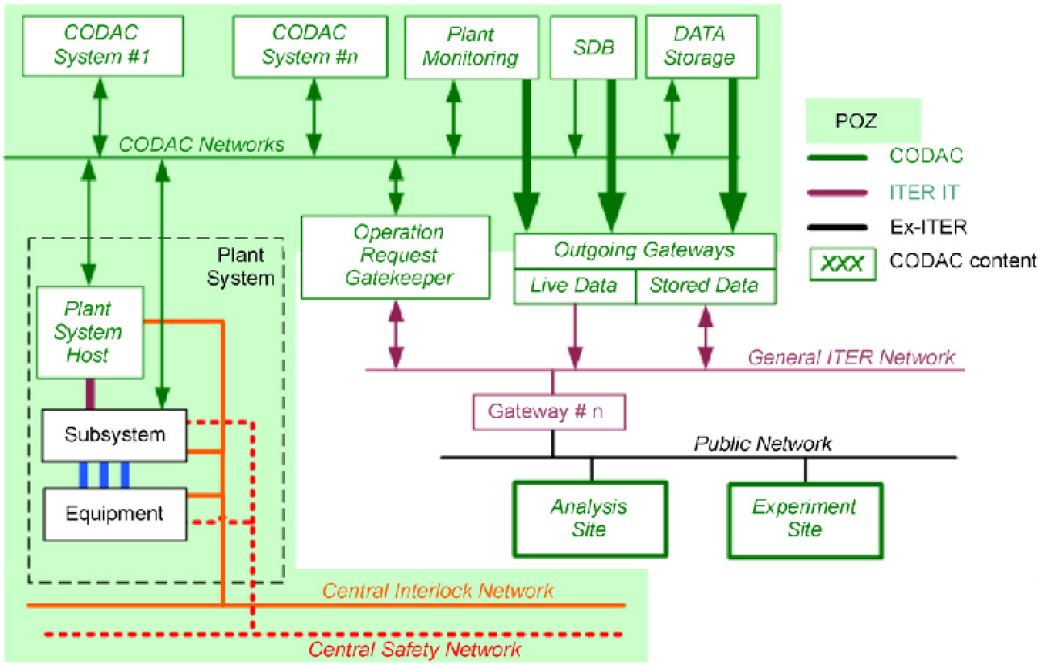


Slika 1.2: Shema fuzijskega reaktorja. (Vir: www.iter.org)

ciji prenosa podatkov – komunikaciji.

1.1 CODAC omrežje in PON omrežje

Kot sem že omenil, je CODAC namenjen neprestanemu spremljanju posameznih podsistemov (v nadaljevanju bom uporabljal besedno zvezo plant sistem): prikazovanju statusa (alarmi), proženju operacij in pridobivanju podatkov iz podsistemov. Slika 1.3 prikazuje konceptualni dizajn CODAC omrežja.



Slika 1.3: Shema CODAC omrežja (Vir: www.iter.org).

CODAC omrežje je zgrajeno iz več namensko ločenih omrežij, ki se ločijo po namenu in tudi zahtevah [4].

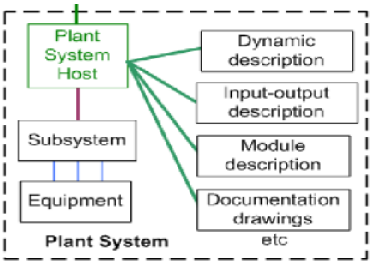
1. Asinhrona omrežja:
   (a) PON (ang. Plant Operation Network) – standardna komunikacija med plant sistemi, namenjena spremljanju, pridobivanju podatkov itd. To omrežje bo glavni del CODAC-a, kar se tiče kontrolnega sistema, zato se bom v diplomskem delu posvetil predvsem temu omrežju.
   (b) PCN (ang. Plant Commissioning Network) – nadzorno omrežje.
   (c) OPN (ang. Open Public Network) – omrežje, namenjeno komunikaciji med oddaljenimi kontrolnimi sobami (ena izmed njih se bo nahajala na Japonskem).

2. TCN (ang. Time Cummunication Network) – omrežje, namenjeno časovni sinhronizaciji.

3. SDN (ang. Synchronous Data-bus Network) – omrežje, ki zagotavlja sinhron prenos podatkov in deterministično zakasnitev odziva.

4. EDN (ang. Event Distribution Network) – omrežje za prenos dogodkov oziroma za proženje operacij (zahtevek po izbedbi operacije). To omrežje je podobno omrežju SDN s to razliko, da zahteva nižjo zakasnitev.

Na tem mestu je smiselno razložiti tudi pojem plant sistema. Kot prikazuje slika 1.4, je plant sistem abstrakcija nekega zaokroženega podsistema, ki komunicira z ostalimi podsistemi preko standardiziranega vmesnika. Pravzapra je abstrakcija neke naprave oziroma skupek naprav. Te naprave so krmiljene in nadzorovane s pomočjo prej omenjenega programskega vmesnika.



Slika 1.4: Konceptualna shema plant sistema (Vir: www.iter.org).

Pomembno je tudi to, da mora vsak plant sistem vsebovati lasten opis, seznam vhodov in izhodov ter dokumentacijo. To je pomembno predvsem takrat, ko se pojavijo težave v sistemu (okvara, težave pri integraciji). Potrebno se je zavedati, da so posamezni plant sistemi zgrajeni v različnih državah in da je glede na veliko število plant sistemov ta način najbolj optimalen za integracijo in vzdrževanje. Tako ima vzdrževalec/integrator na voljo dokumentacijo z vsemi načrti in kontakti proizvajalca, ki je dolžan nuditi podporo v primeru napak.

Vmesna programska oprema je izredno pomembna za vsak obsežnejši kontrolni sistem. S pomočjo pravilne izbire lahko precej zmanjšamo tako stroške gradnje, kot tudi stroške vzdrževanja. V kontrolnem sistemu, ki vsebuje veliko različnih omrežij, je izredno težko komunicirati z različnimi aplikacijami. Z uvedbo vmesne plasti pa omogočimo transparentno komunikacijo in s tem zmanjšamo količino programske kode, potrebne za dosego funkcionalnosti kontrolnega sistema. Velik pomen leži tudi v tem, da se lahko inženir/programer posveti reševanju jedra problema in ne obrobnih problemov, kot so komunikacija, povezovanje ...
Na koncu gradnje je seveda pomembno, da kontrolni sistem lepo deluje kot celota in omogoča kontrolorjem nemoteno in učinkovito delo z eksperimentom.

Poglavje 2

Analiza zahtev

V tem poglavju bom semiformalno opisal primere uporabe in z njimi povezane nefunkcionalne zahteve, ki se uporabljajo za vmesne programske rešitve. Primeri bodo določali merila, na podlagi katerih bom primerjal komunikacijske tehnologije.

Primeri uporabe opisujejo interakcijo med primarnim akterjem (pobudnik interakcije) in samim sistemom, predstavljeno kot zaporedje preprostih korakov. S tem določajo odgovornost sistema do drugih sistemov.

Kjer je mogoče, zahteve posameznega primera uporabe kvantificirajo količino podatkov in število subjektov, ki sodelujejo v kontrolnem sistemu. Primere uporabe bom izvzel iz kontrolnega sistema CODAC oziroma natančneje iz omrežja PON [3]. Najdemo jih lahko v vseh podobnih kontrolnih sistemih.

2.1 Akterji

Povzeto po spletni enciklopediji [20] je akter nekdo ali nekaj zunaj sistema, ki ali deluje na sistem (primarni akter) ali pa sistem deljuje nanj (sekundarni akter). Akter je lahko oseba, naprava, drug sistem ali podsistem. Kot bomo videli v nadaljevanju na posameznih primerih, lahko posamezni akter igra več vlog.

V nadaljevanju bom naštel in opisal posamezne akterje.

2.1.1 Prožilec ukaza

Prožilec ukaza je subjekt, ki poda ukaz.

Prožilec ukaza je lahko uporabniški vmesnik v glavni kontrolni sobi ali proces na enem od centralnih CODAC-računalnikov, na primer CODAC servis (Plant Monitoring, Operation Scheduling, Operation Request Gatekeeper, ...) ali drugi specifični procesi.

Prisotno je okoli 150 prožilcev ukazov (50 na centralnih računalnikih in 100 v kontrolni sobi).

2.1.2 Izvršitelj ukaza

Izvršitelj ukaza je subjekt, ki po prejemu ukaza izvede ukaz. Nato pošlje nazaj izhod ukaza, ki je lahko rezultat, informacija o napaki itd.

Izvršitelj ukaza je bodisi del plant sistema ali proces na enem od centralnih CODAC-strežnikiov.

V sistemu CODAC je približno 200 izvršiteljev ukaza (150 v plant sistemiih in 50 na osrednjem strežniku).

Izvršitelj ukaza lahko hkrati igra tudi vlogo prožilca ukaza (npr. če mora delegirati podopravilo drugemu izvršitelju ukaza ali nazaj do izvršitelja ukaza, če je uporabljen mehanizem povratnih klicev za prenos informacij).

Proženje ukazov je lahko asinhrono.

2.1.3 Vir dogodka

Vir dogodka je entiteta, ki objavlja (pošilja) dogodke. Dogodek vsebuje poljubne podatke in informacije o prejemnikih (ponor dogodka), ki jih lahko interpretira. Nekateri podsistemi (npr. logiranje in arhiviranje) lahko uporabljajo dogodke za distribucijo podatkov. Dogodki so lahko koristni za distribuirano sinhronizacijo dejavnosti, ki ne potrebujejo izpolnjevati zahteve realnega časa.

Sistem CODAC vsebuje približno 200 virov dogodkov.

2.1.4 Ponor dogodka

Ponor dogodka sprejme dogodek z namenom, da se izvede ukrep (npr. persistenca podatkov, izvedba izračuna itd.) Ponor dogodka lahko prevzame vlogo vira dogodka.

V sistemu CODAC je približno 500 ponorov dogodkov.

2.1.5 Izvor spremljanja

Izvor spremljanja objavlja dogodke, ki vsebujejo vrednost spremljane spremenljivke.

CODAC-sistem ima približno 500 izvorov spremljanja.

2.1.6 Ponor spremljanja

Ponor spremljanja sprejema dogodke, objavljene s strani izvora spremljanja. Nato prezentira vrednost uporabniku (preko grafičnega vmesnika) ali jo arhivira.

V sistemu CODAC je predvidoma 500 ponorov spremljanja.

2.1.7 Vzdrževalec Sistema

Vzdrževalec sistema je oseba ali avtomatiziran proces, odgovoren za preverjanje stanja sistema.

Sistem CODAC vsebuje približno 5 vzdrževalcev sistema.

2.1.8 Izvor alarma

Izvor alarma je subjekt, ki lahko ugotovi stanje izven običajnih meja in o njem poroča.

Sistem vsebuje približno 500 izvorov alarmov.

2.1.9 Upravljavec alarmov

Upravljavec alarmov prejema obvestila, ki jih prožijo izvori alarmov, in opravlja dejavnosti, kot so obveščanje uporabnikov o temeljnih vzrokih (na podlagi stanja virov) in v nekaterih primerih tudi avtomatizirano izvajanje ukrepov za odpravo alarmnih stanj.

V sistemu je približno 5 upravljavcev alarmov.

2.1.10 Izvor podatkov

Izvor podatkov neprestano ustvarja podatke in jih daje na razpolago ponorom podatkov.

Vsak plant sistem ima približno 10 virov podatkov (prisotnih je približno 150 plant sistemov).

2.1.11 Ponor podatkov

Vsak plant sistem vsebuje približno 10 ponorov podatkov.

2.2 Primeri uporabe za omrežje PON

V tem poglavju bom podrobno določil zahteve, ki so predivene za delovanje kontrolnega sistema CODAC ali natančneje za omrežje PON (ang. Plant Operation Network), ki predstavlja velik del kontrolnega sistema. Te zahteve bi lahko z lahkoto prenesli na večino kontrolnih sistemov – največje razlike bi bile v številu posameznih akterjev.

2.2.1 Splošne nefunkcionalne zahteve

V tem podpoglavju so opisane nefunkcionalne zahteve, ki se lahko aplicirajo na enega ali več primerov uporabe. Zahteve so povzete po CODAC-ovem konceptualnem pregledu [1].

- Zakasnitev vmesne programske opreme: znana mora biti v terminih, kot so povprečje in 95 percentil.
- Skalabilnost: sprejemljivo je, da hitrost prenosa pada (gledano z strani sprejemnika), če se število sprejemnikov povečuje, vendar se skupna hitrost prenosa ne sme pretirano zmanjšati.

- Ne sme biti kritične točke izvajanja ali/in ozkega grla: če obstaja servis, ki skrbi za distribuiranje sporočil/ukazov, mora obstajati tudi možnost njegove zamenjave.
- Podpora zanesljivemu prenosu: obstajati mora način zanesljivega prenosa podatkov.
- Podpora prenosu po najboljših močeh.
- Možnost avtentikacije: sprejemnik lahko določi identiteto pošiljatelja z avtentikacijo. S to informacijo lahko v nadaljevanju zavrne ali sprejme sporočilo.
- FIFO dostava podatkov: vrstni red sprejetih podatkov mora biti enak vrstnemu redu poslanih podatkov.
- Skupni vrstni red: v primeru večih sprejemnikov mora vsak prejeti podatke v enakem vrstnem redu, kot so bili poslani.
- Časovno žigosanje: za vsako sporočilo mora obstajat način, s katerim se da ugotoviti čas, ob katerem je bilo sporočilo poslano.
- Diagnosticiranje: razvijalec/inženir mora imeti možnost pregleda poslanih podatkov skozi vmesno programsko opremo.
- Kontrola pretoka podatkov: hitrost prenosa podatkov med dvema točkama mora biti izbrana tako, da prepreči izgubo podatkov (preplavitev sprejemnika).
- Stiskanje podatkov.
- Medpomnjenje podatkov.
- Podpora večim platformam.
- Podpora programskim jezikom.

