Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Anže Hrast

Zmogljivostna analiza programske rešitve »SEP2W System« za avtomatsko odčitavanje porabljene energije

DIPLOMSKA NALOGA
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2008

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Anže Hrast

Zmogljivostna analiza programske rešitve »SEP2W System« za avtomatsko odčitavanje porabljene energije

DIPLOMSKA NALOGA
NA UNIVERZITETNEM ŠTUDIJU

izr. prof. dr. Miha Mraz
MENTOR

Ljubljana, 2008

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

ZAHVALA

Zahvaljujem se mentorju za njegovo hitro in temeljito delo. Posebej se zahvaljujem tudi mentorju v podjetju Iskraemeco d. d., Juretu Germovšku, ki je odgovorjal na moja vprašanje glede ustroja sistema SEP. Zahvaljujem se tudi vsem ostalim, ki so kakorkoli pomagali pri nastajanju te diplomske naloge.

— Anže Hrast, Ljubljana, oktober 2008.

SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

- **SEP** (Sistem elektronskega postrojenja): programski paket, ki omogoča oddaljeno branje in upravljanje z merilci energije na terenu.
- **AMM** (angl. Advanced Meter Management): Sistem, v katerem je omogočeno oddaljeno branje in upravljanje merilnih naprav na terenu.
- **AMR** (angl. Automatic Meter Reading): Sistem, v katerem je omogočeno oddaljeno branje merilnih naprav na terenu.
- **PDA** (angl. Personal Digital Assistant): Ročni računalnik.
- **GSM** (angl. Global System for Mobiles): Eden izmed standardov za mobilno telefonijo.
- **GPRS** (angl. General Packets Radio Service): Storitev, namenjena uporabnikom GSM omrežja, ki omogoča pošiljanje in prejemanje podatkov v obliki paketov.
- **PLC** (angl. Power Line Communication): Sistem za prenos podatkov po električnih vodnikih.
- **DLC** (angl. Distribution Line Carrier): Sistem za prenos podatkov po električnih vodih srednje, nizke in hišne napetosti.
- **PSTN** (angl. Public Switched Telephone Network): Mreža vseh svetovnih javno dostopnih telefonskih omrežij. Vključuje tako fiksno kot mobilno telefonijo.
- **FTP** (angl. File Transfer Protocol): Protokol, namenjen izmenjavi datotek preko omrežja.

- **MSMQ** (angl. Microsoft Message Queueing): Mikrosoftov protokol za medprocesno komunikacijo, ki poteka preko več heterogenih omrežij in sistemov, med katerimi so lahko nekateri tudi izklopljeni.
- **WMI** (angl. Windows Management Instrumentation): Vmesnik operacijskega sistema Windows, preko katerega uporabnik dostopa do informacij o računalniku.
- **WBEM** (angl. Web Based Enterprise Management): Množica tehnologij, razvitih z namenom poenotenja nadzora nad porazdeljenimi sistemi.
- **CIM** (angl. Common Information Model): Odprt standard, ki definira predstavitev elementov, nad katerimi se lahko vrši nadzor. Poleg tega definira tudi razmerja med temi objekti.
- **EDF** (fr. Électricité de France): Francosko elektrodistribucijsko podjetje.
- **XML** (angl. eXtensible Markup Language): Specifikacija, ki uporabnikom omogoča definiranje poljubnih opisnih jezikov. Ti so namenjeni izmenjavanju podatkov preko komunikacijskih poti.
- **COSEM** (angl. Companion Specification for Energy Metering): Množica specifikacij, ki določajo prenosno in aplikacijsko plast DLMS protokola.
- **DLMS** (angl. Device Language Message Specification): Zbirka standardov s področja komunikacije z merilnimi napravami.

KAZALO

POVZETEK

Odčitavanje porabe energije je v veliki večini ročno početje. Avtomatično odčitavanje si sicer utira pot v elektrodistribucijska podjetja, vendar s počasnim tempom. Situacija se lahko v naslednjih nekaj letih povsem spremeni in podjetja bodo pričela množično menjati infrastrukturo. Podjetje Iskraemeco d. d. je razvilo tak sistem za avtomatsko odčitavanje in upravljanje z infrastrukturo, ki naj bi bil sposoben uporavljati z 1.000.000 števci na terenu. Do pred kratkim je bil sistem v fazi razvoja; v tem času se je večino truda vlagalo v funkcionalnost. Performančne težave so se reševale, če so se pokazale pri testiranju funkcionalnosti. Sedaj pa se produkt pričenja tržiti in SEP je bilo potrebno testirati tudi z zmogljivostnega vidika. V ta namen je bilo vzpostavljeno testno okolje, ki je čim bolj posnemalo realne razmere, v okviru tega okolja pa so se izvajali testi. Težave smo pričakovali na komunikacijskih poteh in pri shranjevanju podatkov, tekom testov pa so se pokazala še druga ozka grla v sistemu. Najbolj pereč problem sistema so nejgove zahteve po pomnilniku. Za vsa ozka grla sicer obstajajo rešitve, vendar so nekatere za sistem bolj primerne kot druge.

Ključne besede: Sistem elektronskega postrojenja, Automatic Meter Reading, Advanced Metering Management

ABSTRACT

Monitoring energy consumption is still largely accomplished manually. Automatic monitoring is gradually appearing in electricity distribution companies, but this process is slow. However, in the next few years the situation can change completely and so companies will be changing their infrastructure. Iskraemeco d. d. company has developed a system for automatic reading and management of metering infrastructure that can supposedly handle up to 1,000,000 meters. Until recently this system was under development and, during that time, all efforts were concentrated on its functionality. Performance problems that occurred during functionality testing were solved. Since SEP recently hit the market, tests concerning performance issues had to be performed. For this purpose, it was established a test environment that mimicked real-life conditions as closely as possible, where tests were conducted. Problems were to occur with communication paths and while storing data. During the testing process other bottlenecks also appeared. The most concerning one was the system demand for memory. There are solutions to be found for all the bottlenecks, but some are more appropriate for the system than the other.

Key words: Sistem elektronskega postrojenja, Automatic Meter Reading, Advanced Metering Management

1 Uvod

V procesu ekonomske rasti podjetja se je že davno tega izkazalo, da je človek počasen, sploh pri rutinskem delu. Proizvodni procesi so tako večinoma robotizirani, zaposlene v storitvenih procesih pa se da v veliki meri zamenjati z računalnikom in programsko opremo. Poglejte si samo spletne trgovine. Množica kupcev, edini zaposleni pa delajo v skladišču.

Pri elektrodistribucijskih podjetjih ni stanje nič drugačno. Radi bi preverjali dnevno porabo energije pri vseh odjemalcih, a bi bilo v tem trenutku to mogoče doseči le z veliko zaposlenimi ljudmi, ki bi hodili od števca do števca, zapisovali stanja na njih in ta stanja vpisovali v neko centralno bazo podatkov. Sama zamisel, da bi se ta proces odvijal avtomatično, je zelo verjetno precej stara, vendar se je začela udejanjati šele zadnjih 10 let. Majhni primerki, veliki do nekako 30.000 števcev, takih avtomatičnih sistemov že obstajajo, na terenu pa se uporabljajo tudi rešitve, ki jih ponuja podjetje Iskraemeco d. d. Kako pa se bodo te rešitve obnesle pri množičnem prehodu na avtomatizacijo, ko bo eno elektrodistribucijsko podjetje hotelo nadzirati 10.000.000 števcev?

Trenutno komercialno dostopne rešitve podjetja Iskraemeco d. d. taki količini strojne

opreme niso kos. Zaradi tega dejstva je že dve leti v razvoju nova rešitev, ki naj bi zmogla upravljanje s sistemom, ki ga sestavlja 1.000.000 števcev. Če zmore kaj več, odlično, če pa tega cilja ne doseže, potem je treba problem najti in ga popraviti. Pričujoče delo se ukvarja prav s tem problemom. Odgovarja na vprašanje, ali je novi sistem zmožen upravljanja z 1.000.000 števci na terenu in če je odgovor nikalen, poizkuša najti probleme, ki zmogljivosti tega sistema znižujejo.

V drugem poglavju je najprej podrobneje predstavljena arhitektura elektrodistribucijskega omrežja in možnosti, ki jih ta arhitektura, predvsem s področja komunikacij, ponuja. Potem pa je opisan sistem podjetja Iskraemeco, ki s to arhitekturo upravlja.

V tretjem poglavju so najprej podrobneje opisani mehanizmi, s katerimi lahko merimo zmogljivost sistema. V nadaljevanju je predstavljeno pričakovano breme, kamor sodita struktura podatkov, ki se prenašajo od števcev proti nadzornemu sistemu, in pričakovana količina podatkov. Na koncu tega poglavja pa je opisano okolje, v katerem sem izvajal testiranja.

Četrto poglavje je namenjeno obdelavi s testiranjem pridobljenih podatkov. Tu so opisana ozka grla, ki so se pokazala pri testiranjih, predstavljeni so morebitni vzroki za te težave in njihove potencialne rešitve.

V petem poglavju pa pod vsem napisanim potegnemo črto v obliki zaključka.

2 Opis sistema

2.1 Celoten sistem

Do pred kratkim so elektrodistribucijska podjetja električno energijo zaračunavala napol pavšalno. Na podlagi porabe iz preteklega leta so vsak mesec poslala položnico z nekim zneskom. Enkrat na leto so uslužbenci podjetja pri vseh naročnikih izvedli popis števcev, podjetje pa je izvedlo ustrezne popravke. Malo novejši način je, da stranka vsak mesec pokliče distribucijsko podjetje ter sporoči stanje na števcu, podjetje pa ta podatek uporabi pri vsakomesečnem obračunavanju. Vendar ta storitev uporabnika nekaj stane, distribucijsko podjetje pa mora še vedno pošiljati zaposlene na teren. Strankam vseeno ne gre popolnoma zaupati.

