FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Erik Kralj

Razvoj knjižnice za odjemalsko stran gonilnika protokola SML

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

izr. prof. dr. Miha Mraz

Ljubljana, 2010

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: 00029/2010
Datum: 04.10.2010

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

Kandidat: **ERIK KRALJ**

Naslov: **RAZVOJ KNJIŽNICE ZA ODJEMALSKO STRAN GONILNIKA PROTOKOLA SML**

DEVELOPMENT OF CLIENT'S PART OF SML PROTOCOL DRIVER

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Kandidat naj v svojem delu predstavi razvoj knjižnice za odjemalsko stran gonilnika protokola SML, ki se uporablja za komunikacijo med pametnimi merilnimi napravami. Pri tem naj predstavi osnove SML protokola, obrazloži izbiro razvojnih okolij in umesti svojo rešitev v sistemsko rešitev proizvajalca merilnih naprav IskraEmeco d.d. Kandidat naj izdelano knjižnico oceni tudi z vidika pomnilnih in odzivnostnih zmogljivostnih vidikov.

Mentor:

prof. dr. Miha Mraz

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a Erik Kralj,

z vpisno številko 63060169,

sem avtor/-ica diplomskega dela z naslovom:

Razvoj knjižnice za odjemalsko stran gonilnika protokola SML

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  izr. prof. dr. Miha Mraz

  in somentorstvom (naziv, ime in priimek)

  _____

- 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 _____ Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se doktorju Mihu Mrazu, izrednemu profesorju Fakultete za računalništvo in informatiko, za njegovo mentorstvo, pomoč ter smernice pri izdelavi diplomske naloge in predstavitve.

Zahvaljujem se tudi mentorju in sodelavcu Boštjanu Polancu in vsem ostalim sodelavcem podjetja Iskraemeco d.d., za njihovo pomoč, nasvete in motivacijo pri izdelavi aplikacije ter diplomske naloge.

Kazalo

Seznam uporabljenih kratic in simbolov

Kratica/Simbol	Pomen
COSEM	COSEM (*COmpanion Specification for Energy Metering*) so pravila zasnovana na obstoječih standardih za izmenjevanje podatkov med energijskimi števci.
CRC	CRC (*Cyclic redundancy check*) je nezaščitena zgoščevalna funkcija, ki se uporablja za preverjanje pravilnosti podatkov.
DLMS	DLMS (*Device Language Message specification*) je podobno kot protokol SML, eden od mnogih komunikacijskih protokolov, ki se uporablja za komunikacijo s pametnimi merilnimi napravami.
IDE	IDE (*Integrated development environment*) je aplikacija, ki olajša izdelovanje programske opreme, saj vsebuje urejevalnik kode, prevajalnik, orodja za avtomatizacijo prevajanja in razhroščevalnik.
MAS	MAS (*Meter Access Service*) je strežniška aplikacija, ki omogoča drugim aplikacijam komunikacijo z napravami preko različnih prenosnih medijev in komunikacijskih protokolov.
MSB	MSB (*Most Significant Bit*) je bit v bajtu z največjo utežjo. To je glede na format zapisa bajta lahko skrajno levi ali skrajno desni.
OBIS	OBIS (*OBject Identification System*) kode so standardne kode (naslovi) za vse podatke naprav, ki so v skladu standardov DLMS/COSEM.
SML	SML (*Smart Message Language*) je protokol za zajemanje podatkov in nastavljanje delovanja nizko energijskih naprav.
UTC	UTC (*Coordinated Universal Time*) je zapis časa brez dodatka časovne cone ali poletnega časa.

Povzetek

Iskraemeco d.d. se že 60 let ukvarja s proizvodnjo števcev, že vrsto let pa se ukvarjajo tudi z razvojem pametnih števcev, ter z aplikacijami za upravljanje z njimi. Za branje in nastavljanje teh števcev so razvili že mnogo rešitev. Zadnja inovacija v razvoju se imenuje *SEP2W System*. Sistem omogoča s pomočjo enega od mnogih vtičnikov (strežniške aplikacije MAS), brati in nastavljati pametne števce skozi vrsto komunikacijskih in transportnih protokolov.