2.2.2 Izvedba ukaza

Ta primer uporabe je ekvivalenten oddaljenemu proženju ukaza (ang. Remote Command Invocation). V naslednjih točkah bom natančneje opisal korake, s katerimi se ta primer uporabe realizira.

1. Prožilec ukaza pošlje ukaz izvršitelju ukaza. Prožilec ukaza mora znati:
   (a) identificirati izvršitelja ukaza, ki bo izvedel ukaz (ukaz je vedno namenjen samo enemu prejemniku);
   (b) identificirati ukaz, ki bo izveden (ime metode/operacije);
   (c) vhodne parametre ukaza.

2. Omrežje prenese ukaz do izvršitelja ukaza.

3. Izvršitelj ukaza sprejme ukaz.

4. Metoda/operacija je izvedena s pomočjo vhodnih parametrov.

5. Če je operacija izvedena pravilno, se izhod pošlje nazaj do prožilca ukaza.

6. Alternativni potek (ad 5): če pride med izvajanjem ukaza do napake/izjeme, mora biti ta posredovana prožilcu ukaza.

7. Alternativni potek (ad 5): če se ukaz izvaja dlje, kot je pričakovano, izvršitelj ukaza pošlje obvestilo prožilcu ukaza, da je izvajanje v teku.

8. Alternativni potek (ad 2): če pride do napake v prenosu in omrežje ne more usmeriti ukaza do izvršitelja ukaza, se o tem obvesti prožilca ukaza. Obvestilo je izvedeno s pomočjo mehanizma povratnih klicev.

9. Alternativni potek (ad 8): če v določenem časovnem okvirju ni odziva, se o tem obvesti prožilca ukaza (mehanizem povratnih klicev).

Opomba: V specifikacijah CODAC kontrolnega sistema MODEX Task Report [2] je že določen programski vmesnik, ki vsebuje dve metodi: Command_CODAC_PS.reqExecuteCommand (ki jo implementira izvršitelj ukaza) in CommandResponse_CODAC_PS.resCommand (ki jo implementira prožilec ukaza).

Zahteve izvedbe ukaza so [1]:

1. število zahtev: pričakovano je 100 ukazov/s;

2. količina podatkov na ukaz: 1000 bajtov, ampak brez omejitev velikosti;

3. učinkovit ozki programski vmesnik (ang. narrow interface API);

4. sinhron programski vmesnik;

5. asinhron programski vmesnik.

2.2.3 Tok podatkov

V tem primeru uporabe vir podatkov nenehno ustvarja tok podatkov. Podatki so lahko eksperimentalne vrednosti, pridobljeni z meritvami, video ali tok podatkovnih struktur. Na podatkovni tok se lahko prijavi več ponorov podatkov.

Scenarij:

1. Vir podatkov je pripravljen na objavljanje podatkov.

   (a) Vmesna programska rešitev zagotavlja vmesnik, preko katerega lahko vir podatkov potiska podatke, ko so ti na razpolago.
   (b) Vir podatkov mora zagotoviti metainformacije, ki opisujejo strukturo podatkov in druge atribute.
   (c) Vir podatkov specificira kvaliteto prenosa podatkov (QOS, ang. Quality Of Service).

2. Ponor podatkov se prijavi za sprejemanje podatkov.
   (a) Ponor podatkov mora biti sposoben pridobiti metainformacije.
   (b) Ponor podatkov specificira kakovost prenosa podatkov, maksimalno hitrost prenosa, maksimalno zakasnitev ...
   (c) Vmesna programska rešitev mora zagotoviti mehanizem povratnih klicev, ki se sproži, ko je na voljo nov podatek.
3. Ko vir podatkov objavi podatke, jih vmesna programska rešitev transportira do ponora podatkov.

Zahteve primera toka podatkov so [1]:

1. persistenca podatkov - če ponor podatkov ni dosegljiv v določenoem časovnem obdobju (napaka v omrežju, ponovni zagon), sprejme zamujene podatke, ko je spet dostopen;
2. upravljanje:
   (a) Omogočeno mora biti analiziranje in pregledovanje podatkovnih tokov (nadzor).
   (b) Omogočeno mora biti nastavljanje kakovosti prenosa, hitrosti itd. (kontrola).

Primer toka podatkov lahko ima več specializacij:

1. upravljanje dogodkov;
2. spremljanje;
3. masovni prenos podatkov (ang. Bulk Data Transfer);
4. upravljanje alarmov;
5. diagnosticiranje;
6. procesna kontrola - v tem primeru uporabe proces spremlja nek podsistem in ugotovi, katere akcije (če sploh katere) se morajo izvesti zaradi trenutnega stanja.

V naslednjih podpoglavjih bom podrobneje specificiral te specializacije.

Upravljanje dogodkov

Kadar vir dogodka objavi (pošlje) dogodek skozi dogodkovni kanal, ga sprejmejo vsi prijavljeni ponori dogodkov.

Primer uporabe upravljanja dogodkov je izredno podobnen primeru toka podatkov, le da je dogodek oziroma so podatki, ki jih vsebuje dogodek, strukturirani (časovni žig in opis dogodka).

Spremljanje

Spremljanje omogoča ponoru, da spremlja procesno spremenljivko ali stanje izvora.

Spremljanje je v bistvu specializacija primera uporabe toka podatkov, le da podatki vsebujejo informacijo, kaj se je spremenilo.

Iz izkušenj pri gradnji drugih kontrolnih sistemov bi lahko dodali še nekaj zahtev:

- ponor spremljanja lahko določi velikost sprememb, ki se mu zdi relevantna (če se količina ne spremeni dovolj, ponor o tem ni obveščen);
- izvor spremljanja pošlje trenutno vrednost spremljane procesne spremenljivke takoj, ko se ponor prijavi;
- ponor spremljanja lahko določi minimalno in/ali maksimalno frekvenco, s katero želi spremljati procesno spremenljivko - s tem se prepreči preobremenjenost vira ali ponora.

Količinski prenos podatkov

Količinski prenos podatkov omogoča, da se prenese večja količina podatkov od vira do ponora. Ta primer uporabe je specializacija primera toka podatkov z sledečimi omejitvami:

- učinkovito koriščenje pasovne širine – omogočati mora stiskanje, medpomnjenje in druge mehanizme za učinkovitejši prenos podatkov;
- kontrola pretoka podatkov je nujna – izogniti se moramo situaciji, kjer bi lahko izvor preplavil ponor;
- količinski prenos podatkov običajno zahteva zanesljiv prenos podatkov.

Upravljanje alarmov

Kadar alarmni vir zazna nedovoljeno stanje, o tem obvesti upravljavca alarmov. Upravljavec alarmov vodi seznam nedovoljenih stanj in o tem obvešča uporabnike. Dodatno lahko omogoča, da uporabnik preuči nedovoljeno stanje (opis napake, lokacija, itd).

Scenarij:

1. Alarmni vir zazna nedovoljeno stanje (alarm).

   (a) 1.Alternativno: alarmni vir zazna, da stanje ni več nedovoljeno.

2. Alarmni vir posreduje informacijo o spremembi stanja upravljavcu alarmov.

3. Upravljavec alarmov vodi seznam stanj vseh možnih alarmov in dodatno:

   (a) upravljavec alarmov obvesti operaterje in aplikacije, ki spremljajo alarme, o spremembi alarmnega stanja;

   (b) upravljavec alarmov uporablja vzročno-posledično relacijo za poudarjanje vzrokov alarma – tako olajša delo operaterjem.

4. Operater zahteva, da upravljavec alarmov prikaže status neke skupine alarmov – alarmi so lahko filtrirani po različnih kriterijih, npr. pokvarjena naprava, podsistem (plant sistem), pomembnost, nepotrjeni alarmi, ...

5. Operator potrdi alarm – ni nujno, da ukrepa.

Uporabljene splošne zahteve: skalabilnost 2.2.1.

Diagnosticiranje

Ta primer uporabe omogoča sistemskemu vzdrževalcu pregled oziroma pridobitev podrobnih informacij o uporabi in obnašanju omrežja. Pridobljene informacije so uporaben vir informacij pri odkrivanju napak in pri pridobivanju informacij o uporabi (planiranje kapacitet).

Zagotovljene morajo biti sledeče informacije:

1. število sporočil, poslanih ali sprejetih v časovnem okvirju;

2. količina bajtov, poslanih ali sprejetih v časovnem okvirju;

3. na zahtevo se lahko beleži/prestreza sporočila - beležiti se morajo naslednje informacije:

   (a) čas;

   (b) pošiljatelj;

   (c) sprejemnik;

   (d) velikost;

   (e) koristna vsebina (ang. payload).

2.2.4 Procesna kontrola

V primeru uporabe procesne kontrole procesni strežniki spremljajo stanje podsistema in ugotovijo, katere akcije (če sploh) se morajo izvršiti glede na trenutno stanje.

Tipično je ta primer uporabe lahko realiziran s primeroma uporabe spremljanja in proženja ukaza.

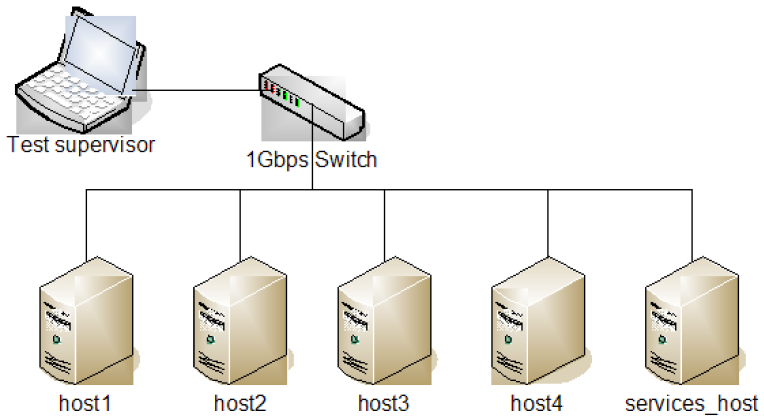
Poglavje 3

Metodologija

V tem poglavju in podpoglavjih bom podrobneje opisal metodologijo praktičnega testiranja vmesne programske opreme, izpustil pa bom podrobnejšo razlago izvorne kode, s katero sem praktično testiral posamezno vmesno programsko opremo. Izvorno kodo si lahko bralec ogleda na priloženi zgoščenki.

3.1 Testno okolje

Testno okolje, ki je bilo uporabljeno za praktično testiranje zmogljivosti vmesnih programskih rešitev, je predstavljeno s sliko 3.1.



Slika 3.1: Shema testnega okolja.

Sestavlja ga 5 ekvivalentnih računalnikov (host1 do host4 in service_host), vsi z naslednjo strojno konfiguracijo:

- CPU AMD Athlon 64-bit 2800+, single core;
- RAM 1GB;

- omrežna kartica 1Gbps NIC;
- operacijski sistem Ubuntu 8.10x64 (kernel 2.6.27-9).

Omrežno stikalo je 8-portno Level One GSW-0806 1Gbps stikalo. S pomočjo „Test superisor“ računalnika sem nadziral delovanje testnih računalnikov in izvajanje testov.

3.2 Testna logika

Izvedel sem dve vrsti testov:

- test hitrosti prenosa (število sporočil na enoto časa);
- test zakasnitve (čas od začetka prenosa do sprejema).

Hitrost prenosa je količina, običajno merjena v bajtih ali bitih na sekundo, ki pove, koliko podatkov se prenese v določenem časovnem obdobju. V mojem primeru je najbolj omejena s procesorsko močjo in pasovno širino omrežja.

Zakasnitev je količina, merjena v sekundah (oziroma mikrosekundah), ki nam pove, koliko časa potrebuje sporočilo od pošiljatelja do sprejemnika. Zakasnitev je ponavadi odvisna predvsem od pasovne širine.

Ti dve količini sem meril na posameznem primeru uporabe 2.2, zato bom v nadaljevanju za vsak primer uporabe razložil način merjenja in promet, ki sem ga za namen čimbolj realne simulacije ustvaril.

Za test hitrosti prenosa je scenarij sledeč:

1. Primer uporabe **upravljanja dogodkov** 2.2.3: kreira se dogodkovni kanal, kamor se registrira ponor dogodka.
2. Primer uporabe **upravljanja dogodkov** 2.2.3: vsak vir dogodka pošilja sporočila dolžine L z določeno frekvenco (λ – število sporočil v sekundi) v dogodkovni kanal.
3. Primer uporabe **izvedbe ukaza** 2.2.2: vsak prožilec ukaza pošilja ukaze s koristno vsebino v velikosti L in frekvenco λ do izvrševalca ukaza. V vsakem primeru sporočilo vsebuje:
   (a) identifikacijsko številko sporočila (sekvenčna številka) pošiljatelja;
   (b) čas, ob katerem je bilo ustvarjeno sporočilo (resolucija v nanosekundah).
4. Obnašanje sprejemnika:
   (a) Test zakasnitve:
   upravljanja dogodkov 2.2.3 Prejemnik dogodka (ponor dogodka) takoj ob prejetju odpošlje isti dogodek pošiljatelju. Sporočilo vsebuje identifikacijsko številko prejetega sporočila in časovni žig kreacije prejetega sporočila.

izvedbe ukaza 2.2.2 Odgovor je preprosto sporočilo (ang. return).

(b) Test hitrosti prenosa: sprejemnik sporočila/ukaza logira čas prejetja sporočila (v mikrosekundah).

5. Obnašanje prožilca/vira dogodka ob prejemu odziva (nanaša se samo na test zakasnitve): prožilec/vir dogodka izračuna časovno razliko od trenutka, ko je bilo sporočilo poslano, do prejema odziva.