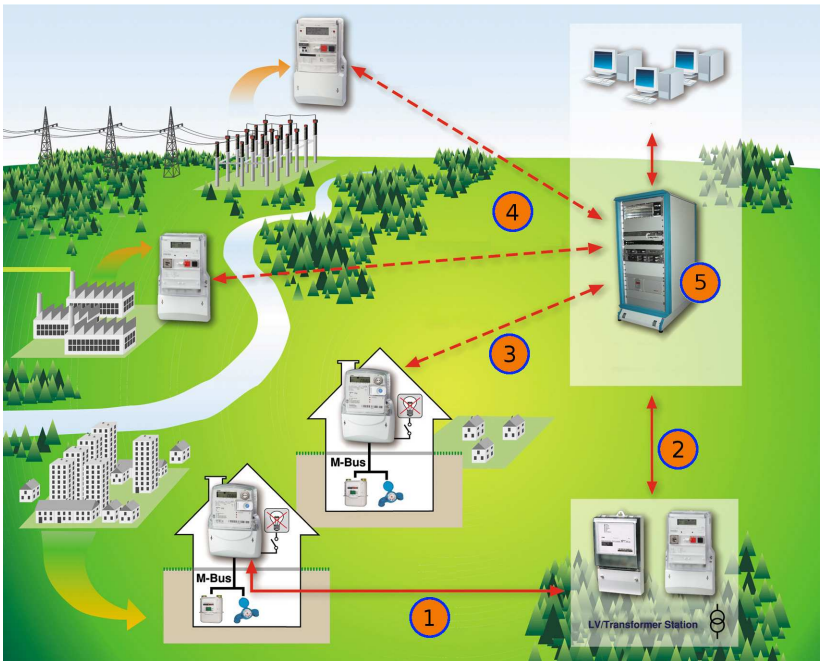
Preverjanje stanja števca na terenu je počasno, drago, povrhu vsega pa lahko tudi nevarno (nekega čitalca v ameriškem mestu Corpus Christi, Texas, je med opravljanjem dolžnosti napadel pes). Poleg tega so se v industriji pojavile zahteve o dnevnem ali celo urnem branju števcev. In ideja o sistemu, ki je zadolžen za avtomatično zbiranje podatkov s števcev (električnih, vodnih, plinskih), hranjenje in obdelavo pridobljenih podatkov, je bila rojena. Sistem se imenuje AMR (angl. *Automatic Meter Reading*), vključuje pa tako

tehnologije branja podatkov na daljavo, kot tudi branje podatkov neposredno s števca.

Znotraj AMR sistema je lahko uporabljenih veliko tehnologij:

- Ročno branje: Ta način se pravzaprav niti ne razlikuje od klasičnega popisovanja števcev. Uslužbenec podjetja neko napravo (prenosni računalnik, PDA ...) z uporabo kabla poveže s števcem in sproži njegovo branje.
- Oddaljeno branje:
  - Z ročnimi napravami: precej podobno ročnemu branju, le da tu med napravo za zbiranje podatkov in števcem komunikacija poteka preko brezžične povezave;
  - Preko fiksnega omrežja: v tem primeru se za prenos podatkov uporablja že vzpostavljeno ali posebej v ta namen vzpostavljeno omrežje. Izkoristi se lahko brezžični ethernet, GSM in GPRS omrežja, PLC komunikacijo, razno kombiniranje omrežij ... Topologija omrežja je najpogosteje zvezdasta. Več števcev je tako ali drugače v navezi s koncentratorjem, ki podatke s števcev zbira in jih posreduje zbirnemu centru.

Kako izgleda AMR sistem v praksi, nam pokaže slika 2.1:



Slika 2.1 AMR sistem v praksi.

Komunikacija med števcem in koncentratorjem se vrši z DLC tehnologijo (točka 1 na sliki 2.1), koncentrator pa se lahko s centrom sistema pogovarja na več načinov (točka 2 na sliki 2.1). Komunikacija lahko poteka preko GPRS ali GSM omrežja, mogoče je uporabiti infrastrukturo klasične telefonije, možno je celo povezovanje preko ethernet omrežja.

Števci lahko s centrom komunicirajo tudi neposredno (točka 3 na sliki 2.1). Seveda mora biti števec za to komunikacijo pripravljen, recimo tako, da vsebuje GPS modul.

Števci težke kategorije (industrijski, precezijski) so za komunikacijo s centrom dobro pripravljeni (točka 4 na sliki 2.1). Komunicirajo lahko preko telefonskih linij, omrežij mobilnih operaterjev, PSTN omrežja, pa še kakšen način bi se našel.

Korak naprej je AMM ali Automatic Meter Management. AMM sistem poleg AMR funkcionalnosti predvideva še upravljanje z električnim omrežjem. S tem sistemom je mogoče realizirati predplačniško porabo energije, omogočeno je ugotavljanje napak na terenu, nekatere izmed napak je mogoče odpraviti ...

Nekam v to shemo spada SEP (kaj točno SEP je, je opisano v naslednjem poglavju). Bralcu bi se sicer že lahko svitalo, kam točno spada. SEP spada v center vsega dogajanja, torej na strežnik, ki ga na sliki 2.1 predstavlja točka 5. Ta strežnik bo v središču naše pozornosti. V to točko se stekajo vsi podatki, iz te točke izvirajo vsa navodila napravam na terenu.

2.2 SEP2W.NET

Naj najprej povem, kaj pomeni ime SEP. Kratica je akronim za Sistem elektronske postrojenosti, izvira pa še iz časov samoupravljanja in je celo avtorsko zaščitena. Prvi SEP je bil spisan za okolje DOS, naslednji SEPW pa že za operacijski sistem Windows. Naslednja inkarnacija SEP-a se je imenovala SEP2W (druga verzija za Winodows okolje), najnovejša pa se imenuje SEP2W .NET. Tisti .NET pomeni, da je zasnovan na .NET ogrodju in bi bilo bolj pravilno, da se imenuje SEP3W. Ampak pri poimenovanju je treba tudi nekoliko marketinško razmišljati. V nadaljevanju bom uporabljal oznako SEP in s tem mislil na verzijo SEP2W .NET. Moduli so spet poimenovani po nekoliko drugačnem vzorcu, vendar naj to bralca ne zavede. Pisanje bo zadevalo samo SEP2W .NET verzijo sistema.

Povsem na kratko, SEP je implementacija AMM sistema. Sestavljen je iz več modu-

lov, da so načrtovanje, implementacija in vzdrževanje čim lažji. Moduli so glede na svojo vlogo ter izpostavljenost organizirani v štiri sklope.

Prvi sklop z imenom *Perimiter* na sliki 2.2 sestavljajo komponente, ki komunicirajo z merilnimi napravami, sistemi zunanjih strank in internetnimi uporabniki. Da bi minimizirali možne varnostne težave, se morajo komponente na tem nivoju podrejati varnostni politiki sistema SEP, kar pomeni, da storitve na tem nivoju nimajo direktnega dostopa do SEP2 podatkovne baze, lahko pa komunicirajo s storitvami srednjega nivoja.

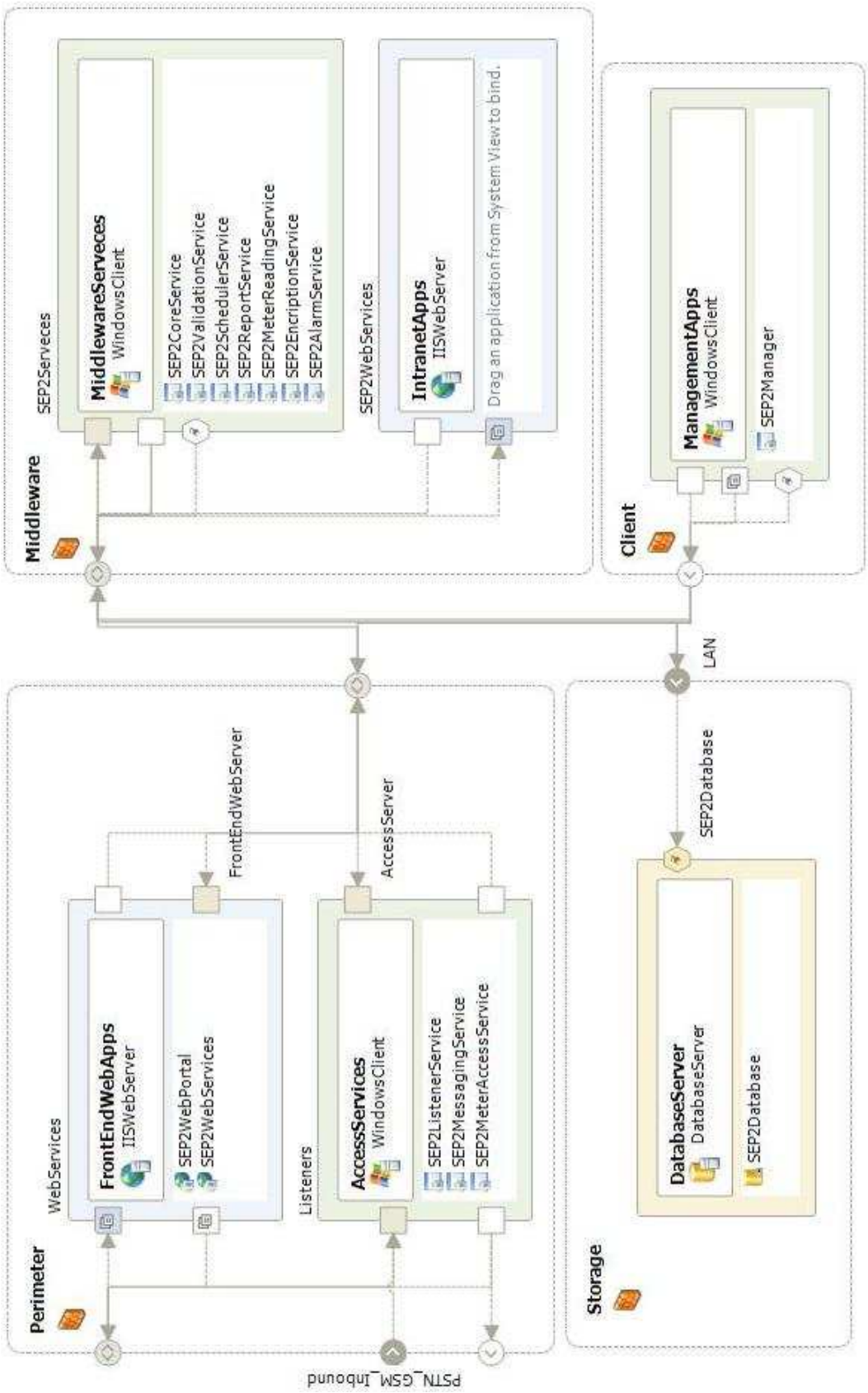
Komponente te sekcije so:

- SEP2 Meter Access Service: preko njega se vrši celoten dostop do naprav v AMM sistemu. Dostop se opravlja preko GPRS omrežja, serijskih vrat ali PSNT/GSM modemov.
- SEP2 Listener Service: ta modul prejema sporočila, ki jih naprave pošiljajo centru AMR sistema. Uporablja modeme GSM in omrežje GPRS.
- SEP2 Messaging Service: uporablja FTP, elektronsko pošto, datotečni sistem in sporočilne vrste MSMQ za pošiljanje oziroma sprejemanje sistemskih podatkov.
- SEP2 Web Service: predstavlja zunanji vmesnik v obliki spletnih storitev, ki omogoča integracijo sistema SEP2W v zunanje sisteme strank.
- SEP2 Web Portal: predstavlja spletni portal namenjen končnim odjemalcem. Omogoča pregled nad obstoječo, trenutno in predvideno porabo ...

Drugemu sklopu, ki predstavlja tako imenovani srednji sloj ali *Middleware*, bi lahko rekli tudi jedro sistema. V njem se nahajajo moduli, ki omogočajo komunikacijo s komponentami prvega sklopa in vršijo dostop do SEP2 podatkovne baze.

Moduli srednjega sloja so:

- SEP2 Meter Reading Service: generira zahteve za branje naprav z uporabo podatkov o napravah shranjenih v SEP2 podatkovni bazi, jih posreduje SEP2 Meter Access strežnikom in pridobljene podatke shrani nazaj v SEP2 podatkovno bazo.
- SEP2 Alarm Service: interpretira alarme, ki jih sporoča SEP2 Listener Service, jih shrani v SEP2 podatkovno bazo in obvesti administratorja sistema oziroma stranko o morebitnih težavah.



Slika 2.2 Zgradba SEP sistema.

- SEP2 Validation Service: preveri pridobljene podatke z uporabo predhodno definiranih testov. Preverjene podatke ustrezno označi in pripravi poročila o uspehu, na podlagi katerih se odločimo o morebitnem ročnem preverjanju podatkov.
- SEP2 Report Service: izdeluje uporabniško definirana poročila, bazirana na podatkih v SEP2 podatkovni bazi. Poročila se lahko shranijo v SEP2 podatkovno bazo, na datotečni sistem, se prikažejo na ekranu ...
- SEP2 Scheduler Service: izvaja naloge v sistemu SEP v skladu z definiranimi urniki. Naloga predstavlja delovni tok, sestavljen iz opravil organiziranih v drevesno strukturo. Posamezno opravilo predstavlja operacijo izvršeno v enem izmed sistemskih modulov (SEP2 Meter Reading Service, SEP2 Validation Service, SEP2 Report Service ali SEP2 Messaging Service).
- SEP2 Core Service: opravlja nadzor nad licencami v sistemu, upravlja s pravicami uporabnikov in storitev ter implementira podporne funkcije za SEP2 Web Service in SEP2 Web Portal modula.

Tretji sklop z imenom *Storage* predstavlja sistem za hrambo podatkov. V sistemu SEP se praktično vsi podatki hranijo v SEP2 podatkovni bazi. Ti podatki so:

- podatki o topologiji merilnega omrežja,
- podatki o merilnih rezultatih,
- podatki o dogodkih v sistemu,
- podatki o urnikih, po katerih se izvajajo naloge v sistemu,
- podatki o opravilih, ki sestavljajo naloge ...

SEP2 podatkovna baza sestoji iz več ohlapno povezanih tabel, kar omogoča hranjenje podatkov na enem ali več podatkovnih strežnikih. V prvem podatkovnem strežniku se na primer lahko hranijo podatki o merilnem sistemu in porabi, na drugem pa informacije o strankah, izdelana poročila ... Ne glede na to, ali so podatki porazdeljeni med več strežnikov ali ne, jih uporabnik sistema vedno vidi kot celoto.

Zadnji sklop modulov z imenom *Client*, omogoča nadzor nad sistemom SEP. Aplikacije preko drugega sklopa dostopajo do podatkovne baze ter do povezav v svet. Program SEP2 Manager deluje kot lupina, v kateri prebivajo moduli, namenjeni upravljanju s

posameznimi deli sistema (SEP2 Meter Reading Plugin je tesno povezan z SEP2 Meter Reading Service modulom, SEP2 Database Plugin pa omogoča urejanje SEP2 podatkovne baze).

Ko je bil sistem začrtan, ga je bilo potrebno še realizirati. Programiranje poteka v programskem jeziku C#, kot razvojno orodje pa je uporabljen Microsoftov Visual Studio 2005, oba konstrukta pa se naslanjata na .NET 2.0 tehnologijo. Kar se tiče uporabniškega vmesnika, smo (predvsem zaradi uporabe knjižnic Krypton Toolkit in CodeJock) omejeni na Windows okolje, medtem ko naj bi se komponente prvega in drugega sklopa prevedle tudi za operacijski sistem Linux. V zgodnji fazi se je namreč programiralo tako, da so se moduli prevedli tudi z ogrodjem MONO, kar nam je samodejno prineslo podporo ravnokar omenjenemu operacijskemu sistemu. Vendar prevajanja v Linux okolju ni nihče izvedel že vsaj leto in pol. Analiza modulov z orodjem MoMA, ki že prevedeno kodo pregleda, ali je skladna z MONO ogrodjem, pa razkrije, da na modul obstaja vsaj 50 klicev .NET funkcij, ki v MONO okolju še niso implementirane, ali še huje, da so uporabljeni sistemski klici, ki so specifični za operacijski sistem Windows.

3 Pristop k zmogljivostni analizi

Cilj pričujočega dela je izvedba analize zmogljivosti sistema SEP. Način, na katerega so se podatki o zmogljivosti zbirali, sta določali dve okoliščini. Prva je ta, da sistem teče v okolju Windows, druga pa je dostopnost izvorne kode. Operacijski sistem Windows nam daje na voljo mehanizme, s katerimi je mogoče meriti učinkovitost sistema, izvorna koda pa omogoča preprosto vklapljanje teh mehanizmov v sistem.

3.1 Zmogljivostni kazalniki

Mehanizmi za nadzorovanje učinkovitosti sistema se imenujejo kazalniki učinkovitosti (angl. *Performance Counters*). To so programski monitorji [12]. Razdeljeni so v več kategorij, vsak kazalnik pa ima lahko več instanc. Na primer v kategoriji ≫Network Interface≪ najdemo kazalnik ≫Current Bandwidth≪, ta pa se pojavlja v toliko instancah, kolikor je v sistemu mrežnih vmesnikov.

Dostop do kazalnikov v operacijskem sistemu je s sistemskega vidika nekonsisten-ten. Do nekaterih kazalnikov je mogoče dostopati le preko knjižnice ADVAPI32.DLL, do drugih pa samo preko WMI (Windows Management Instrumentation) vmesnika, ki im-

plementira WBEM (Web-Based Enterprise Management) in CIM (Common Information Model) standarda, spet tretji so dostopni preko obeh vmesnikov. Poleg tega je mogoče uporabiti še knjižnico PDH.DLL, s katero se imena nekaterih kategorij in kazalnikov prevedejo v lokalni jezik. Na srečo obstaja ogrodje .NET, ki vse kazalnike spravi pod skupno streho in programerju olajša njihovo uporabo.

3.2 SEP2W zmogljivostni kazalniki

Svoje kategorije in kazalnike lahko uporabnik po potrebi dodaja v sistem. Seveda je nekaj kazalnikov zašlo tudi v SEP2W sistem. Trenutno ima svoje kazalnike le nekaj komponent sistema, vendar pa so to komponente, ki so pri delu z infrastrukturo na terenu najbolj obremenjene. Kazalniki so razdeljeni v sledeče kategorije:

- SEP2W Database Counters,
- SEP2W Scheduler Scheduled Jobs,
- SEP2WScheduler Task,
- SEP2W Scheduling in
- SEP2 ThreadPool Counters.

3.2.1 SEP2W Database Counters

Ti kazalniki nadzorujejo SEP-ovo interakcijo z bazo. Res je, da z vsako pošteno bazo dobimo zvrhano mero kazalnikov, ki se ukvarjajo z nadzorovanjem delovanja tiste baze, vendar nam tisti kazalniki ne morejo povedati, kateri del od nadzorovanega je SEP-ov. Kazalniki te kategorije so naslednji:

- **Active transactions:** Število transakcij v izvajanju.
- **Opened connections:** Število trenutno vzpostavljenih povezav z bazo.
- **Transactions/sec:** Število izvršenih transakcij v sekundi.