3.3 Statistična obdelava podatkov

Podatki, zbrani med izvajanjem testov, so bili obdelani po naslednjih postopkih:

Test hitrosti prenosa: Pri tem testu se pošiljajo sporočila/ukazi iz točke X v točko Y, kolikor hitro je mogoče (brez postanka med dvema poslanima sporočiloma). Pri tem se zabeležita čas, ob katerem se sporočilo/ukaz pošlje, in čas, ob katerem se sporočilo prejme. Pri tem lahko pride do dveh pojavov oziroma režimov:

1. Kapaciteta sistema je zadostna ($\lambda <= \lambda_{max}$). V tem primeru se pričakuje, da je razlika med dvema prejemoma enaka $1/\lambda$, vendar s statistično razpršenostjo okrog te vrednosti (ang. jitter).

2. Kapaciteta sistema ni zadostna ($\lambda <= \lambda_{max}$). V tem primeru se pričakuje razlika prihodnjih časov $1/\lambda$. Če vmesna programska oprema uporablja medpomnilnik, se bo ta napolnil.

Iz podatkov, zbranih pri posameznem testu hitrosti prenosa, se nato določijo naslednji parametri:

1. Statistična distribucija prihodnih časov, zmanjšanih za $1/\lambda$:

   (a) minimum;

   (b) maksimum;

   (c) 50 in 95 percentil;

   (d) povprečje;

   (e) standardno deviacijo.

Opomba: testi so bili izvedeni s sporočili različnih dolžin (25, 50, 100, 200, 400, 800, 1000, 1200, 1300, 1400, 1500, 1600, 1700, 2000, 5000, 100k, 50k, 100k, 500k in 1M). Poleg metrike sporočila/sekundo se večkrat uporablja tudi metrika bajt/sekunda.

Test zakasnitve: Pri teh testih se zakasnitev izračuna tako, da se obhodna zakasnitev deli z 2 in tako dobi zakasnitev. Tukaj se povzame, da je zakasnitev pošiljanja in sprejemanja enaka. Iz tega podatka so nato izračunani naslednji parametri:

- minimum;

- maksimum;
- 50 in 95 percentil;
- povprečje;
- standardna deviacija.

Opomba: meritev zakasnitve ima pomen samo v primeru, če sistem ni preobremenjen ($\lambda < \lambda_{max}$). Test zakasnitve je bil izveden z različnimi dolžinami sporočil (25, 50, 100, 200, 400, 800, 1000, 1200, 1300, 1400, 1500, 1600, 1700, 2000, 5000, 100k, 50k, 100k, 500k in 1M).

3.4 Avtomatizacija testov

Ker je bilo potrebno izvesti veliko število testov (22 različnih velikosti sporočil, 2 vrsti testov, 6 vmesnih programskih oprem), sem uporabil testno okolje, s pomočjo katerega sem si poenostavil testiranje.

Testno okolje sestavljajo naslednje entitete:

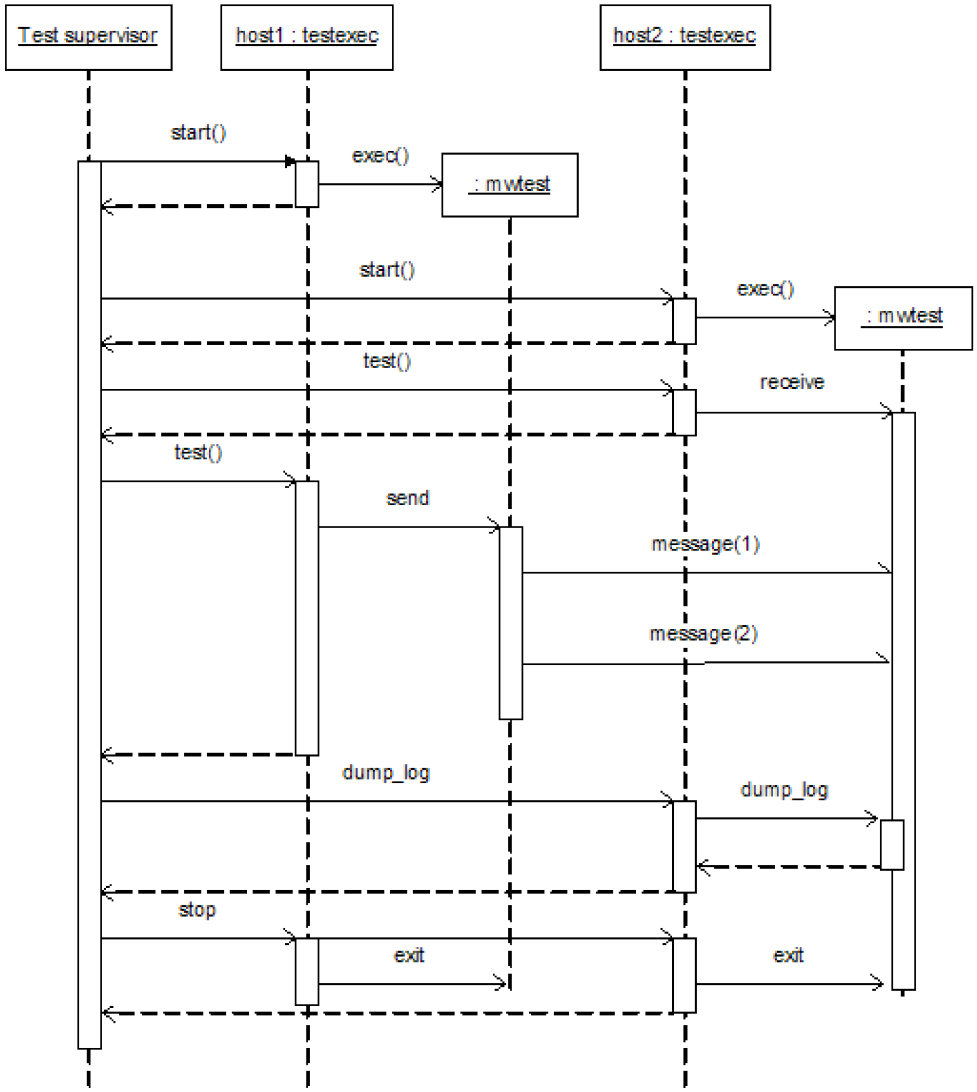
- Testexec: Java proces, kontroliran preko HTTP protokola (integriran spletni strežnik Jetty), ki zaganja testne programe s pomočjo standardnih tokov.
- Mwtest: C++ proces, ki izvaja vlogo sprejemnika oziroma oddajnika z uporabo izbrane vmesne programske opreme. Tip vmesne programske opreme je podan kot vhodni parameter, kontrola se izvaja preko standardnih tokov.
- JUnit test: definira testni skript in poskrbi za analizo rezultatov. Ta proces se nahaja na nadzornem računalniku ("Test supervisor"). JUnit test shrani rezultate posameznega testa kot tekstovni dokument.

Interakcijo med entitetami predstavlja slika 3.2.

Ko se testni nadzornik (JUnit test) zažene, nadzira/kontrolira testexec procese, ki na zahtevo izvedejo test. Najprej se poskrbi za zagon mwtest procesa z ukazom start, ki poskrbi za uporabo izbrane vmesne programske opreme, določene s testnim nadzornikom. Nastavijo se tudi potrebni parametri posamezne vmesne programske opreme.

Testni nadzornik poskrbi za zagon sprejemnika (ukaz receive). Nato se zažene pošiljatelj prek ukaza receive, ki določa število in velikost sporočil, ki bodo poslani do sprejemnika.

Ko pošiljatelj odpošlje vsa sporočila, testni nadzornik zbere listo sporočil sprejemnika (ukaz dump_log). Lista za vsako sprejeto sporočilo vsebuje podatek o času oddaje in sprejema (V primeru testa zakasnitve, ki ni prikazan na sekvenčnem diagramu, se sporočilo, ki ga prejme sprejemnik, nemudoma pošlje nazaj do pošiljatelja). Ta lista podatkov omogoča, da se izvede statistična analiza podatkov.



Slika 3.2: Sekvenčni diagram izvajanja testov.

Poglavje 4

Študija vmesnih programskih oprem

V tem razdelku bom pregledno opisal arhitekturo posamezne vmesne programske opreme. Iz teh opisov se da razbrati nekatere nefunkcionalne zahteve, ki bodo v nadaljevanju predstavljene v pregledni tabeli.

Za študij sem izbral sledeče razširjene vmesne programske rešitve:

1. EPICS Channel Access;
2. OmniOrb (CORBA);
3. TAO CORBA;
4. RTI DDS;
5. Open Splice DDS;
6. ZeroC.

4.1 EPICS Channel Access

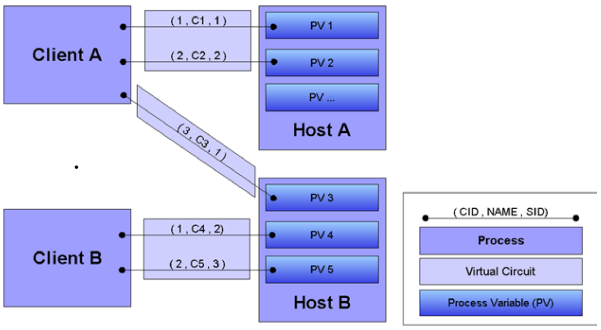
Channel Acces (CA) je omrežni protokol, ki ga za delovanje uporablja EPICS (ang. Experimental Physics and Industrial Control Systems) [6]. EPICS je niz odprtokodnih programskih orodij, knjižnic in aplikacij, ki tvorijo porazdeljen kontrolni sistem za znanstvene instrumente, kot so pospeševalniki delcev, teleskopi in druge veliki znanstveni poskusi.

Ta protokol je oblikovan tako, da zagotovi minimalne režijske stroške in poveča zmogljivost omrežja za prenos velikega števila majhnih paketov. Poleg tega potrebuje malo virov, kar omogoča delovanje na sistemih z omejenimi viri.

4.1.1 Arhitekturni pregled

Podprte so naslednje gostiteljske platforme: Solaris, Linux, RTEMS, vxWorks, itd. Podprti jeziki so C/C++ in Java (verzija 1,4 ali novejša). S pomočjo zunanjih knjižnic sta podprta tudi Python in Perl.

CA sledi standardni arhitekturio djemalec-strežnik [6] in je optimiziran za spremljanje (ang. event handling) procesnih spremenljivk; procesna spremenljivka (PV, ang. Process Variable) je predstavitev vrednosti v kontrolnem sistemu. Komunikacija med strežnikom in odjemalcem se izvede tako, da pošlje zahtevek preko UDP-ja in TCP-ja. Odjemalec bo uporabil UDP za iskanje gostitelja in procesnih spremenljivk, strežnik pa za oddajanje obvestil o zagonu, prisotnosti in zaustavitvi (implicitno tudi sesutje ali izgubo povezavo). Ko odjemalec zahteva posebne procesne spremenljivke (z navedbo imena), se pošlje UDP sporočilo, ki je oddano bodisi v podomrežje bodisi v seznam vnaprej določenih naslovov, in strežnik, ki gosti zahtevano procesno spremenljivko, se bo ustrezno odzval. Odjemalec bo vztrajno iskal spremenljivke (v primeru okvare bo poskusil znova), dokler ne bo povezan. Ta mehanizem odkrivanja napak in dinamičnega povezovanja enostavno omogoča podvajanje procesnih spremenljivk na večih gostiteljih.



Slika 4.1: Shema CA protokola. (Vir: www.aps.anl.gov/epics)

Izmenjave podatkov med odjemalcem in strežnikom se izvajajo preko povezave TCP (hkratno pošiljanje se ne uporablja za distribucijo vrednosti procesnih spremenljivk). Po iskanju procesne spremenljivke, odjemalec vzpostavi TCP povezavo s strežnikom. Če je na istem gostitelju lociranih več procesnih spremenljivk, bo odjemalec ponovno uporabil obstoječo TCP povezavo 4.1.

TCP povezava med odjemalcem in strežnikom se imenuje virtualna povezava. Vsaka povezava ima dodeljeno prioriteto, ki jo strežnik lahko uporablja za priorizacijo zahtev glede na trenutno obremenitev. Ustvari se neodvisna povezava za vsako prednostno stopnjo, ki jo izbere odjemalec, kar omogoča predkupno razvrščanje odpreme v strežniku, če jo operacijski sistem podpira, in tudi navedbo omrežja ter načrtovanje prednosti, če usmerjevalnika in/ali LAN to podpirata . Ne glede na število procesnih spremenljivk, vzpostavljenih s strani odjemaleca ali strežnika, je vsak par odjemalec-strežnik povezan z natanko eno TCP povezavo za vsako prednostno stopnjo. Prioriteta je določena pri ustvarjanju virtualnih povezav. FIFO dostava je zagotovljena samo znotraj posamezne TCP povezave za vsako prednostno raven (in ne preko več povezav).

Vsako CA sporočilo je sestavljeno iz zaglavja, ki mu sledi vsebina. Zaglavje je vedno

prisotno. Njegova struktura je fiksna in vsebuje vnaprej določena polja. Glavo predstavljata najmanj ID ukaza in velikost tovora . Druga polja lahko vsebujejo podatke, ki imajo pomen za posamezen ukaz. Če polje ni uporabljeno v ukazu, mora biti njegova vrednost postavljena na 0.

Vsebina koristnega tovora je zaporedje bajtov, ki je odvisno od ukaza in verzije. Skupna velikost posameznega sporočila je omejena na 16384 bajtov za različice CA protokola, starejše od 4.9. V različici 4.9 in več se lahko uporabi razširjeno sporočilo, ki omogoča tovor velikosti do 4GB. Vendar se v praksi največja velikost sporočila omeji z velikostjo predhodno dodeljene sistemske spremenljivke (konfigurabilne ob zagonu). CA ne podpira toka podatkov, temveč opredeljuje posebne strukture za prenos podatkov. Glavni razlog je učinkovitost, saj je v veliko primerih mogoče prenesti več kot eno vrednost. Obstaja več vrst osnovnih podatkov: ASCII niz, short, integer, float, enumeration, znak (oktet) in double. CA protokol zahteva vrednosti v plavajoči vejici, ki jih je potrebno preoblikovati v skladu s standardom IEEE 754. Vse vrednosti se prenašajo prek omrežja v omrežnem vrstnem redu (ang. big-endian). Vsaka od sedmih osnovnih vrst podatkov se lahko uporablja tudi kot element v nizu.