3.2.2 SEP2W Scheduler Scheduled Job, SEP2W Scheduler task

Sledeči dve kategoriji kazalnikov sta lastni modulu SEP2W Scheduler. Malo se ustavimo pri pojmih, ki jih bomo uporabljali. Opravilo (angl. *job*) je drevo nalog (angl. *task*).

Vsako opravilo ima natanko eno korensko nalogo, vsaka naloga pa ima pod sabo lahko več drugih nalog. Naloge se vedno izvajajo za neko množico objektov. Podnaloge se izvajajo glede na uspeh ali neuspeh izvajanja starševske naloge. Pa si te kazalnike oglejmo:

- SEP2W Scheduler Scheduled Job:
  - **Avg execution time:** Povprečni čas izvajanja opravila v milisekundah.
  - **Avg scheduling objects per job.** Povprečno število objektov v opravilu.
  - **Jobs compleeted:** Število izvedenih opravil.
  - **Jobs executing:** Število trenutno izvajajočih opravil.
  - **Max execution time:** Najdaljši čas izvajanja opravila v milisekundah.
  - **Max scheduling objects per job:** Največje število objektov v opravilu.
  - **Min execution time:** Najkrajši čas izvajanja opravila v milisekundah.
  - **Min scheduling objects per job:** Najmanjše število objektov v opravilu.
- SEP2W Scheduler Task:
  - **Max tasks succeeded per hour:** Največje število nalog na uro.
  - **Max tasks succeeded per hour time:** Čas, ko je bilo doseženo največje število nalog na uro.
  - **Max tasks succeeded per minute:** Največje število nalog na minuto.
  - **Max tasks succeeded per minute time:** Čas, ko je bilo doseženo največje število nalog na minuto.
  - **Tasks executing:** Število trenutno izvajajočih se nalog.
  - **Tasks failed:** Število neuspešno izvedenih nalog.
  - **Tasks faield in last hour:** Število neuspešno izvedenih nalog v zadnji uri.
  - **Tasks faield in last minute:** Število neuspešno izvedenih nalog v zadnji minuti.
  - **Tasks succeeded:** Število uspešno izvedenih nalog.
  - **Tasks succeeded in last hour:** Število uspešno izvedenih nalog v zadnji uri.
  - **Tasks succeeded in last minute:** Število uspešno izvedenih nalog v zadnji minuti.

3.2.3 SEP2W Scheduling

Naslednja kategorija je namenjena modulom SEP2W MeterReading in SEP2W Report. Zakaj je temu tako? Če hočemo, da se z modulom da upravljati s SEP2W Scheduler modulom, morajo ti moduli implementirati nek Scheduler-jev vmesnik, del tega vmesnika pa so tudi naslednji kazalniki:

- **Objects aborted:** Število objektov, za katere je bilo izvajanje prekinjeno.
- **Objects canceled:** Število objektov, za katere je bilo izvajanje preklicano.
- **Objects executing:** Število objektov, ki se trenutno izvajajo.
- **Objects failed:** Število neuspešno izvedenih objektov.
- **Objects succeeded:** Število uspešno izvedenih objektov.
- **Objects waiting:** Število čakajočih objektov.
- **Tasks aborted:** Število prekinjenih nalog.
- **Tasks avg execution time:** Povprečen čas izvajanja naloge v milisekundah.
- **Tasks avg waiting time:** Povprečen čas čakanja naloge v milisekundah.
- **Tasks canceled:** Število preklicanih nalog.
- **Tasks executing:** Trenutno število nalog v izvajanju.
- **Tasks failed:** Število neuspešno izvedenih nalog.
- **Tasks max execution time:** Najdaljši čas izvajanja naloge v milisekundah.
- **Tasks max waiting time:** Najdaljši čas čakanja naloge v milisekundah.
- **Tasks min execution time:** Najkrajši čas izvajanja naloge v milisekundah.
- **Tasks min waiting time:** Najkrajši čas čakanja naloge v milisekundah.
- **Tasks succeeded:** Število uspešno izvedenih nalog.
- **Tasks waiting:** Trenutno število čakajočih nalog.

3.2.4 SEP2 ThreadPool Counters

SEP, ali pa vsaj nekateri njegovi moduli, uporablja svoj sistem za upravljanje z večnitnostjo. .NET-ov vmesnik ima namreč neke pomanjkljivosti, ki jih SEP-ov odpravlja, poleg tega pa nam omogoča vpogled v stanje čakalnih vrst in še nekaj malega statistik. V tem sistemu nit ostane živa še nekaj časa po tem, ko je opravila svoje delo. Ko pride do zahteve po novi niti, ki se v tem kontekstu imenuje naloga, sistem poišče prosto nit in ji nalogo dodeli. Če proste niti ni, se ustvari nova. Na ta način prihranimo nekaj časa, ki se sicer porabi za ustvarjanje nove niti. Kazalniki te kategorije so naslednji:

- **Active threads:** Število niti v sistemu, ki svoje naloge izvajajo.
- **Avg. work item process time/sec:** Povprečen čas izvajanja naloge.
- **Avg. work item wait time/sec:** Povprečen čas, ki ga naloge preždijo v čakalni vrsti.
- **Running threads:** Število razpoložljivih niti v sistemu.
- **Work items:** Število nalog v sistemu; tako tistih v izvajanju, kot tistih v čakalnih vrstah.
- **Work Items in queue:** Število nalog, ki čakajo na izvajanje v čakalni vrsti.
- **Work items processed:** Število dokončanih nalog.
- **Work items processes/sec:** Število dokončanih nalog v sekundi.
- **Work items queued/sec:** Število nalog, ki so prišle v čakalno vrsto v sekundi.

Seznam je torej kar obsežen. Zraven bi lahko dodal še kazalnike, ki jih vsebuje SEP2W Core Service. Teh pa je za malo enciklopedijo, saj nam dajejo podatke o poizvedbah po posameznih tabelah v bazi, poleg tega pa Core Service pri komuniciranju z infrastrukturo sodeluje le toliko, da preveri uporabnikove privilegije ter licenco po kateri sistem deluje.

Na žalost ti kazalniki nekaj povejo le o zanesljivosti sistema, ne pa toliko o njegovi zmogljivosti. Če hočemo meriti obremenitev, ki jo za računalniški sistem predstavlja programski paket SEP2W, moramo uporabiti kazalnike, ki jih ponuja operacijski sistem. Ti kazalniki nam omogočajo vpogled v zasedenost procesorja, potrebah po delovnem pomnilniku in vhodno/izhodnih napravah (diski, mrežni vmesniki).

3.3 Branje števcev

Morda se neposvečenemu človeku zdi to nekoliko trivialno vprašanje. Naj program nadomesti človeka in enkrat na mesec prebere trenutno porabo. Od te porabe naj odšteje porabo prejšnjega meseca, rezultat pa je tisto, kar se znajde na položnici. Ljudem v gospodinjstvih se kaj več ne zdi smiselno delati.

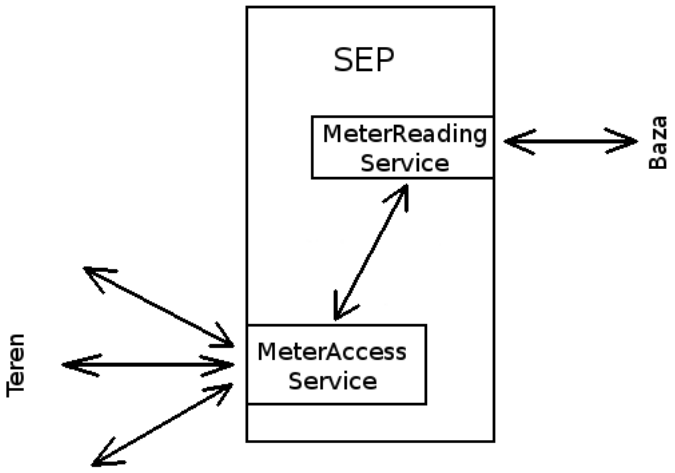
Drugače je s podjetji, ki z energijo uporavljajo. Tam zelo radi vedo, kako se giblje dnevna poraba ali poraba znotraj enega dneva. Električni števci so za to že pripravljeni. Porabo energije vsakih 15 minut zapišejo v pomnilnik (kjer se nahajajo rezultati predhodnih meritev), od koder jih lahko koncentrator ali skrbnik električnega omrežja na zahtevo dobita. Poleg tega števec pomni tudi vse pomembnejše dogodke (ugašanje ali prižiganje števca). Take zbirke podatkov imenujemo *profili*. Poleg teh profilov lahko beremo še posamezne registre števca, v katerih so zapisane trenutne vrednosti porabe.

Kaj je torej problem? Recimo, da smo elektro distribucijsko podjetje, ki hoče precej natančno vedeti, koliko energije porabijo vse stranke. Brali bomo torej 15 minutni profil. Odločiti se moramo samo še, kako pogosto ga bomo brali. Sodelavci predlagajo, naj se podatki berejo vsako uro, vsak dan ali pa vsak teden. Lahko bi brali tudi vsak mesec ali še bolj poredko, ampak potem s podatki nismo več na tekočem. Odločimo se za branje vsak dan, med drugim tudi zato, ker koncentrator bere podatke enkrat na dan. Nato se odločimo, kaj bomo brali. Beremo lahko uro na števcu (sinhronizacija), profile energije (aktivne in reaktivne), dnevnik dogodkov ... Odločimo se, da bomo brali glavne (in tako tudi časovno in prostorsko najbolj potratne) podatke, torej profil aktivne energije. Sedaj pa že lahko naredimo nekaj izračunov (nekaj stvari moramo še vedno predpostaviti, ampak ni tako zelo hudo).

Recimo, da imamo na terenu 1.000.000 števcev. Brali bomo 15 minutni profil, vsak zapis v profilu je dolg 26 B (če koga zanima, kako je sestavljen sam zapis, lahko preštudira standard COSEM), zapisov v enem dnevu pa je 96. V enem dnevu bomo torej prebrali 2,32 GB podatkov. Dostop do koncentratorjev podjetja Iskraemeco d.d. se vrši na dva načina. Starejši je FTP dostop, novejši pa je dostop preko Web Service-ov, s tem da se FTP dostop opušča. XML datoteke, na katerih Web Service-i slonijo, pa generirajo ogromno kontrolnih podatkov. No, ti niti niso tako zelo problematični, če se koncentratorji nahajajo na ethernet omrežju. Povsem drugačno sliko dobimo, če je koncentrator priključen na ISDN (zelo počasna komunikacija) ali GPRS (zelo draga komunikacija)

omrežje.

In tu se pojavi prvi problem, ki ga mora sistem SEP2W rešiti: Kako dovolj hitro prebrati vse podatke? Kot smo že namignili, preko ethernet omrežja (vsaj danes, ko 100 Mb omrežje ni nobena redkost in ko se po svetu vzpostavlja optično omrežje) hitrost prenosa ni problem. Ovira je odzivnost koncentratorja. Izmerili smo, da od trenutka, ko MeterAccessService pošlje zahtevo po branju profila enega števca, pa do trenutka, ko prejme rezultat, preteče v povprečju 40 sekund. Za našo količino števcev to pomeni (če bi se vsi števci brali eden za drugim) 463 dni. Seveda se da to branje zelo paralerizirati. Že MeterAccessService je sposoben brati 50 naprav naenkrat, torej branje traja le še 9 dni. Če v sistem postavimo več MeterAccessService-ov, pa smo ta problem že rešili. Drugi način za pohitritev te operacije je pohitritev koncentratorja. Z novo verzijo WebService-ov, ki je trenutno v testiranju, je mogoče profil dobiti tudi v 2 sekundah.



Slika 3.1 Prenos podatkov znotraj sistema SEP.

No, končno pride do teh 2,32 GB podatkov tudi SEP2W. Naprej to množico podatkov prebere MeterAccess Service, prebrane podatke posreduje MeterReading Service-u, ta pa mora te podatke zapisati v bazo. To ne bi smel biti problem, glede na to, da ima dandanes večina računalnikov v drobovju 2 GB delovnega spomina. Vendar MeterAccess Service in MeterReading Service komunicirata preko XML datotek, te pa poleg podatkov meritev vsebujejo še malo morje kontrolnih podatkov. Koliko točno je teh podatkov? Datoteka, ki vsebuje 15 minutni profil za 100 števcev, je velika 61 MB. Za miljon števcev se torej nabere 595 GB podatkov. Kaj storiti s to količino podatkov? Pri prenosu podatkov med

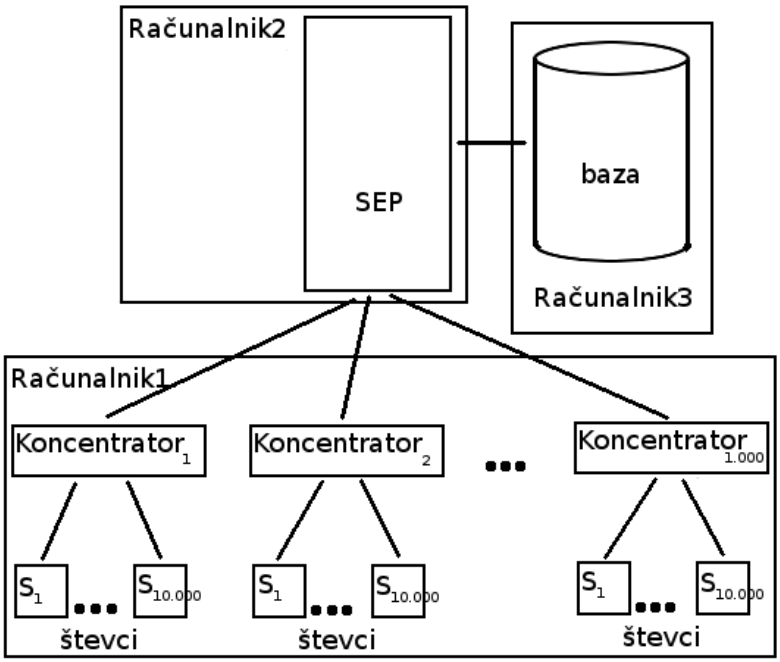
servisi sodeluje kompresija, s katero nekoliko zmanjšamo količino prometa. Kaj pa se zgodi, ko je potrebno to količino podatkov obdelati v servisu samem?

In končno MeterReading Service vse prejete podatke zapiše v bazo. Zopet, če gre samo za vpisovanje rezultatov v bazo, do hujših problemov ne sme priti. Lahko pa se zgodi, da se v istem trenutku podatki vpisujejo v bazo, hkrati pa uporabnik išče podatke o pretekli porabi, SEP-ovi servisi pa zahtevajo podatke o napravah, da bodo lahko začeli novo branje. V tem primeru zelo pomaga pametno razbitje baze na več računalnikov.

3.4 Testni poligon

Osebno kar veliko časa, ki ga porabim za pregledovanje vsebine interneta, namenim branju testov zmogljivosti računalniških komponent. Tak test, vsaj v mojih očeh, hitro izgubi na kredibilnosti, če testnega okolja ne opiše dobro. Tako v opisu testa pričakujem, da bom izvedel osnovno konfiguracijo sistema, se pravi v računalnik nameščene komponente, kateri gonilniki so bili uporabljeni v testih, v kakšnem stanju je operacijski sistem in tako naprej. Ne preostane mi torej drugega, kot da v skladu z lastnimi merili opišem okolje, v katerem so se testi izvajali.

Za lažjo predstavitev besedila v nadaljevanju prilagam sliko.



Slika 3.2 Največja možna konfiguracija poligona.

Koncentratorje in števce sem simuliral z internim orodjem. To orodje omogoča simuliranje do 1.000 koncentratorjev, na vsak koncentrator pa lahko priključimo 10.000 števcev, potreboval pa sem le delček simulatorjeve zmogljivosti. Pripravil sem si nekaj različnih konfiguracij koncentratorjev in števcev, ki pa so se razlikovale le po številu enot posamezne vrste. Najmanjša konfiguracija je bila sestavljena iz 10-ih koncentratorjev, na vsakega pa je bilo priklopljenih 100 števcev. Da ne bom na dolgo in široko opisoval nekaj tako kratko opisljivega, kot je število koncentratorjev in števcev, predlagam sledeče poimenovanje konfiguracij. Oznaka 10x100 naj pomeni konfiguracijo 10 koncentratorjev, na vsakega od katerih je priklopljenih 100 števcev. Preostale sestavljene konfiguracije bi torej lahko označil z 10x1000, 100x100, 100x1000, 1000x100 in 1000x1000. Zakaj take vrednosti? O številu koncentratorjev ne gre izgubljati besed. Na terenu jih je pač toliko, kolikor jih je. Koliko je števcev na koncentratorju pa je drugo vprašanje. 100 je število števcev, ob katerem na primer koncentrator podjetja Iskraemeco d.d. še kolikor toliko stabilno deluje. 1.000 števcev na koncentrator pa je EDF-ova zahteva. Simulator števcev in konventratorjev je tekel na posebnem računalniku (Računalnik 1 na sliki 3.4) in ni zasedal takorekoč nobenih sistemskih virov. Razlog za odločitev, da simulator teče na posebnem računalniku, je moja želja po analizi obremenjenosti komunikacijskih poti. Operacijski sistem Windows namreč prometa, ki ga ustvari mrežna komunikacija (pa čeprav ta komunikacija poteka le znotraj računalnika), ne prikaže. Mimogrede, nekaj vrstic višje sem omenil podjetje EDF, ki preko razpisov stalno dviguje zahteve, katerim naj bi zadoščal njimov AMM. Zakaj bi kaj takega sploh upoštevali? Podjetja, ki ima v lasti 58 jedrskih elektrarn, ki proizvede 20 % električne energije v Evropski uniji, ki račune pošilja 35.600.000 odjemalcem in ki ima namen v naslednjih nekaj letih povsem preiti na AMM sistem, pač ne moreš kar tako prezreti.

Moduli sistema SEP (SEP2Core Service, SEP2Scheduler Service, SEP2 MeterReading Service in SEP2MeterAccess Service) so tekli na drugem računalniku (Računalnik 2 na sliki 3.4). Ta računalnik je performančno sicer precej pod raznimi strežniškimi zverinami, pa vendar kar spodoben stroj. Oglejmo si ga podrobneje:

- HP matična plošča, osnovana na Intel Q35 čipu,
- Core 2 duo E8400 @ 3GHz processor,
- 2x2 GiB, DDR2 – 800 pomnilnik,

- Hitachi HDP725050GLA360 ATA, 500 GB, SATA disk,
- ATI RadeonHD 2400XT grafični vmesnik,
- Intel 82566DM-2 Gigabit Network Connection mrežni vmesnik,
- Windows Vista Busines SP1, 32 bit operacijski sistem.

Malce pozornosti velja nameniti sami strukturi SEP2MeterAccess Service modula. Sestavljen je iz jedra in gonilnikov, ki skrbijo za komunikacijo z napravami samimi. Tako obstajajo gonilniki za komunikacijo z raznimi števci kot tudi za komunikacijo s koncentratorji. V tem testu je bil uporabljen le gonilnik, ki skrbi za komunikacijo s koncentratorjem preko WebService-ov (obstaja namreč tudi gonilnik, ki se s koncentratorjem pogovarja preko FTP protokola).