Poleg osnovnih podatkovnih tipov obstajajo strukturirane vrste, ki omogočajo dostop do več kot ene vrednosti. Te strukture so organizirane v tipizirane hierarhije. Te strukture so status (STS), timestamp (TIME), graphic (GR) in control (CTRL). Statusna struktura vsebuje tudi resnost alarma. Graphic struktura razširja status strukturo z zagotavljanjem omejitev alarma, enoto in natančnostjo. Control struktura razširja graphic strukturo z mejnimi limitami.

CA uveljavlja paradigmo zahtevek-odgovor. Podprte zahteve so: ustvari/uniči povezavo (poveže/prekine povezavo s procesno spremenljivko), pridobi zahtevo (pridobi vrednost iz procesne spremenljivke), nastavi (določi vrednost procesne spremenljivke) in zahteva prijavi (spremlja vrednosti/alarm statusa procesne spremenljivke). RPC (ang. Remote Procedure Call) ni podprt. CA protokol ponuja zanesljive pristope (odjemalec dobi informacijo o zaključku) in prenos po najboljših močeh. CA protokol omogoča tudi asinhron programski vmesnik.

CA spremljanje, kot poseben primer upravljanja dogodkov, obvesti naročnike, kadar procesna spremenljivka spremeni svoje stanje. V podporo upravljanju alarmov, ki se opravlja preko monitoriranja, CA podpira dve dodatni polji: global alarm acknowledgement flag (če je potrebno, da so prehodni alarmi potrjeni) in global alarm acknowledgement severity (potrjen je najvišji nivo prehodnih alarmov).

Če se želi preprečiti poplavo odjemalcev z dogodki, CA protokol izvaja kontrolo pretoka. Dostopne pravice so lahko opredeljene za vsak kanal. Možne pravice so: dostop, branje in/ali pisanje. Določene so s parom (uporabniško ime – gostitelj) in se lahko spreminjajo dinamično.

4.1.2 Proces razvijanja

CA je sestavljena iz dveh knjižnic: CA odjemalca (CAC) in CA strežnika (CAS). CAS sam po sebini samostojen proces (potrebuje vir podatkov). EPICS uporablja CAS na vrhu svoje podatkovne baze v realnem času za oblikovanje IOC (ang. software input-output controler). Vendar lahko razvijalci razvijejo vir podatkov z implementacijo nekaterih programskih vmesnikov CAS-a. Druga možnost je, da se implementira generičen in'memory PV; to pretvori CAS v sporočilno storitev, kjer je funkcija CA put uporabljena za objavljanje dogodkov in CA spremljanje pa za prijavni mehanizem. Primere izvorne kode lahko bralec najde na domači strani EPICS-a (http://www.aps.anl.gov/epics/docs/index.php).

4.1.3 Uvajanje in vzdrževanje

Zaradi dinamičnega odkrivanja CA ne zahteva veliko (če sploh) konfiguracije. Vsaka konfiguracija se opravi prek sistema sistemskih spremenljivk (C++) ali zagonskih parametrov/konfiguracijske datoteke (Java).

Če je prisotnih več podomrežij, mora biti med vsakim vzpostavljen prehod (CA gateway) in odjemalci morajo biti nastavljeni tako, da odpošiljajo sporočila preko prehoda.

4.2 CORBA

4.2.1 OMG organizacija

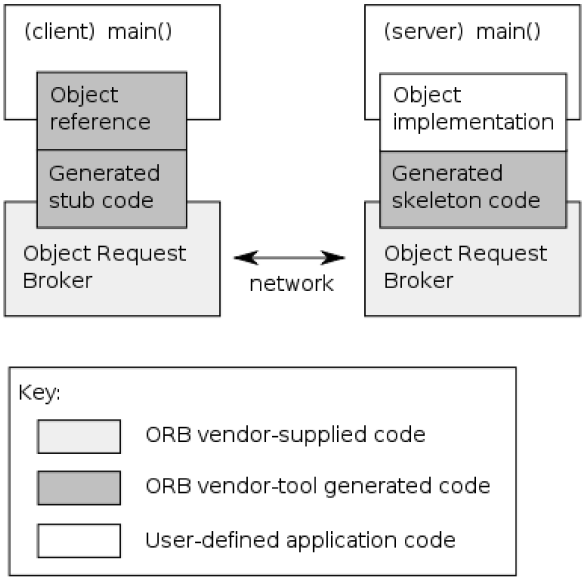
Object Management Group, Inc. (OMG) je odprt standardizacijski konzorcij, ki ustvarja in obravnava zahteve za vmesno programsko opremo, modeliranje in vertikalna domenska programska ogrodja. V naslednjem poglavju sledi kratek pregled arhitekture, ki je opredeljena v OMG Object Request Broker Architecture (CORBA) 3.1 specifikaciji [8]. Ta specifikacija je sestavljena iz treh dokumentov:

- CORBA Object Model and the operation of Object Request Broker (ORB);
- ORB interoperability architecture and protocols;
- CORBA Component Model (CCM).

4.2.2 Arhitekturni pregled

CORBA temelji na povezavno usmerjeni arhitekturi odjemalec/strežnik. Poleg tega zagotavlja mehanizem za objektno orientirano oddaljeno klicanje metod. Standard podrobno določa tudi storitve, ki temeljijo na bolj ohlapno sklenjeni paradigmi objavi/naroči, kot je Event Service in njena nadgradnja Notification Service. Čeprav ti dve arhitekturi zagotavljata konkurenčne pristope za vzpostavitev komuniciranja, sta pogosto komplementarno uporabljeni znotraj istega sistema.

CORBA aplikacije sestavljajo objekti, ki izpostavljajo nekatere funkcionalnosti ali zagotavljajo podatke z dobro opredeljenimi programskimi vmesniki. Prisotnih je lahko poljubno število objektov istega tipa. Vsak tip objekta zahteva opredelitev vmesnika v obliki OMG Interface Definition Language (IDL) [9]. Ta opredelitev vmesnika se uporablja na strani odjemalca za lokalno ali oddaljeno proženje operacij in pakiranje argumentov, ki jih odjemalec pošlje. Ciljni objekt uporablja isti programski vmesnik za odpakiranje argumentov ter izvedbo operacije 4.2. Objekt nato zapakira rezultate in jih pošlje nazaj do odjemalca. Končno, ko se klic metode vrne do odjemalca, se vmesnik zopet uporabi za odpakiranje rezultatov. To opisuje zanesljivo sinhrono, povezavno usmerjeno oddaljeno proženje metod. Alternativni pristop je določitev enosmerne semantike v definiciji IDL, rezultat pa je prenos po najboljših močeh.



Slika 4.2: Ilustracija avtogenerirane infrastrukture (CORBA IDL) in komunikacije med odjemalcem in strežnikom. (Vir: http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture)

V specifikaciji OMG CORBA je standardizirana sintaksa in semantika IDL. Je neodvisna od obstoječih programskih jezikov, vendar ločene specifikacije zagotavljajo Preslikavo IDL v C, C+ +, Java, COBOL, Smalltalk, Ada, Lisp, Python in IDLscript.

Vsak objekt CORBA ima unikatno referenco, ki omogoča odjemalcem sklicevanje na točno določeno instanco objekta. Čeprav odjemalec vidi to kot sklicevanje na neposreden objekt, se dejansko sklicuje na nastavek IDL. Ta nato poskrbi za propagiranje operacije skozi servis ORB (ang. Object Request Broker), ki poskrbi za dostavo operacije do ciljne instance objekta. Rezultat operacije se podobno usmerja nazaj do odjemalca (prikazuje slika 4.2).

V primeru da se ciljni objekt nahaja izven procesnega prostora ali celo na drugem

gostitelju, mora servis ORB na odjemalčevi strani poskrbeti za delegacijo do ciljnega objekta. Ker pa je zahtevana interoperabilnost vseh servisov ORB, se morajo le-ti najprej dogovoriti o uporabi istega protokola GIOP (ang. General Inter-ORB Protocol). GIOP je abstrakten protokol, s katerim komunicirajo različne izvedbe servisov ORB. Nekatere konkretne implementacije GIOP protokola so:

- Internet Inter-ORB Protocol (IIOP);
- SECure Inter-ORB Protocol (SECIOP);
- SSL InterORB Protocol (SSLIOP);
- HyperText InterORB Protocol (HTIOP).

Vsi ti protokoli so standardizirani in podprti s strani skoraj vseh implementacij CORBA. Specifikacija zahteva vsaj podporo protokola IIOP.

4.2.3 TAO CORBA

TAO CORBA [11] je odprtokodna implementacija standarda CORBA. Bazira na ACE (ang. ADAPTIVE Communication Environment) knjižnici (ADAPTIVE je akronim za Dynamically Assembled Protocol Transformation and Validation Environment). Zadnje izdaje TAO so kompatibilne s standardom CORBA 3.0.

4.2.4 OmniORB

OmniOrb [12] implementacija CORBE je bila prvotno razvita na ORL (Olivetti Research Ltd.). Kasneje je bila izdana kot odprta koda. Leta 1999 je ORL postal del AT&T Laboratories Cambridge, dokler ni bilo podjetje zaprto. Vendar se je razvoj nadaljeval kod neodvisen projekt. Zadnji stabilni izvod ustreza verziji standarda CORBA 2.6.

4.2.5 Proces razvoja

Implementacija enostavnega primera strežnik/odjmalec zahteva sledeče korake [9]:

1. Definicija vmesnika v IDL jeziku.
2. Generacija kodnih nastavkov z uporabo IDL prevajalnika. Prevajalnik zgenerira kodo z določenim vmesnikom.
3. Implementacija servanta.
4. Implementacija odjemalca.

4.2.6 Uveljavljanje in vzdrževanje

CORBA ne omogoča dinamičnega odkrivanja instanc objektov, zato mora odjemalec poznati lokacijo objekta, ki ga želi izvesti (ime gostitelja, port, na katerem teče ORB servis, in ime objekta), lahko pa najde instanco objekta s pomočjo servisa Naming/Trading, na katerem mora biti objekt registriran.

Entitete servisa Naming/Trading morajo biti previdno upravljane – proces mora počistiti objektne reference. Sistemske napake lahko privedejo do potrebe ponovnega zagona celotnega sistema ali do potrebe po ročnem editiranju entitet servisa Naming [10]. Aplikacije, ki temeljijo na CORBI, so izredno občutljive na prisotnost NAT-a in požarnih zidov.

4.3 DDS

DDS (ang. Data-Distribution Service) [5] je specifikacija vmesna programske opreme, ki temelji na podatkovno usmerjeni paradigmi objavi/naroči, primerni za realnočasovne aplikacije, ki zahtevajo (selektivno) distribucijo podatkov, kot so porazdeljeni krmilni sistemi, industrijska avtomatizacija, telekomunikacijske opreme za nadzor, senzor omrežij in omrežnih sistemov za upravljanje. Specifikacija je zasnovana tako, da zagotavlja distribucijo podatkov z minimalno režijo in s sposobnostjo skaliranja na tisoč izdajateljev in naročnikov.

V specifikaciji DDS (verzija 1.2) vmesnik opisuje dve plasti [13]:

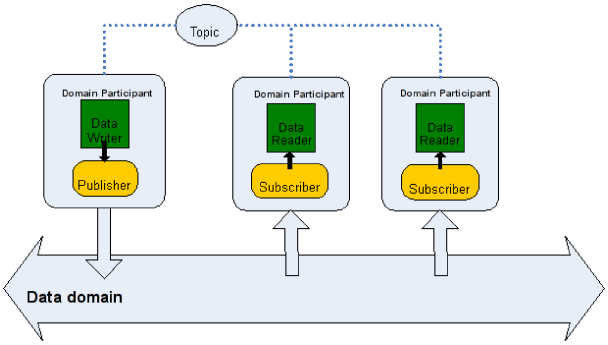
- DCPS (ang. Data-centric Publish-Subscribe) plast, ki je odgovorna za učinkovito dostavo pravilnih podatkov od izdajatelja/-ev do prejemnika/-ov).
- višja opcijska plast se imenuje DLRL (ang. Data Local Reconstruction Layer), ki je zgrajen na vrhu plasti DCPS, ter zagotavlja preprost objektno usmerjen dostop do izmenjanih podatkov.

Specifikacija OMG DDS še ne opredeljuje skupnega omrežnega protokola, zato DDS-i različnih izvedb uporabljajo različne pristope za transportno plast. Za zagotovitev interoperabilnosti je bil ustvarjen žični protokol, ki se imenuje Interoprability Wire Protocol Specification (RTPS-Wire protokol) [14]. Poleg standardnih zahtev RTPS vključuje tudi dobro učinkovitost, omogoča različne vrste prenosa preko standardnega IP protokola, kompatibilnost za starejše verzije ...

RTPS Wire-protokol je razdeljen na dva modela, platformno neodvisni model (PIM, ang. Platform Independant Model), ki opisuje protokol v smislu „navideznega stroja”, in platformno specifičen model (PDM – Platform Specific Model), ki opredeljuje reprezentacijo (biti in bajti) vseh RTPS tipov in sporočil. PSM-ji lahko podpirajo več vrst prenosnih protokolov, vendar morajo po specifikaciji RTSP-Wire protokola podpirati vsaj PSM na vrhu UDP/IP prometa, ker je na voljo skoraj na vseh platformah, je enostaven, lahek in zagotavlja najboljši kompromis med determinizmom (brez raznih zamud, kot jih ima sistem, uveden zs TCP) ter zanesljivostjo in njegovo podporo za oddajanje več naročnikom (ang. multicast).