Na tretjem računalniku (Računalnik 3 na sliki 3.4) je bila postavljena baza. Natančneje, postavljeni sta bili dve bazi, in sicer Microsoft-ova SqlExpress in IBM-ova DB2 baza. SqlExpress baza je na voljo zastonj preko spleta, vendar je velikost baze omejena na velikost 4 GB. Toliko pa v konfiguracijah 1000x100 in 1000x1000 zasedejo samo podatki o napravah, tako da sem za te baze uporabil prav tako prosto dostopno DB2 bazo. Sam računalnik, na katerem je baza tekla, pa je bil daleč od kakršnega koli spodobnega sistema, ki ga sicer najdemo kot gostitelja velikih baz. Hočete podatke?

- IBM matična plošča osnovana na Intel 828 čipu,
- Pentium 4 @ 3 GHz, HyperThreading procesor,
- 1 GiB pomnilnik,
- Samsung SP0802N, ST340014A diska,
- ATI Radeon 9200 SE grafični vmesnik,
- LevelOne 32/64-bit Gigabit Ethernet Adapter mrežni vmesnik,
- Windows XP professional SP3, 32 bit operacijski sistem.

Operacijski sistem in prostor na disku sta bila zamazana z vso mogočo šaro. Kaj boljšega mi v podjetju niti v testne namene niso namenili. Tudi prav.

Zajemanje podatkov sem opravil z Microsoftovim Reliability and Performance Monitor orodjem. Ta program omogoča tako pregledovanje kot logiranje vseh kazalnikov

zmogljivosti, ki se na nekem računalniku nahajajo. Vzorci so se zbirali vsakih 10 sekund, spremljal pa sem naslednje kazalnike:

- iz kategorije "Process" za instance kazalnikov "SEP2MeterReadingService", "Sep2MeterAccessService" in "SEP2SchedulerService":
  - "% Processor Time"
  - "Private Bytes"
- iz kategorije "Network Interface" za edini mrežni vmesnik v računalniku:
  - "Bytes Sent/sec"
  - "Bytes Received/sec"

4 Rezultati analize

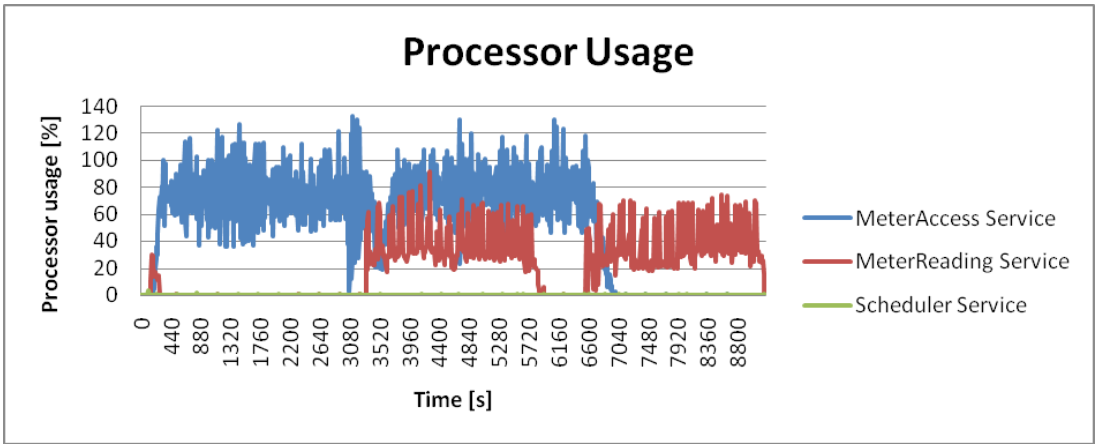
Surovih podatkov, dobljenih iz teh testov, je zelo veliko, zato se mi zdi samo navajanje leteh nesmiselno. Tako bom najprej navedel nekaj splošnih ugotovitev za vse konfiguracije, podrobneje pa analiziral samo eno.

4.1 Splošne ugotovitve

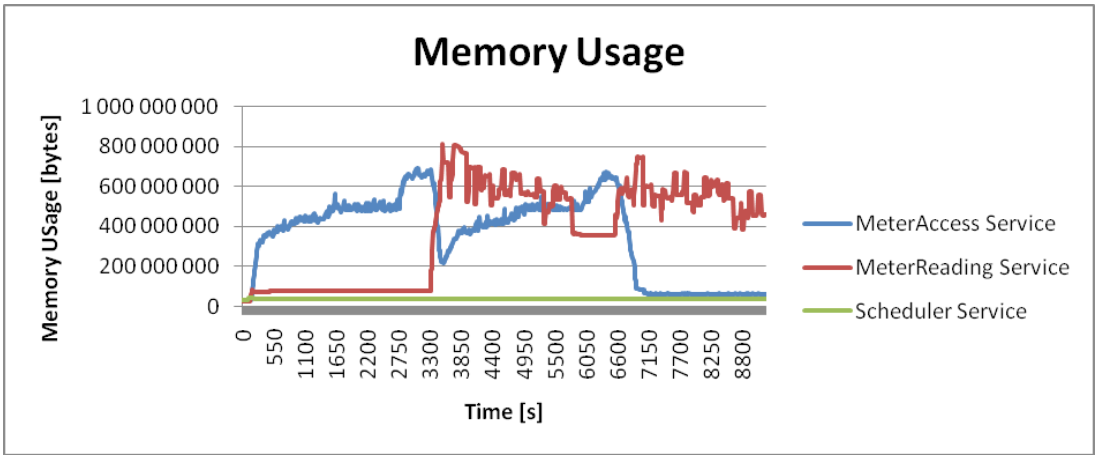
Z manjšimi konfiguracijami, torej 10x100 in 10x1000, SEP ni imel problemov. Zahteve po procesorju so bile nizke, saj je MeterAccess Service večino časa čakal, da od koncentratorjev dobi zahtevane podatke. Nekoliko več dela je zahteval le MeterReading Service, ki je skorajda vse rezultate dobil naenkrat. Poraba pomnilnika je sicer naraščala, sploh v konfiguraciji 10x1000, vendar do kakšnih posebnih ozkih grl ni prihajalo. V konfiguraciji 100x100 pa so stvari začele postajati zanimive. Procesor je imel delo ves čas, poraba pomnilnika pa se je nevarno približevala mejam, ki jih postavlja operacijski sistem. Testi konfiguracij 100x1000 in večjih pa se niso uspešno zaključili. V nekem trenutku je namreč operacijski sistem SEP-u zavrnil zahtevo po svežem kosu pomnilnika in branje je propadlo.

4.2 Konfiguracija 100x100

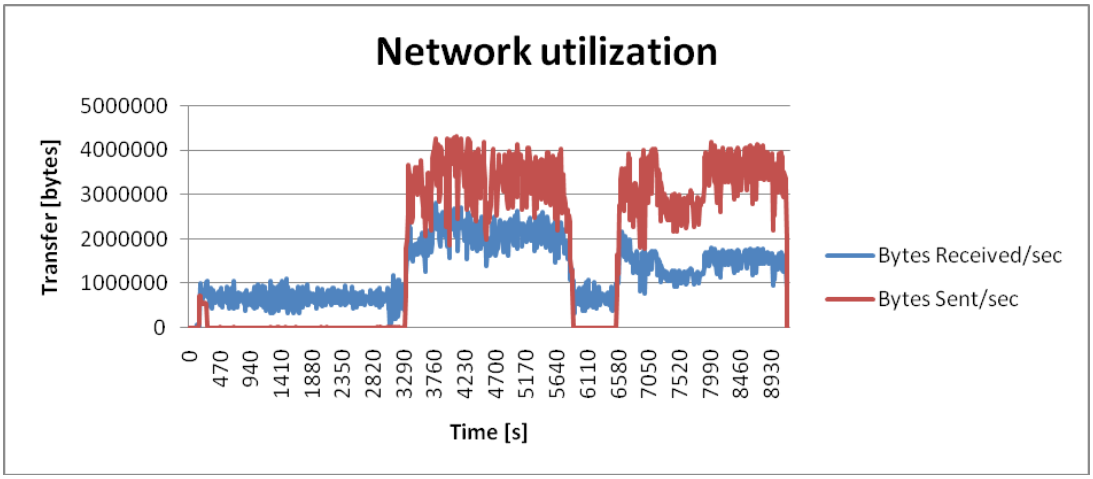
Kot že rečeno, iz testov pridobljenih podatkov je veliko. Na vpogled vam ponujam nekaj grafov, ki so iz teh podatkov nastali.



Slika 4.1 Poraba procesorja.



Slika 4.2 Poraba pomnilnika.



Slika 4.3 Zasedenost omrežja.

Pokomentirajmo najprej graf zasedenosti procesorja. Takoj na začetku je imel delo MeterReading Service, ko je iz baze bral podatke, ki so potrebni za izvršitev naloge, ter jih uporabljal za generiranje XML dokumentov, ki jih je potem poslal MeterAccess Service-u. Ko je MeterAccess Service XML dokumente prejel, se je spravil na delo. Vsak od koncentratorja prejet odgovor je bilo potrebno pregledati in ustvariti XML odgovor, ki se je potem posredoval nazaj MeterReading Seivice-u, ta pa je prejete rezultate shranil v bazo. Očitno je, da se je ta postopek izvedel dvakrat. Ta trenutek se v MeterAccess Service-u nahaja omejitev 50 hkratnih izvajanj zahtev. Ta omejitev je tam pač zato, ker neka omejitev mora obstajati. Z nastavljanjem le-te bi lahko dosegli optimalnejše delovanje sistema.

Naslednji graf nam prikazuje porabo pomnilnika. MeterAccsee Service takoj na začetku zahteva veliko pomnilnika, potem pa se prične pri svojih zahtevah brzdati. Ko je prvih 50 odgovorov poslanih, se lahko veliko pomnilnika sprosti in celoten proces se prične znova. MeterReading Service na drugi strani vse do prejema odgovora nima nobenih posebnih prostorskih zahtev. Ko pa odgovori pridejo, modul hitro zasede vse, kar se zasesti da.