4.3.1 DCPS (ang. Data Centric Publish Subscribe) plast

DCPS plast opredeljuje "globalni podatkovni prostor", ki je dostopen vsem zainteresiranim aplikacijam in določa, kako se lahko sprejmejo zahtevki, ki se nanašajo na dele podatkovnega prostora. Določa predvidljivo in zanesljivo distribucijo podatkov z minimalno režijo.



Slika 4.3: Ilustracija DCPS plasti. (Vir: www.rti.com)

Izmenjava podatkov med komunikacijskimi vozlišči teče s pomočjo naslednjih subjektov (slika 4.3):

- Data Writer je objekt, ki se uporablja za sporočanje o obstoju in vrednosti podatkovnega objekta izdajatelju (Publisher).
- Izdajatelj (ang. Publisher) je odgovoren za distribucijo podatkov. Poskrbi, da se podatki različnih tipov distribuirajo do vseh zainteresiranih naročnikov. Distibucija je izvedena skladno s politiko QoS (ang. Quality of Service), določeno z izdajateljem ali DataWriterjem.
- Bralec podatkov (ang. DataReader) zagotavlja prejetje podatkov določenega tipa (ang. topic), ki jih prejme naročnik.
- Naročnik (ang. Subscriber) je objekt, odgovoren za sprejemanje objavljenih podatkov, ki so na voljo bralcem podatkov (v skladu s svojimi QoS). Naročniki lahko prejmejo podatke iz različnih podatkovnih tipov (topicov).
- Topic je konceptualno umeščen med naročnikom in izdajateljem. Identificira podatke v globalnem podatkovnem prostoru. Topic asociira unikatno ime v domeni, tip podatkov in QoS. V nekem časovnem obdobju lahko obstaja več instanc istega topica. S pomočjo koncepta ključev (ang. key) se zagotovi preprost način za ločevanje med njimi.
- QoS (ang. Quality of Service) je splošni pojem, ki določa vedenje storitev. Z določitvijo posebnih politik je omogočeno želeno obnašanje storitve. Storitev obnašanje

lahko opišemo kot seznam neodvisnih politik, ki zagotavljajo še večjo prilagodljivost. Tipične politike: zanesljivost, trajnost, življenjska doba, omejitve virov in mnogi drugi.

- Domena je porazdeljen koncept, ki povezuje vse aplikacije, sposobne sporazumevanja znotraj določene domene. Znotraj iste domene lahko komunicirajo med seboj samo izdajatelji in naročniki.

4.3.2 OpenDDS

OpenDDS je odprtokodna C++ implementacija OMG DDS specifikacije. Zgrajen je na vrhu ACE plasti, ki zagotavlja platformno prenosljivost [15]. Uporablja nekatere zmogljivosti TAO (ang. The ACE Orb), kot je prevajalnik IDL in strežnik CORBA za DCPS Information Repository.

Prenos podatkov v OpenDDS se izvede preko specifičnih transportnih plasti (PTL, ang Plug Transport Layer), ki omogočajo uporabo storitev z različnimi protokoli. OpenDDS trenutno predvideva štiri privzete izvedbe transportnih plasti:

- Enostaven UDP (nezanesljiva dostava);
- Enostaven TCP (zanesljiva dostava);
- Nezanesljiva dostava večim naročnikom (prek UDP, nezanesljiva dobava);
- Zanesljiva dostava večim naročnikom (prek UDP, zanesljive dostave).

4.3.3 RTI DDS

RTI (Real-Time Innovations, Inc.) DDS (ang. Data Distribution Service) je lastniška implementacija specifikacije OMG DDS [16]. V preteklosti je bil znan kot NDDS. V skladu DDS specifikacijami je v celoti skladen z DCPS plastjo. Ima vgrajeno veliko transportnih protokolov:

- Shared memory (komunikacija znotraj vozlišča);
- Multicast and Unicast preko UDP ;
- Sposobnost za uporabo več različnih vrst transporta.

Poleg vgrajenih vrst prenosa je na voljo tudi WAN transportni vtič, ki omogoča varno komuniciranje preko odprtih omrežij in prehod skozi NAT in požarne zidove.

RTI ponuja razne dodatne servise, kot so: RTI Persistence service, RTI Recorder, RTI Real-Time Connect, itd.

4.3.4 OpenSplice DDS

OpenSplice DDS (Prism Tech, Ltd.) je implementacija OMG DDS standarda. Prism Tech je v začetku leta 2009 objavil odprtokodno verzijo [17].

OpenSplice DDS arhitektura ustreza generalni arhitekturi DDS. Dodatno uvaja Domain Service: na vsakem vozlišču mora teči natanko ena instanca za DDS domeno. Domain Service upravlja prenos podatkov z drugimi servisi v isti domeni. Z drugimi besedami, komunicira z specifičnimi aplikacijskimi procesi, ki tečejo na vozlišču preko skupnega pomnilniškega prostora.

Ta arhitektura ima prednost, ker reducira omrežni promet med procesi v istem vozlišču; vendar to predstavlja tudi slabost, kadar mora Domain Service poslati podatke preko omrežja, kar prinese dodatno zakasnitev.

4.4 ZeroC Ice

Ice (ang. Internet Communication Engine) je moderna, objektno orientirana vmesna programska rešitev [18]. Cilje dizajna bi lahko opisali z enim stavkom: "Zgradimo vmesno programsko opremo, ki bo zmogljivejša od CORBE, vendar brez vseh njenih napak."

4.4.1 Arhitekturni pregled

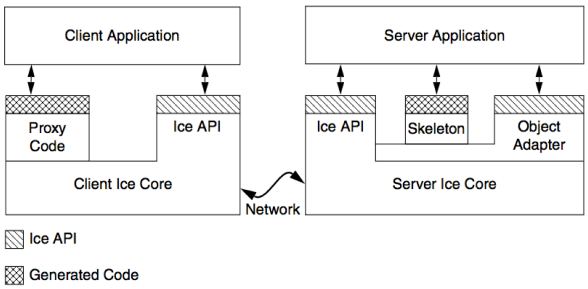
Ice vsebuje orodja, programski vmesnik in knjižnice, ki omogočajo gradnjo objektno orientiranih odjemalcev in strežnikov. Primeren je za aplikacije v heterogenem okolju, saj omogoča gradnjo v različnih jezikih in na različnih platformah. Ice podpira sledeče programske jezike in operacijske sisteme:

- C++ podpora za x86 in x64 na Windows/Linux/MacOSX/Solaris, SPARC in Solaris;
- .NET (x86 in x64 na Windows);
- Java (verzija 1.5 ali več);
- Python, Ruby in PHP.

Ice bazira na mehanizmu oddaljenega proženja ukazov (RPC, ang. Remote Procedure Call) (slika 4.4). Programski vmesnik, operacije in tipi podatkov, ki se izmenjujejo, so določeni z uporabo jezika Slice. Ta omogoča definicijo konvencije odjemalec-strežnik, tako da je neodvisna od programskega jezika. Definicije v jeziku Slice se potem prevedejo s prevajalnikom v vmesnik izbranega programskega jezika.

Ice omogoča uporabo protokola RPC na TCP/IP ali UDP povezavi in možnost uporabe varne povezave SSL z vtičem.

Vsako sporočilo ima 14-bajtno glavo. To je tudi minimalna velikost sporočila, ki se uporablja za preverjanje povezave. Najmanjše sporočilo s koristno vsebino zahteva 36



Slika 4.4: Koncept komunikacije v ZeroC Ice. (Vir: www.zeroc.com)

bajtov, minimalna velikost odgovora pa 25 bajtov. Ice omejuje velikosti zahtevkov in odgovorov. Po privzetih nastavitvah je omejitev nastavljena na 1MB, ampak teoretično se lahko dvigne do 4GB.

Ice podpira kompresijo na žici in s tem omogoča prihranek pasovne širine. To je uporabno v aplikacijah, ki izmenjujejo veliko količino podatkov. Sporočila, manjša od 100 bajtov, se ne kompresirajo. Kompresija ni podprta na vseh platformah za vse programske jezike. Če se uporablja kompresijski način delovanja, se celotno sporočilo (brez glave) stisne z algoritmom bzip2. V primeru hitre povezave je procesorski čas, porabljen za stiskanje podatkov kompresiranega sporočila, daljši od časa prenosa nezgoščenih podatkov.

Ice je primeren za gradnjo visoko učinkovitih mehanizmov za dogodkovno posredovanje, ker podpira posredovanje sporočil brez znanja o vsebini. To pomeni, da ne potrebuje pakiranja in odpakiranja.

Ice protokol prav tako podpira dvosmerno delovanje: če strežnik želi poslati sporočilo objektu, ki je registriran na samodejni povratni klic (ang. Callback Object), priskrbljen s strani odjemalca, le-to pošlje preko že vzpostavljene povezave. To je pomembno za primere, ko je odjemalec za požarnim zidom, ki ne dovoljuje vhodnih povezav.

Če je objekt izvajanja v istem naslovnem prostoru, Ice uporablja običajni lokalni klic funkcije in se tako izogne režijskim stroškom.

Ice zahteve sledijo "at-most-once" semantiki: narejeno je vse za dostavo zahteve do pravilne destinacije, odvisno od okoliščin pa lahko ponovno pošlje neuspešno dostavljen zahtevek. Ice garantira, da bo poslal sporočilo natančno enkrat ali o napaki obvestil odjemalca s primerno izjemo. Izjema temu pravilu je lahko le mehanizem datagramskega proženjea preko UDP povezave.

Ice uporablja privzeto sinhrono oddaljeno proženje metode – v času klica se proces ustavi in se ob koncu ponovno vrne v izvajanje. Ice ponuja tudi mehanizme časovnikov, ki ustavi metodo, če ni izvedena v določenem časovnem okviru. Prav tako podpira asinhrono oddaljeno proženje metod (AMI, ang. Asynchronous Method Invocation). V tem primeru se mora zagotoviti dodaten povratni objekt (ang. callback object) kot parameter. Strežniški ekvivalent AMI-ja je mehanizem asinhronega razpošiljanja metod (AMD, ang. Asynchronous Method Dispatch). Za sinrono (privzeto) klicanje metod se za vsak klic naredi proces, ki teče, dokler se metoda ne konča. Namesto da vsilimo takojšnjo izvedbo

operacije, lahko strežniška aplikacija časovno odloži izvajanje procesa in ko se nato proces končno izvede, se o tem obvesti odjemalca. S tem se doseže skalabilnost – več tisoč operacij se lahko izvaja istočasno ali pa čakajo v željenem vrstem redu.

Odjemalci lahko uporabljajo enosmerno semantiko za proženje metod (ang. one-way invocations), ki se ujema s prenosom po najboljših močeh. Ko odjemalec pošlje sporočilo, ne dobi nobenega potrdila o prejemu s strani strežnika – podatkovni tok teče samo v eno smer. Enosmerni prenos je mogoč samo prek tokovno orientiranih podatkovnih protokolov, kot so TCP/IP ali SSL. Čeprav so podatki/operacije poslani preko tokovno orientiranih protokolov, lahko vseeno pride do izgube vrstnega reda, ker strežnik obdeluje zahteve v več nitih. Če hočemo zagotoviti pravilni vrstni red, lahko zahtevamo uporabo samo ene niti ali nastavimo strežnik tako, da sinhronizira niti.

Datagramske operacije/sporočila imajo prav tako semantiko "po najboljših močeh", vendar ta uporablja UDP transportni protokol. Podobno kot pri enosmernem proženju se lahko datagramska operacija izvede samo, kadar ne pričakuje vrnjene vrednosti ali izjeme. Pošiljajo se asinhrono. Datagramske operacije/sporočila so primerne za majhna sporočila v omrežjih z nizko verjetnostjo izgube sporočila. Prav tako so primerne za situacije, ki zahtevajo nizko zakasnitev. Datagramske opreacije se lahko uporabljajo za pošiljanje operacij večim strežnikom (multicast).

Ice omogoča podvajanje objektnih adapterjev (in njihovih objektov) na več gostiteljih. S tem zagotovimo redundanco in/ali porazdeljen strežnik.

Ice vsebuje naslednje servise:

- Freeze in FreezeScript: persistenčni servis.
- IceGrid: implementacija lokacijskega srevisa.
- IceBox: enostaven aplikacijski servis.
- IceStorm: učinkovit servis objavi/naroči.
- Blacier2: varna komunikacija prek požarnih zidov.
- IcePatch2: zagotovitev posodabljanja programske opreme.

4.4.2 Razvojni proces

Slice (ang. Specification Language for Ice) je fundamentalen abstarkcijski mehanizem za ločevanje objektnih vmesnikov od same implementacije. Prvi korak razvoja je definiranje vmesnikov v jeziku Slice.

Definicije Slice se v nadaljevanju prevedejo v izbran programski jezik. Prevajalnik poskrbi za prevedbo jezikovno neodvisnih definicij v jezikovno specifične tipe in vmesnike. Ker je jezik Slice deklarativne narave, ne moremo pisati izvajalnih ukazov.

4.4.3 Izvedba in vzdrževanje

Izvedljive datoteke (strežnik/odjemalec) se lahko postavijo kamorkoli, kljub temu da gostitelji uporabljajo drugačne operacijske sisteme in je strežnik ali odjemalec napisan v drugačnem jeziku. Problemi z verzijami se lahko elegantno rešijo z uporabo dodatnih vmesnikov „facets“. Ice objekt ima glavni vmesnik, dodatno pa lahko ponudi več ali nič vmesnikov („facets“), med katerimi lahko izbira.

Poglavje 5

Primerjava vmesnih programskih rešitev

V naslednjih podpoglavjih bom predstavil rezultate študij vmesnih programskih oprem in rezultate testov modela kontrolnega sistema.

5.1 Splošno

V tabeli 5.1 je na enem mestu zbrana relativna primerjava vmesnih programskih rešitev v naslednjih aspektih:

- Ponudniki: koliko proizvajalcev/ponudnikov nudi podporo in razvija tehnologijo.
  - Odprtokodno: veliko število različnih soustvarjalcev, tehnologija je prosto dostopna.
  - Komercialna podpora: obstajajo podjetja, ki ponujajo različno podporo (šolanje, vzdrževanje). Če obstaja samo eno podjetje, je navedeno z imenom. Če podjetje primarno vodi razvoj in je tehnologija odprtokodna, je njeno ime navedeno pred ključno besedo odprtokodno.
  - Lastniška podpora: eno samo podjetje, ki nudi podporo in vzdrževanje. V tem primeru je ime podjetja navedeno.
- Open Source, podprti operacijski sistemi, podprti programski jeziki in podpora Javi: seznam podprtih platform.
- Proces učenja: kako hitro povprečen razvijalec spozna določeno tehnologijo. Oznaka počasno pomeni, da tehnologija zahteva razumevanje kompleksnih konceptov, da je tehnologija težka za uporabo (veliko konfiguracijskih parametrov) in da so učni materiali redki oziroma težko dostopni. Oznaka hitro pomeni, da so koncepti lahko razumljivi, konfiguracija je nezahtevna in da je dokumentacija dostopna in ujemajoča se z dejansko implementacijo.

- Kompleksnost razvoja: primerjava napora, ki ga je potrebno investirati v razvoj posamezne funkcije kontrolnega sistema. Nizka kompleksnost pomeni, da se funkcije zlahka dodajajo (avtomatsko generiranje kode, grafična orodja, ..), visoka pa, da je za funkcionalnost potrebno več korakov, ki si sledijo v togem zaporedju, možne so napake, število vrstic, ki jih mora programer napisati ročno, je večje.

	EPICS CA	**TAO CORBA**	**OmniORB CORBA**	**OpenDDS**	**RTI DDS**	**ZeroC Ice**
Ponudniki	Odprtokodno, komercialna podpora	Odprtokodno, Object Computing, Inc	Odportokodno, Apasphere Ltd.	Odportokodno, Object Computing, Inc	Real-Time Innovations, Inc. (RTI)	ZeroC Labs, odprtokodno
Podprti operacijski sistemi	Windows, Linux, Solaris, Mac OS X, vxWorks, RTEMS, ...	Windows, Linux, Solaris, Mac OS X, HP-UX, VxWorks, ...	Windows, Linux, Solaris, Mac OS X, OpenVMS, HP-UX, ...	Windows, Linux, SunOS, QNX	Windows, Linux, Solaris, VxWorks, LynxOS, QNX	Windows, Linux, Solaris, Mac OS X, SPARC
Java podpora	Da	Ne	Ne	Ne	Da	Ne
Podprti programski jeziki	C/C++, Java 1.4 ali več, Perl, Python	C++	C++, Python	C++	C/C++, Java 1.4 ali več, RTSJ, C#	C++, Java 1.5 ali več, Python, Ruby, PHP, .NET (Windows)
Stroški licenciranja	Prosto	Prosto	Prosto pod LGPL	Prosto	100k EUR do 1M EUR	Prosto pod GPL
Hitrost učenja	Srednje	Počasno	Počasno	Srednje	Hitro	Hitro
Zahtevnost razvoja	Nizka/Srednja	Visoka	Visoka	Visoka	Srednja	Niska/Srednja
Zgodovina	3 desetletja	1 desetletje	1-2 desetletji	2 desetletji	2 desetletji	2 desetletji
Razvojni potencial	Sredji	Srednji	Srednji	Visok	Visok	Visok

Tabela 5.1: Relativna pirmerjava vmesnih programskih oprem (podprti jeziki, platforme, ...)

5.2 Primernost za primere uporabe

Tabela 5.2 povzema primernost posamezne vmesne programske opreme za posamezen primer uporabe. Primernost bom klasificiral po naslednjih kategorijah:

1: sploh ni primerno;

2: primerno, vendar ni optimalna rešitev (znatna degradacija zmogljivosti/kvalitete), potreben poseben dizajn;

3: primerno, vendar obstaja nekaj degradacije zmogljivosti/kvalitete v primerjavi z optimalno rešitvijo, potreben poseben dizajn;

4: primerno, vendar obstaja nekaj degradacije zmogljivosti/kvalitete v primerjavi z optimalno rešitvijo, rešitev že obstaja v sedanjem dizajnu;

5: primerno in blizu idealni rešitvi, rešitev že obstaja v sedanjem dizajnu.

Nekaj primerov uporabe je navedenih glede na zmogljivost (prva številka) in primernost dizajna/koncepta (druga številka). Poleg tega ponekod navajam tudi kontrolni sistem, ki uporablja posamezno vmesno programsko rešitev.

	EPICS CA	TAO CORBA	OmniORB CORBA	RTI DDS	ZeroC Ice
Proženje ukaza	4 2	5 4	5 5	4 3	5 5
Upravljanje dogodkov	4 3	3 4	4 4	5 4	4 5
Spremljanje	5 (EPICS) 5 (EPICS)	4 (ACS) 5 (ACS)	5 (TANGO) 5 (TANGO)	5 3	5 3
Masovni prenos podatkov	4 3	3 4	4 4	5 4	4 4
Diagnosticiranje	5	4	4	5	3
Procesna kontrola	5 (EPICS)	5 (ACS)	5 (TANGO)	4	3
Upravljanje alarmov	5 (EPICS)	5 (ACS)	5 (TANGO)	3	3

Tabela 5.2: Primernost posamezne vmesne programske opreme za primere uporabe.

5.3 Ujemanje splošnih nefunkcionalnih zahtev

V tabeli 5.3 so povzete ugotovitve izpolnjevanja nefunkcionalnih zahtev 2.2.1.

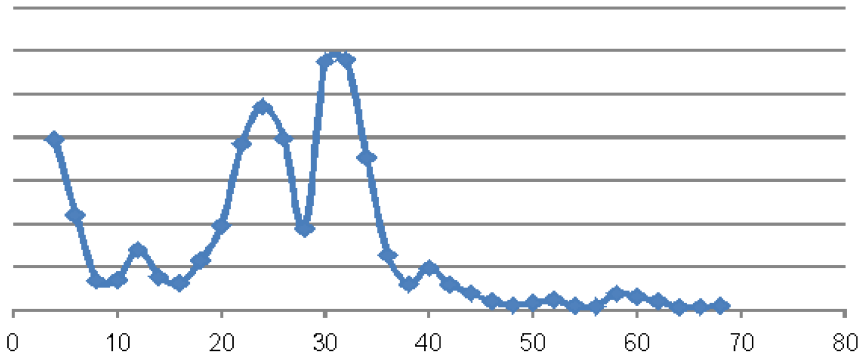
	EPICS CA	**TAO CORBA**	**OmniORB CORBA**	**RTI DDS**	**ZeroC Ice**
Zakasnitev pod 1ms	Da	Da	Da	Da	Da
Izkoroščenost omrežne kapacitete nad 50%	Da	Da	Da	Da	Da
Skalabilnost	Srednja	Nizka	Nizka	Visoka	Nizka
[GNF0]: no single point of failure or bottleneck	Da Posamezni kanal: Ne	Da	Da OmniNotify: Ne	Da	Da IceStorm: Ne
Zanesljiv prenos podatkov	Da (TCP)	Nastavljivo	Nastavljivo	Nastavljivo	Nastavljivo
Prenos po najboljših močeh	Ne	Nastavljivo	Nastavljivo	Nastavljivo	Nastavljivo
Možnost avtentikacije	Da (glede na gostitelj)	Da (SSL)	Da (SSL)	Da	Da
FIFO dostava	Da	Da	Da (ne za enosmerni način)	Nastavljivoe	Nastavljivo
Skupni vrstni red	Ne	Ne	Ne	Ne	Ne
Časovno žigosanje	Da	Ne	Ne	Ne	Ne
Diagnosticiranje	Dobro	Srednje	Srednje	Dobro	Srednje
Kontrola pretoka	Da	Da	Da	Nastavljivo	Da UDP: Ne
Stistkanje podatkov	Ne	Ne	Ne	Ne	Da
Medpomnjenje podatkov	Da	Ne	Ne	Da	Da
Konfigurabilna politika prenosa podatkov preko programskega vmesnika	Da	Da	Da	Da	Da

Tabela 5.3: Ujemanje splošnih nefunkcionalnih zahtev vmesnih programskih oprem.

5.4 Primerjava zmogljivosti

V tem poglavju bom predstavil rezultate testa zakasnitve in hitrosti prenosa. Najprej bom predstavil statistično distribucijo posameznih sporočil pri testu zakasnitve in hitrosti prenosa 3.2.

Sliki 5.1 in 5.2 prikazujeta primer obnašanja vmesne programske opreme OmniORB z enosmernim načinom delovanja. Pri meritvi hitrosti sem meril razliko med dvema zaporednima sporočiloma, kjer so bila sporočila poslana z enim samim pošiljateljem, kakor hitro je bilo mogoče. Za to meritev bi bil pričakovani rezultat Gaussova krivulja z povprečno vrednostjo 1/hitrost obdelave in (idealno) standardno deviacijo 0, vendar se zaradi značilnosti, kot so medpomnjenje in večnitna obdelava zahtev, lahko zgodi, da so sporočila obdelana/sprejeta v skoraj istem času. Prav tako je statistična deviacija zaradi nerealne časovne narave operacijskega sistema in transporta različna od 0.

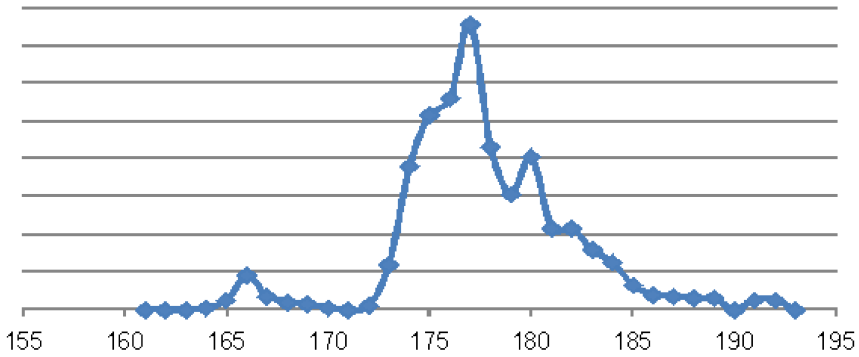


Slika 5.1: Statistična porazdelitev časa med prihodoma dveh sporočil (OmniORB enosmerni način, 800 bajtov koristne vsebine, 10k vzorcev). Čas je podan v mikrosekundah.

Pri testu zakasnitve (slika 5.2) je sporočilo poslano iz enega vozlišča v drugo, to pa odgovori prvemu. Prvo vozlišče nato izmeri čas med časom pošiljanja in prejema. Čas med dvema poslanima sporočiloma je dovolj velik, da se odgovor na poslano sporočilo obdela, še preden se pošlje drugo. Razlika med trenutkom oddaje in sprejema sporočila je obhodni čas (ang. round-trip latency). V idealnem primeru (operacijski sistem v realnem času in deterministično omrežje) bi pričakovali fiksno zakasnitev. V praksi se izkažejo naslednje ugotovitve:

- Zakasnitev ima minimum. Ni mogoče, da ima sporočilo krajšo zakasnitev od določenega minimalnega časa. Ta minimalni čas je dosežen, kadar ni nobene zakasnitve zaradi čakalne vrste, omrežnega sklada ali omrežja.

- Zakasnitev ima poljubno dolg rep. Maksimalna zakasnitev se lahko določi le v redkih



Slika 5.2: Statistična porazdelitev obhodnega časa/zakasnitve (OmniORB navadni način, 800 bajtov koristne vsebine, 1k vzorcev). Čas je v mikrosekundah.

primerih (sistem v realnem času). Zato je bolj smiselno meriti 95 percentil, kot pa dolžino repa.

Tabela 5.4 povzema zmogljivost posamezne vmesne programske opreme. Detajli glede zmogljivosti in skalabilnosti so v nadaljevanju tega poglavja in poglavja 5.5 (Opombe glede skalabilnosti).

	EPICS CA	**omniORB CORBA**	**RTI DDS**	**ZeroC Ice**
Enosmerna zakasnitev (min/povprečje/95th percentil v μs); 1 izvor, 1 ponor, 25 bajtov koristne vsebine		47/55/63	80/101/119	79/85/92
	128/156/185	OmniNotify: 157/175/187		IceStorm:165/220/231
Maksimalna prepustnost (izkoriščenost pasovne širine v procentih); 1 izvor, 1 ponor		90%	100%	90%
	69%	OmniNotify: 56%		IceStorm: 57%
Skalabilnos ob številu ponorov (degredacija prepustnosti ob povečevanju števila ponorov)	Proporcionalno z številom izvorov (multiplication of sources possible through CA gateway). Inverzno proporcionalno z številom ponorov.	Inverzno proporcionalno z številom odjemalcev serverja. OmniNotify: inverzno proporcionalno z številom ponorov.	Zelo majhna degredacija.	Inverzno proporcionalno z številom odjemalcev serverja. IceStorm: inverzno proporcionalno z številom ponorov.

Tabela 5.4: Performančna primerjava vmesnih programskih oprem.

V primeru CA (ang. Channel Access) testno okolje vsebuje EPICS bazo, kar je razlog za merjenje zmogljivosti EPICS-a kot celote. Ker predstavlja baza EPICS suboptimalno arhitekturo (rabi centraliziran servis za odpremljanje sporočil), so rezultati bolj primerljivi s pridobljenimi pri OmniNotify in IceStorm.

Tabela 5.5 prikazuje zaporedje komunikacijskih tehnologij od najboljše do najslabše v terminih kot so zakasnitev, zmogljivost prenosa in skalabilnost.

	Zakasnitev	**Prenosna hitrost**	**Skalabilnost**
Najboljše	OmniOrb	RTI DDS	RTI DDS
2	Ice	OmniORB (oneway)	EPICS/CA
3	RTI DDS	Ice (oneway)	Ice
4	EPICS/CA	EPICS/CA	OmniORB
5	OmniNotify	IceStorm	IceStorm
Najslabše	IceStorm	OmniNotify	OmniNotify

Tabela 5.5: Zaporedje komunikacijskih tehnologij od najboljše do najslabše.