Poleg tega lahko opazimo precejšnje nihanje v krivuljah na tem grafu, predvsem pri MeterReading Service-u. Ta nihanja se pojavljajo zaradi .NET-ovega upravljanja s pomnilnikom. V nekem trenutku, ko program zahteva dodaten pomnilnik, .NET ugotovi, da je prostega pomnilnika nevarno malo. Takrat se požene Garbage Collector, ki sprosti ves pomnilnik, katerega zasedajo objekti, ki jih program ne uporablja več.

Tretji graf nam prikazuje količino v omrežju pretočenih podatkov. V primerjavi s komunikacijsko infrastrukturo na terenu sem imel na voljo tako rekoč neskončno pasovno širino. Na ta način se je zelo točno pokazalo, kakšne so potrebe sistema po komunikacijskih kapacitetah. V začetni fazi je sistem vse podatke prejemal s terena. Ko je bilo treba te podatke shraniti, se je pojavila še komunikacija v drugo smer, poleg tega pa opazimo izrazit porast prometa proti komponentam sistema. Ta dodaten promet je posledica kontrole med SEP-om in bazo, v katero so se rezultati shranjevali.

Sedaj, ko vemo, kaj so naši rezultati, pa že lahko povemo, kje se težave pojavljajo, oziroma kje jih lahko pričakujemo.

4.3 Pomnilnik

Kakšne so zahteve modulov po pomnilniku? Že moduli, ki nič ne delajo, niso ravno majhen porabnik pomnilnika, saj v mirovanju povprečno zasedajo okrog 30 MB. Že res, da jih lahko operacijski sistem prepiše v izmenjevalno datoteko, vendar s tem izgubimo na odzivnosti sistema. Ko pa servisi delujejo in obdelujejo podatke, pa se situacija samo še poslabša.

Datoteka, ki vsebuje podatke za pretekli mesec za 100 števcev, je velika 120 MB. Tako datoteko je potrebno prebrati in iz nje izluščiti uporabne podatke. Koliko je dejansko teh podatkov, na žalost ni mogoče izvedeti. Del teh podatkov je namreč tudi časovna značka rezultata, objekt, ki predstavlja čas v .NET ogrodju, pa nima konstantne velikosti, oziroma te velikosti ni mogoče izvedeti. Kakorkoli že, datoteke, ki jih generira MeterAccess Service, ko bere podatke s poligona, so skupaj velike 12 GB. Če uporabimo kompresijo, se velikost teh datotek zmanjša na "pičlih" 350 MB. Vendar to velja za množico 10.000 števcev, kar pa je daleč od SEP-ovega cilja, ki je 1.000.000 števcev.

Kakšne so možne rešitve?

- Kot je že bilo nakazano, nam lahko malo pomaga kompresija podatkov. Vendar res samo pomaga, kajti z večanjem števila števcev na poligonu se večajo tudi datoteke. Če je na koncentrator priključenih 1000 števcev, je stisnjen odgovor MeterAccess service-a velik 50 MB. Lahko se torej zgodi, da MeterReading Service poklekne že pod enim samim odgovorom. No, na tem mestu velja poudariti, da je sam MeterReading Service sprogramiran precej potratno. Vse zahteve hrani pri sebi v pomnilniku, tako kompresirane kot nekompresirane. Osnovni ukrep bi bil, da

kompresirane zahteve zapiše na disk ter tako sprosti nekaj prepotrebnih MB pomnilnika.

- Drugi ukrep, ki vsaj malo diši po programerski lenobi, je postavitev več strežnikov. Na ta način posamezen strežnik prevzame manjše breme.
- Bolj kompliciran rešitev bi vključevala obdelovanje samo dela datoteke z rezultati. Na ta način bi lahko obdelovali podatke iz delno razširjene datoteke, ali celo začeli z obdelavo še preden bi nam MeterAccess Service posredoval vse podatke.
- Najbolj radikalen ukrep pa bi bila sprememba formata podatkov v datoteki. Ta je v tem trenutku zasnovana kot XML datoteka, v kateri pa veliko podatkov poberejo kontrolni znaki. Vendar to pomeni spremembo za stranke, ki so se v nakup sistema SEP že podale. To sicer ni problem, če uporabljajo SEP kot celoto. Če pa uporabljajo samo MeterAccess, in ostale komponente sistema zamenjajo z svojimi rešitvami, morajo te svoje rešitve prilagoditi. Tega pa nihče ne bi bil vesel. Zaradi tveganosti te poteze, bi se izplačalo to optimizacijo pogledati podrobneje.

```
<struct Name="SEP2Register">
    <string>1.0.1.8.0.255</string>
    <double>124600</double>
    <enum>0</enum>
    <bitString>00000000000000000000000000000000</bitString>
    <datetime>2008-10-11T22:15:00Z</datetime>
</struct>
<struct Name="SEP2Register">
    <string>1.0.1.8.0.255</string>
    <double>231300</double>
    <enum>0</enum>
    <bitString>00000000000000000000000000000000</bitString>
    <datetime>2008-10-11T22:30:00Z</datetime>
</struct>
```

Prikazana sta dva rezultata. Pa si argumente oglejmo po vrsti. Prvi argument (`string`) nam pove, točno kateri rezultat smo brali (10.0.1.8.0.255 po COSEM

standardu pomeni profil aktivne energije). Naslednji argument je izmerjena vrednost. `Enum` argument nam pove, kakšne enote je ta izmerjena vrednost. Prišli smo do argumenta, ki najbolj bode v oči. `BitString` hrani informacijo o stanju, v katerem je bila merilna naprava v času meritve. Precej prostora zasede tudi časovna značka rezultata v ISO8601 formatu.

Sedaj vemo, kaj gledamo in vso stvar lahko podrobneje analiziramo. Ničesar ne moremo spremeniti pri argumentih double, enum in datetime. Rezultata ne moremo in niti ne smemo spreminjati. Bolj učinkovitega načina za shranjevanje vrednosti, ki se pogosto pojavljajo, pač ni. Skopariti s prostorom ne gre niti pri časovni znački, saj hočemo ohraniti čim bolj točno informacijo o času meritve.

Veliko pa se lahko naredi pri ostalih dveh argumentih. Na začetek XML dokumenta bi lahko vnesli oznako registra, zraven nje pa oznako za ta register (recimo zaporedno številko), ki bi se uporabljala znotraj tega argumenta. Če beremo samo en register, bi tako namesto 13 znakov uporabili zgolj enega. Ogromno se da prihraniti tudi pri bitString argumentu. Ta serija ničel ni nič drugega kot po bitih zapisano 32 bitno celo pozitivno število. Namesto 32 ničel, bi brez vsake izgube informacije lahko zapisali le eno. Nadalje se da nekaj prihraniti tudi pri XML oznakah. Namesto celih besed bi lahko uporabili enoznačne oznake, recimo namesto string "s", namesto enum "e" in namesto datetime "dt".

Pa zapišimo iste rezultate v optimizirani verziji.

```
<s Name=≫SEP2Register≪>
    <str>0</str>
    <d>124600</d>
    <e>0</e>
    <i>0</i>
    <dt>2008-10-11T22:15:00Z</dt>
</s>
<s Name=≫SEP2Register≪>
    <str>0</str>
    <d>231300</d>
    <e>0</e>
    <i>0</i>
```

```
    <dt>2008-10-11T22:30:00Z</dt>
</s>
```

Koliko smo pridobili? Hitro štetje znakov pokaže, da je stara struktura velika 200 znakov, nova pa le 97 znakov. Ni slabo, kajne? Pri 300 MB veliki datoteki to pomeni prihranek zajetnih 150 MB.

4.4 Procesorski čas

Zasedenost procesorskih kapacitet v procesu branja podatkov se na prvi pogled morda ne zdi problematična. Konec koncev je računalnik ob tem izvajanju imel nekaj rezerve, saj je bil ob konicah 75 odstotno zaseden. Kratka opomba, testni računalnik je bil dvojedern, 200 odstotna zasedenost pa bi pomenila, da sta povsem zasedeni obe jedri procesorja.

Potem pa sem začel razmišljati, iz česa je delo sestavljeno. MeterAccess Service je poslal zahtevo za branje števca koncentratorju in nato na podatke v povprečju čakal 40 sekund. Če je to počel za 50 števcev hkrati, je ta proces ponavljal na nekako 0.8 sekunde. Zahteva po branju je dolga 1 kB, koncentratorjev odgovor pa 400 kB. Tega je potrebno prebrati in si ga shraniti v pomnilnik. Pa nekoliko analizirajmo to dogajanje.

Konstrukcija zahteve je opravilo, ki ga lahko brez slabe vesti zanemarimo. Osredotočimo se torej na prejemanje odgovora. Bi lahko prejemanje datoteke tako zelo obremenilo računalnik? Niti pod razno, vsakdo ve, da je pobiranje datotek z interneta opravilo, ki zahteva čas, ne pa procesorko moč. Analiza odgovora? Kaj pa je težkega pri zaporednem branju datoteke? Ali pa pisanju?

Da se odkrije vzroke za tolikšno procesorsko zahtevnost, nam ne preostane drugega, kot zakopati se v kodo. Z uporabo profiliranja, ki je vgrajeno v programski paket Visual Studio 2008, se je hitro pokazalo, v katerih funkcijah se program zadržuje največ časa.

- Izkazalo se je, da program veliko časa prebije v funkciji Regex.Split, ki razbija regularne izraze. Ta funkcija je del ogrodja .NET, torej optimizacija te funkcije ne pride v poštev. Najbolje bi torej bilo, da bi se uporabi te funkcije čimbolj izogibali.