Slika 5.3 prikazuje zmogljivost posamezne vmesne programske opreme kot funkcijo velikosti sporočila. Zmogljivost je izražena kot relativna primerjava zmogljivosti (v procentih), dosežene z UDP transportom. V mojem testnem okolju nisem mogel doseči prenosne hitrosti 1Gbps, čeprav so mrežne kartice in stikalo to omogočali. Ozko grlo je bil procesor, ki je bil polno zaseden z UDP/IP skladom.

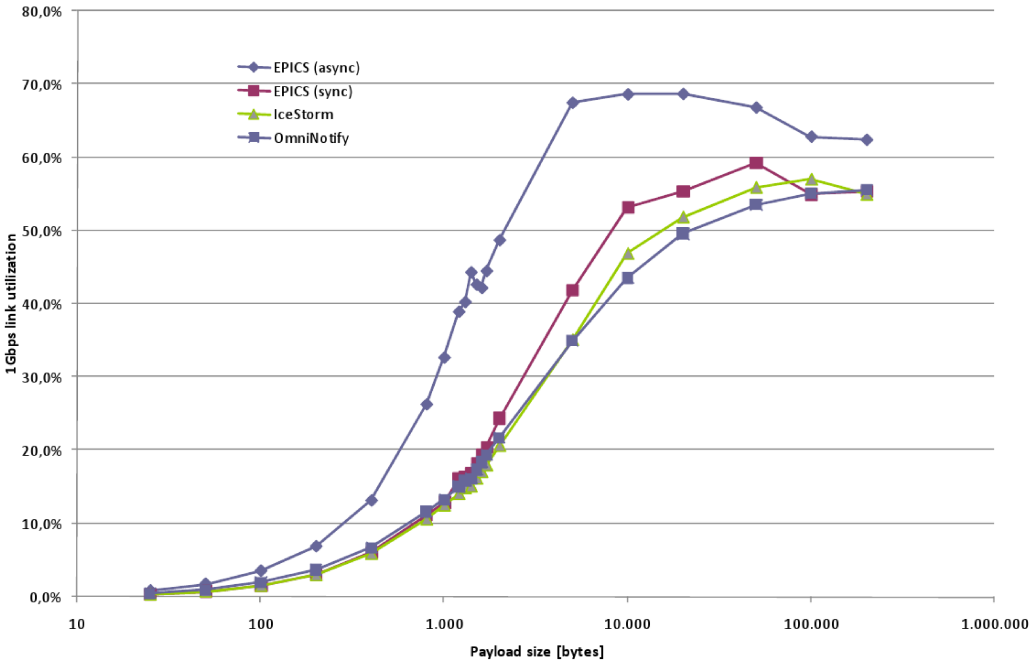
Po pričakovanjih velika sporočila veliko bolje izkoristijo pasovno širino kot manjša, ki privedejo do višjih režijskih stroškov. V primeru EPICS (asinhrono pošiljanje podatkov – naslednje sporočilo se pošlje brez čakanja, da centralni strežnik obdela prejšnje) prenosna hitrost pri velikih sporočilih pada, kajti EPICS baza postane ozko grlo, saj mora procesirati zapise velikih velikosti.

Slika 5.4 prikazuje prenosno hitrost kot funkcijo velikosti sporočila za vmesne programske opreme, ki za razpošiljanje podatkov ne potrebujejo centraliziranega strežnika. Uporabljen je bil prenos po najboljših močeh („oneway“ semantika). Prenosna hitrost je pričakovano večja, ker ni posrednika med pošiljateljem in sprejemnikom. RTI DDS recimo doseže 100% zasedenost (v primerjavi z UDP prenosom).

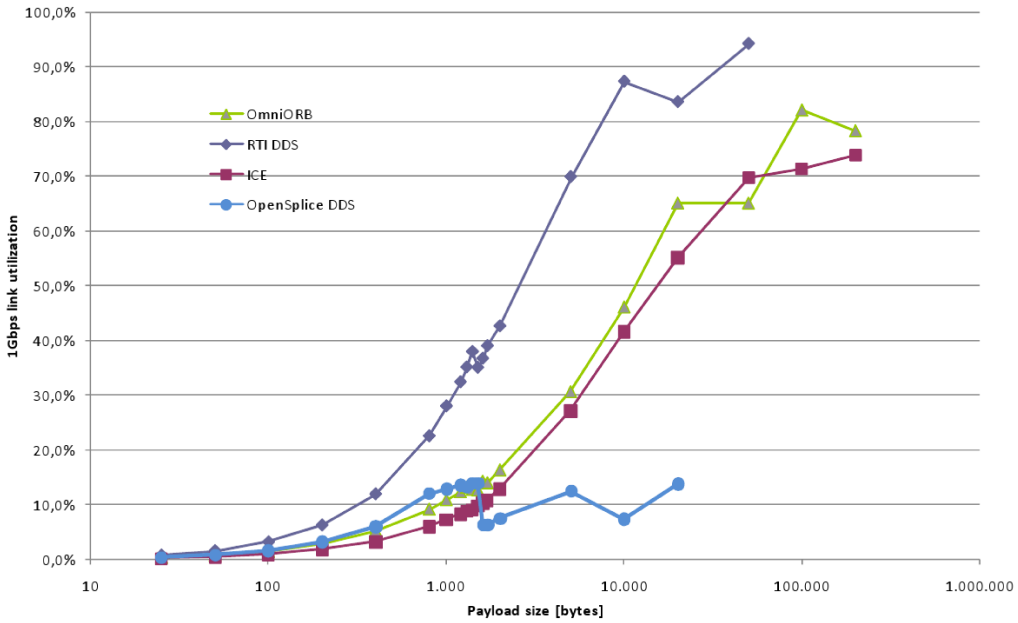
Vzrok nepovezanosti krivulje pri sporočilih velikosti 1.5kB je maksimalna dolžina Ethernet datagrama (1514 bajtov). Padec prenosne hitrosti v primeru RTI DDS pri sporočilih okoli 100kB se je vedno ponovil. Ugotovil sem, da je RTI DDS izredno prilagodljiv, tako da dopuščam možnost, da se padec hitrosti pri kakšni drugačni konfiguraciji ali velikosti sporočil ne bi zgodil.

Privzet način prenosa pri OmniORB in Ice je zanesljivi prenos (ang. non-oneway). V tem primeru prenosna hitrost pade, kot je prikazano na naslednji sliki. RTI DDS prav tako omogoča zanesljiv prenos večim sprejemnikom hkrati (ang. multicast). Za manjša sporočila se RTI DDS obnese bolje.

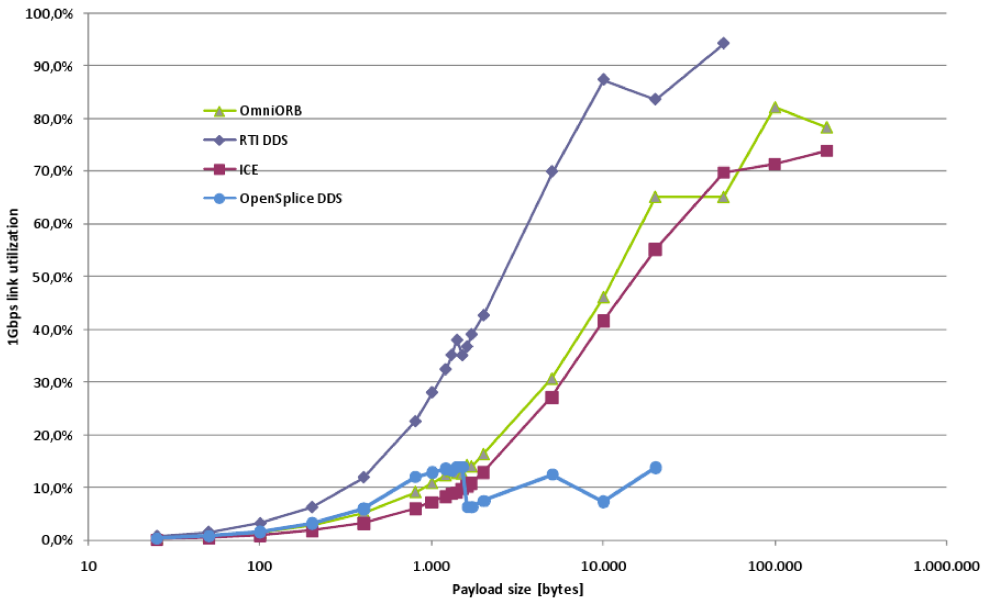
Slika 5.6 prikazuje rezultate meritve zakasnitve kot funkcijo velikosti sporočila. Za vsako vmesno programsko opremo, ki ne vsebuje centraliziranega strežnika za razpošiljanje sporočil, so prikazane 3 krivulje: minimum, povprečje in 95 percentil zakasnitve. Zakasnitev je izračunana iz obhodnega časa. Vmesne programske opreme so bile nastavljene



Slika 5.3: Prenosna hitrost vmesnih programskih oprem, ki **vsebujejo** centraliziran servis za razpošiljanje podatkov.

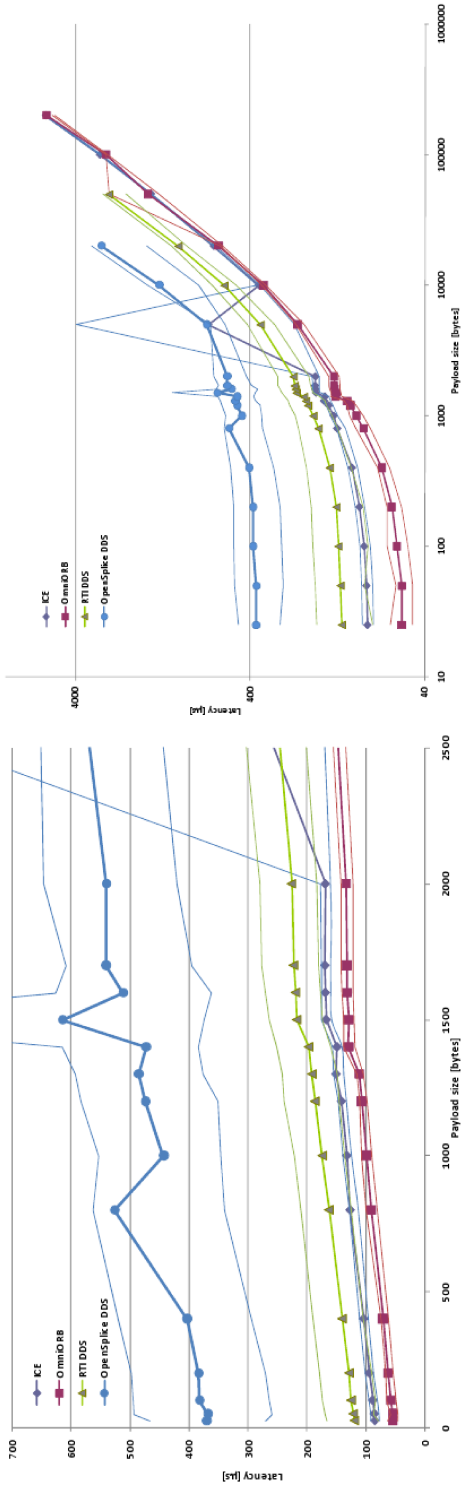


Slika 5.4: Prenosna hitrost vmesnih programskih oprem, ki **ne** vsebujejo centraliziran servis za razpošiljanje podatkov (**prenos po najboljših močeh**).

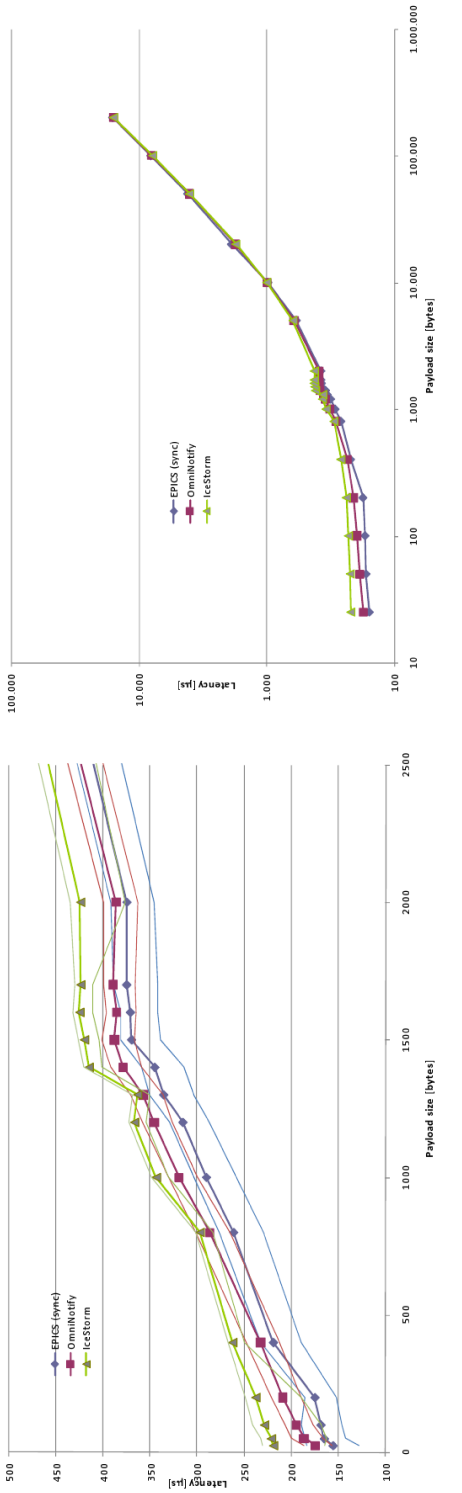


Slika 5.5: Prenosna hitrost vmesnih programskih oprem, ki **ne** vsebujejo centraliziranega servisa za razpošiljanje podatkov (**zanesljiv prenos**).

za zanesliv prenos. Pri majhnih sporočilih (leva stran slike) na zakasnitev primarno vpliva učinkovitost vmesne programske opreme (serializacija/deserializacija, večnitnost, ...). Pri velikih sporočilih (desna stran slike) zakasnitev raste približno linearno z velikostjo sporočil in ni velike (relativne) razlike med vmesnimi programskimi opremami. To je tudi pričakovano, saj je vzrok za večino zakasnitve čas, potreben za prenos sporočila po mreži. Desna stran slike kaže, da ima Ice težave pri sporočilih velikosti 5kB – rep distribucije zakasnitve je za 2 reda večji, kot bi pričakovali.