  Da ne bomo govorili preveč splošno, si poglejmo izsek funkcije, ki `Regex.Split` največ uporablja. Najbolj problematične so sledeče vrstice.

```
if (Regex.Split(logicalName, "\\\\.").Length == 5)
    return ParseLogicalNameDots5(logicalName);
```

```
if (Regex.Split(logicalName, "\\\\.").Length == 6)
    return ParseLogicalNameDots6(logicalName);
```

Funkcija `Regex.Split` prvi argument razbije na polje podstringov glede na ločitveni niz, ki ga podamo kot drugi argument. Od tega polja pa vzamemo samo njegovo dolžino. Bistvo teh nekaj vrstic je torej ugotavljanje, na koliko podnizov bo niz razpadel, na podlagi rezultata pa pokličemo primerno funkcijo, ki mimogrede to razbijanje potem naredi še enkrat. Ukrepamo torej tako, da `Regex.Split` funkcijo pokličemo samo takrat, ko potrebujemo celoten rezultat te funkcije, torej samo znotraj funkcij `ParseLogicalNameDots5` in `ParseLogicalNameDots5`. Ko to naredimo, pa moramo nekako ugotoviti, katero funkcijo poklicati. V ta namen pokličemo funkcijo, ki v nizu prešteje število ločitvenih nizov in na podlagi tega pove, na koliko delov bo niz razpadel. Popravljena koda bi torej izgledala nekako takole:

```
int substirngsNumber = CountSubstrings(logicalName, "\\\\.");
if (substringNumber == 5)
    return ParseLogicalNameDots5(logicalName);
else if (substringnumber == 6)
    return ParseLogicalNameDots6(logicalName);
```

Kratek test je pokazal, da funkcija `CountSubstrings` število podnizov določi tudi do 300-krat hitreje kot funkcija `Regex.Split`, kjer je število podnizov pravzaprav stranski rezultat. Z reševanjem enega problema smo torej rešili dva.

- Konkretno v prej omenjeni funkciji `ParseLogicalNameDots6` najdemo primer takoimenovane *kratke zanke* [12]:

```
for (int i = 0; i < parsed.Length; i++)
    ret[i] = byte.Parse(parsed[i]);
```

Verjemite mi na besedo, `parsed.Length` je vedno 6. In kar naenkrat ni več potreb po for zanki, s čimer privarčujemo pri šestih primerjanjih in inkrementih.

4.5 Podatkovna baza

Upravljanje z velikimi količinami podatkov je od nekdaj problematično. Količina podatkov in hitrost dostopa sta glavni težavi tudi v SEP-u.

- Baza, ki hrani samo podatke o 1.000.000 števcih, je na terenu velika 5 GB. Koliko podatkov v bazi zasedejo podatki o naročnikih, geografskih lokacijah in rezultatih, ne ve nihče. Lahko pa nekoliko ugibamo. Na podlagi stolpcev v tabeli, ki opisujejo odjemalce, sem ocenil, da posamezen zapis zahteva 500 B. Pomnožimo to z 2.000.000 vnosi in ta tabela zasede 1 GB. Pravzaprav niti ne tako zelo veliko. Povsem druga pesem so rezultati. Vsak rezultat v bazi zasede 16,5 B. Če hranimo rezultate za 1.000.000 števcev, potem bo baza v enem letu na račun prebiranja števcev zrasla za 525 GB. Poleg rezultatov v bazi hranimo tudi dnevnike dogodkov, ki so se dogajali na napravah. Ti vnosi pa niso periodični, tako da za ta del baze ne moremo predvideti prostorske zahtevnosti. Glavni del podatkov vseeno predstavljajo rezultati meritev in zdi se povsem dovolj, če stranki rečemo: "Vsako leto kupite nov disk. Večji kot je, boljše je. Od viška glava ne boli."

- Kako pa je s samo hitrostjo baze? Posebnih testov v tej smeri ni bilo narejenih, sem pa podrobneje spremljal vnašanje naprav v bazo. Vendar lahko povem le, da je Microsoftova SqlExpress baza vsaj dvakrat počasnejša od IBM-ove DB2 baze. V tej smeri se da sklepati iz grafa o zasedenosti procesorja. Vidimo lahko, da zapisovanje 15 minutnega mesečnega profila za 5.000 števcev traja približno eno uro. V enem dnevu je torej teoretično mogoče v bazo zapisati rezultate za 120.000 števcev. Na prvi pogled se to zdi malo, vendar si to dejstvo zasluži razlago. V realnosti taka branja niso pogosta. Prvič je razlog v obsegu podatkov. Če hoče biti elektrodistribucijsko podjetje na tekočem s porabo, potem jim mesec ali celo teden stari podatki ne pomenijo ravno veliko. Branja se največkrat vršijo za pretekli teden, pri čemer se uporabi optimizacija branja, pri kateri se berejo le podatki, ki v bazi manjkajo. Če tako branje izvajamo vsak dan, je podatkov, ki jih je potrebno zapisati v bazo, tridesetkrat manj, zapisovanje pa je tridesetkrat hitrejše. Tako se podatki za 5.000 števcev zapišejo v dveh minutah in v enem dnevu je baza sposobna shraniti rezultate za 3.600.000 števcev. Kar pa je že spodobna številka.

Stvari se zapletejo, ko ne gre samo za shranjevanje podatkov v bazo. Sistem prav tako potrebuje podatke o napravah na terenu, strankam je potrebno izstaviti račune in tako naprej. Vsi ti podatki se nahajajo v bazi in če naenkrat prihaja do veliko branj in pisanj, se lahko pojavijo težave. Analiza tega dela sistema bi prav lahko bila samostojno diplomsko delo.

5 Zaključki

V tej diplomski nalogi sem poskušal analizirati zmogljivostne sistema SEP z uporabniškega stališča. Identificiral sem nekaj težav, na katere lahko naleti uporabnik sistema, navedel vzroke za te težave in predlagal rešitve. Torej je cilj tega pisanja dosežen.

SEP je dosegel svoj namen in je sposoben upravljati z 1.000.000 števci, za kar je bil ustvarjen. Vendar tega ne dela za nizko ceno. Sistem rešuje njegova modularna zasnova, s katero se lahko delo porazdeli med več strežniki. Operater omrežja mora zelo dobro poznati SEP-ove omejitve, da lahko sistem pravilno nastavi.

Kaj pa lahko razvijalci na področju zmogljivosti naredimo v prihodnosti?

Najlažje rešljiv je problem procesorske zahtevnosti. Potrebujemo le teden dni, da se koda prečisti.

Najbolj pereč pa je problem porabe pomnilnika, ki ni tako zelo enostavno rešljiv. MeterAccess Service in MeterReading Service bosta morala večino podatkov imeti shranjenih na disku, za razliko od sedaj, ko vse podatke hranita v pomnilniku. Potrebno se je odločiti za enega od dveh pristopov. Ali podatke razbiti na nivolju datoteke (tisto, kar je ta trenutek ena datoteka, razbiti na več datotek) ali pa na nivoju branja dato-

teke (obdelovati le del datoteke). Predvsem zaradi ohranjana združljivosti s starejšimi razvojnimi verzijami sistema je bolj privlačna druga pot, prva pa je lažja.

Baza je ostala v veliki meri netestirana, saj ni bila izpostavljena realnim razmeram. Poleg bremena, ki ga za bazo predstavlja vpisovanje rezultatov, je potrebno preveriti še, kako se baza obnaša, ko do nje dostopa več odjemalcev, kot so recimo odjemalci energije, ki bi radi preverjali lastno porabo energije. Če se izkaže, da baza ni kos vsem zahtevam, smo v velikih težavah.

Pa tudi če sistem teh problemov ne bi imel, dela ne bi tako hitro zmanjkalo. Stranke se hitro razvadijo in če hočejo danes imeti 15 minutni profil, bodo jutri hoteli imeti 5 minutnega, pojutrišnjem pa le minutnega.

SEZNAM SLIK

LITERATURA

[1] Wikipedia, Électricité de France, dostopno na
http://en.wikipedia.org/wiki/%C3%89lectricit%C3%A9_de_France.

[2] Wikipedia, Advanced Metering Management, dostopno na
http://en.wikipedia.org/wiki/Advanced_Metering_Management

[3] Wikipedia, Automatic Meter Reading, dostopno na
http://en.wikipedia.org/wiki/Automatic_meter_reading

[4] A Tropos Networks Case Study, Corpus Christi Pioneers Metro-Wide Wi-Fi Mesh Network for AMR, July 2007, dostopno na
www.tropos.com/pdf/case_studies/tropos_casestudy_corpus_christi.pdf

[5] Wikipedia, Public Switched Telephone Network, dostopno na
http://en.wikipedia.org/wiki/Telephone_network

[6] Wikipedia, Windows Management Instrumentation, dostopno na
http://en.wikipedia.org/wiki/Windows_Management_Instrumentation

[7] Wikipedia, Power Line Communication, dostopno na
http://en.wikipedia.org/wiki/Power_line_communication

[8] Wikipedia, IEC 62056, dostopno na
http://en.wikipedia.org/wiki/IEC_620565

[9] SWS_SEP2Scheduler.doc, interna specifikacija podjetja Iskraemeco d.d.

[10] MAS Classes Description.doc, interna specifikacija podjetra Iskraemeco d.d.

[11] DatabaseResultsStatus.xls, interna specifikacija podjetja Iskraemeco d.d.

[12] Nikolaj Zimic, Miha Mraz, *Temelji zmogljivosti računalniških sistemov*, Založba FE in FRI, Ljubljana 2006.

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja izr. prof. dr. Miha Mraza. Pomoč drugih sodelavcev sem v celoti navedel v zahvali.

— Anže Hrast, Ljubljana, oktober 2008.