Slika 5.6: Zakasnitev komunikacijskih tehnologij, ki ne vsebujejo centraliziranega strežnika za razpošiljaje sporočil.



Slika 5.7: Zakasnitev komunikacijskih tehnologij, ki vsebujejo centraliziran strežnik za razpošiljaje sporočil.

Slika 5.7 prikazuje rezultat meritve zakasnitve kot funkcijo velikosti sporočila. Za vsako tehnologijo, ki ima centraliziran strežnik za razpošiljaje sporočil, so prikazane 3 krivulje: minimum, povprečje in 95 percentil. Pri majhnih sporočilih (leva stran slike) je zakasnitev primarno odvisna od učinkovitosti posamezne tehnologije (serializacija/deserializacija, upravljanje s procesi, ...). Pri velikih sporočilih (desna slika) zakasnitev raste približno linearno z velikostjo sporočil in ni velike (relativne) razlike med posamezno tehnologijo. To je pričakovano, ker je ta čas odvisen od prenosne hitrosti, ki je omejena z pasovno širino.

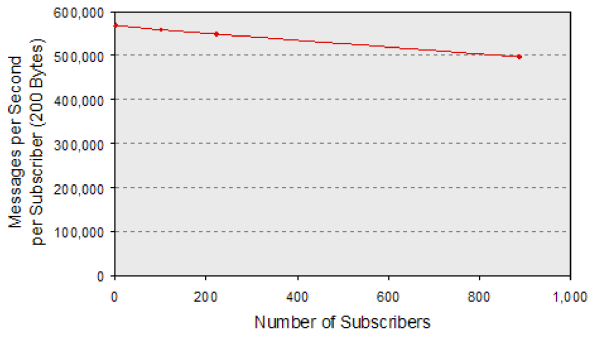
5.5 Opombe glede skalabilnosti

V tem podpoglavju bom opisal, kako prizadeta je zmogljivost ([e posebno maksimalna dosežena hitrost), ko število pošiljateljev in sprejemnikov pri posamezni vmesni programski opremi raste.

Ker je test skalabilnosti težko izvedljiv v testnemu okolju, ki sem ga imel na razpolago, sem predvidel skalabilnost glede na arhitekturo in iz rezultatov testov kot so [5] in [19].

Najboljšo skalabilnost vmesne programske opreme zagotavlja **RTI DDS**, ker nudi možnost pošiljanja sporočil večim sprejemnikom hkrati (ang. multicast). S tem pristopom usmerjevalniki, ki se dobro zavedajo IP protokola, poskrbijo, da so ista sporočila med usmerjevalniki poslana samo enkrat, le-ti pa prav tako učinkovito prenesejo sporočila na več povezav hkrati.

Glede na rezultate testov (slika 5.8), ki jih je izvedlo podjetje RTI, število sporočil, ki jih lahko pošiljalec pošlje in so sprejeta pri vsakem registriranem naročniku, počasi pada, hitrost prenosa pa pade za komaj 13% pri 888 naročnikih. Ti podatki so bili dobljeni z zanesljivim hkratnim prenosom (ang. reliable multicast). Moja ugibanja so, da bi bil v primeru hkratnega nezanesljivega prenosa padec hitrosti še manjši.



Slika 5.8: Pretok kot funkcija števila prejemnikov (Vir: [5]).

EPICS/CA ponuja prav tako visok nivo skalabilnosti, kar dokazuje dejstvo, da največje ekperimentalne naprave uporabljajo EPICS kontrolni sistem, kjer je prisotnih

deset do več sto tisoč zapisov (ang. record), kljub temu pa ni zabeleženih večjih skalabilnostnih problemov.

V EPICS-u je največji skalabilnostni problem v številu odjemalcev, ki jih server lahko streže (oziroma število prijavljenih na zapis pri istem gostitelju). Težava je v tem, da mora server poslati sporočilo vsakemu odjemalcu posebej (ne podpira hkratnega pošiljanja sporočil), kar pomeni, da če se število odjemalcev poveča za faktor 2, se prenosna hitrost zmanša za isti faktor (v polno zasedenem sistemu).

Število TCP povezav lahko prav tako postane izredno veliko. Če imamo npr. N-odjemalcev in M-strežnikov, ki gostijo P-zapisov, in vsak odjemalec dostopa do vseh rekordov, potem obstaja NxM povezav (faktor P je prihranjen, ker CA uporabi isto povezavo za več parov odjemalec-zapis). V tipični aplikaciji odjemalec dostopa do razmeroma malo zapisov (reda velikosti 10), zaradi česar zelo redko pride do problemov. Če pa se le-ti pojavijo (veliko število odjemalcev dostopa do istega/istih zapisov), se lahko uporabi CA prehod (ang. CA gateway), ki omogoča replikacijo zapisov na drugih gostiteljih.

Ice in **OmniORB** sledita RPC (ang. Remote Procedure Call) semantiki, zato je pomembno vprašanje glede skalabilnosti, koliko odjemalcev lahko zadovolji server.

Z Ice in OmniORB je možno narediti skalabilnostne topologije z vmesnimi prehodi, ki pa niso na razpolago in jih je potrebno razviti za vsako aplikacijo posebej.

Najmanj skalabilni so - mogoče paradoksalno - dogodkovni servisi (ang. notification/event service) kot recimo **IceStorm** in **OmniNotify**. Test, ki ga je izvedlo podjetje ZeroC [19], pokaže, da je celotno število sporočil/dogodkov, ki so lahko poslani preko servisa, konstantno. S povečanjem števila odjemalcev se hitrost prenosa manjša.

Poglavje 6

Izbira vmesne programske rešitve

V tem poglavju bom izbral po mojem mnenju najprimernejšo vmesno programsko opremo in navedel tudi razloge, zakaj sem se odločil zanjo. Seveda bom predstavil tudi alternative tej izbiri.

Kljub temu da nobena tehnologija zahtevam in primerom uporabe, ki so bili identificirani med analizo zahtev, ne ustreza popolnoma, je ne moremo označiti kot neprimerno, saj vse ustrezajo primerom uporabe, čeprav rešitev ni direktna.

Z namenom, da se prihrani napor in čas pri gradnji kontrolnega sistema, se mi zdi najprimernejša izbira ene od tehnologij, s katerimi so že zgrajeni kontrolni sistemi.

Po mojem mnenju je najprimernejša tehnologija **EPICS/CA**, saj EPICS ponuja veliko število orodij za gradnjo kontrolnega sistema. Prav tako EPICS/CA vsebuje dobro definiran programski vmesnik, ki se izredno redko spreminja, in podporo različnim napravam.

EPICS je primerljiv z OmniORB in RTI DDS, vendar je robustnejši in manj problematičen. Ima pa tudi nekaj slabosti:

- Če se potrebe po skalabilnosti izredno visoke (veliko odjemalcev povezanih na en sam strežnik, zahteva po izredno majhnih zakasnitvah, kar ne dovoljuje vmesnega CA prehoda), se zna zgoditi, da jih EPICS/CA ne bo mogel zadovoljiti. Edina vmesna programska oprema, ki nima teh težav, je RTI DDS. Če se pojavi ta problem, so možne sledeče rešitve:
  - Če je protokol po najboljših močeh zadosten, potem se lahko doda funkcionalnost hkratnega pošiljanja sporočil (ang. multicast) v CA protokol (relativno enostavno).
  - Če se terja zanesljiv prenos, potem je mogoče dodati tudi zanesljivo hkratno pošiljanje. Ob prezahtevni implementaciji se lahko CA zamenja za RTI DDS kot prenosno plast za EPICS bazo.
- Oddaljeno proženje metod je slabo podprto z EPICS/CA. Pri tem problemu se lahko razvije orodje, ki generira proxyje in stubs (serializacije/deserializacija).

- Poimenovanje procesnih spremenljivk je enolično, ne odraža organizacije in ne vsiljuje modela naprave.

Druga najprimernejša tehnologija je **OmniORB** oziroma infrastruktura kontrolnega sistema, ki bazira na OmniOrb, kot sta npr. TANGO ali CERN LHC. Slabost te vmesne programske opreme je manjša uporabniška baza in manjša zrelost kontrolnih sistemov, ki so že zgrajeni okrog teh tehnologij. Vendar je oddaljeno proženje metod veliko bolje podprto.

RTI DDS bi lahko bila zmagovalna tehnologija, kar se tiče skalabilnosti in podpore zanesljivega prenosa, vendar ne ponuja nobene specifične storitve ali orodja glede kontrolnih sistemov. Bazirano na izkušnjah drugih kontrolnih orodij (EPICS, TANGO) bi razvoj takih orodij lahko trajal tudi več let.

Načeloma bi lahko uporabljali več kot le eno vmesno programsko opremo kot prenosno plast kontrolnega sistema, vendar se bi pri tem pojavile vsaj 3 težave:

- Odločitev, kateri del funkcionalnosti implementirati v eni tehnologiji in kateri v drugi.
- Natančno poznavanje vseh tehnologij.
- Večja podpora in težje vzdrževanje vseh tehnologij, s tem pa x-krat višji stroški.

Poglavje 7

Sklepne ugotovitve

Študij vmesnih programskih oprem in kontrolnega sistema CODAC zna biti precej problematičen. Dokumentacija je ponavadi izredno obsezna - specifikacije kontrolnega sistema obsegajo več tisoč strani in so v nakaterih delih pomankljive ali celo nedoločene. Vse vmesne programske opreme sem preučil in jih preizkusil v praksi ter s tem pridobil detajlno poznavanje le-teh.

Žal mi je, da nisem uspel narediti tudi testov skalabilnosti, saj bi bilo prav zanimivo videti, kako se posamezna vmesna programska oprema obnese pri zelo velikem številu odjemalcev/strežnikov. Tako sem lahko podal samo pavšalne ocene in sem se moral zanesti na druge vire.

Pravo presenečenje je bila vmesna programska oprema Open Splice, saj se je proti pričakovanjem izredno slabo obnesla. Očitno je, da tehnologija še ni zrela za uporabo v aplikacijah z zelo visokimi zahtevami. Pri Open Splice-u mi je bila izredno všeč arhitektura same vmesne programske opreme in enostavnost uporabe v primerjavi z RTI DDS (obe vmesni programski opremi ustrezata objavi/naroči paradigmi). Ravno med časom pisanja diplomskega dela je izšla nova verzija in prav zanimivo bi bilo narediti primerjavo med staro in novo verzijo (verzija 3 proti verziji 4), kajti po besedah proizvajalca se je učinkovitost povečala za več kot 30%.

Ravno tako je v času pisanja sklepnih ugotovitev ITER razglasil, da bo za vmesno programsko opremo izbral EPICS (samo za del kontrolnega sistema), zato bo zanimivo spremljati njegov nadaljni razvoj (vloženo bo veliko denarja).

Literatura

[1] "CODAC Conceptual Design Overview," ITER, apr. 2008.

[2] Pujara, H. et al, "MODEX Task Report," ITER, apr. 2008.

[3] "Plant Control Design Handbook," ITER, jul. 2008.

[4] "Plant System Interface," ITER, 2008.

[5] (2009) RTI, Throughput and Scalability Benchmarks C++ on Linux. Dostopno na: http://www.rti.com/products/dds/benchmarks-cpp-linux.html

[6] (2009) Advance Photon Source: EPICS CA. Dostopno na: http://www.aps.anl.gov/epics/

[7] (2009) Object Management Group: OMG Specifications. Dostopno na: http://www.omg.org/

[8] (2008) Object Management Group: Common Object Request Broker Architecture (CORBA) Specification, Verzija 3.1. Dostopno na: http://www.omg.org/spec/CORBA/3.1/

[9] (2009) C. McHale, CORBA Explained Simply. Dostopno na: http://www.ciaranmchale.com/corba-explained-simply/index.html

[10] P. Gore, R. Cytron, D. Schmidt, C. O'Ryan, "Designing and Optimizing a Scalable CORBA Notification Service," v zborniku Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems, Utah, United States, avg. 2001, str. 196-204.

[11] (2009) Object Computing, Inc, The ACE ORB (TAO). Dostopno na: Object Computing, Inc

[12] (2009) D. Grisby, omniORB. Dostopno na: http://omniorb.sourceforge.net/

[13] (2007) Object Management Group: Data Distribution Service (DDS) for Real-time Systems Specification, Verzija 1.2. Dostopno na:
http://www.omg.org/spec/DDS/1.2/

[14] (2008) Object Management Group: The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification, Verzija 2.0. Dostopno na:
http://www.omg.org/spec/DDSI/2.0/

[15] (2009) Object Computing, Inc: OpenDDS. Dostopno na:
http://www.opendds.org/

[16] (2009) Real-Time Innovations, Inc: RTI DDS. Dostopno na:
http://www.rti.com/

[17] (2009) OpenSplice DDS Overview. Dostopno na:
http://www.prismtech.com/section-item.asp?id=558

[18] (2009) ZeroC Labs: ZeroC Ice. Dostopno na:
http://www.zeroc.com/

[19] (2009) ZeroC Labs: Event Distribution Service Tests. Dostopno na:
http://www.zeroc.com/performance/eventService.html

[20] (2009) Use case. Dostopno na:
http://en.wikipedia.org/wiki/Use_case

[21] (2009) ITER. Dostopno na:
http://www.iter.org