Zadnjih nekaj let se v Nemčiji razvija nov protokol imenovan SML. Da bo strežniška aplikacija MAS lahko komunicirala s pametnimi napravami, ki uporabljajo ta protokol, mora Iskraemeco d.d. razviti gonilnik. V dokumentu bom opisal zahteve ter omejitve pri razvoju gonilnika, orodja, ki so razvoj omogočala, funkcionalnosti ter nastavitve gonilnika, predstavil aplikacijo, ter jo analiziral.

Ključne besede: SML, gonilnik, knjižnica, števec, komunikacija

Summary

Iskraemeco d.d. is engaged in production of meters for 60 years now and for many years also in production of smart meters and applications used for handling them. Company already developed many solutions for reading and configuration of meters. The last innovation in development is called *SEP2W System*. Through one of its many plug-ins (MAS service) system enables reading and configuration of smart meters over a range of different communication and transport protocols.

Over the last few years a new protocol named SML is being developed in Germany. In order for MAS service to support communication with smart devices that use this protocol, Iskraemeco d.d. must develop a new driver. In this document I will describe driver requirements, limitations I encountered in development, tools used, driver functions and settings. I will also demonstrate how this driver is used and perform an analysis of its workings.

Key words: SML, driver, library, meter, communication

1. Uvod

Z komercialno uporabo električne energije leta 1880 je nastopila potreba po natančnejšem merjenju njene porabe [1]. Do tedaj porabe sploh niso merili. Zaračunali so jo tako, da so prešteli število svetilk stranke in zaračunali fiksno ceno brez ozira na dejansko porabo. Nastala je potreba po napravi, ki bi delovala podobno, kot že obstoječi merilniki porabe plina. Ustvarjenih je bilo več prototipov. V Ameriki je Thomas Edison izdelal števec za odčitavanje enosmernega toka, ki je s pomočjo elektro-kemične reakcije odčital porabo električne energije. Števec je uporabljal elektrolitno celico v kateri sta se nahajali dve ploščici. Ko so želeli odčitati porabo, so te ploščice odstranili ter jih stehtali. Te števci niso bili preveč priljubljeni, saj je bilo odčitavanje porabe zelo zahtevno. Leta 1885 je Sebastian Ziani de Ferranti v Veliki Britaniji izdelal števec, ki je omogočal uporabnikom lažje odčitavanje energije s pomočjo prikazovalnika podobnega števcu plina. Prvi pravi števec električne energije je patentiral Dr. Hermann Aron leta 1883. Leta 1888 so jih v Veliki Britaniji začeli komercialno uporabljati. Že pred tem so se uporabljali števci električne energije, vendar so znali odčitati le porabo energije v danem trenutku. Aronov števec je znal odčitavati porabo, jo pomniti in nato prikazati celotno porabo za določen časovni interval. Slaba stran Aronovega števca je bila, da je znal odčitavati le enosmerni tok. Leta 1889 se je pojavil števec, ki ga je zasnoval Ottó Bláthy. Bláthy-števci so bili prvi števci, ki so merili izmenjujoči tok v vatnih urah. Okoli istega leta je tudi v Ameriki Elihu Thomson razvil števec, ki je bil zmožen meriti vatne ure ter jih sešteti za določen časovni interval. Števec je zaradi načina izdelave celo omogočal merjenje izmenjujočega in enosmernega toka. Leta 1894 je Oliver B. Shallenberg izdelal prvi indukcijski števec. Slednji se še danes na veliko uporablja. Čeprav indukcijski števci delujejo samo za merjenje izmenjujočega toka, to v današnjem času ni več problem, saj se tok izključno prenaša v izmenični obliki.

Za mehanskimi števci se počasi začenja doba pametnih števcev [2]. Kadar govorimo o pametnih števcih, večinoma mislimo kar na števce za merjenje električne energije. Vendar vedno več pametnih števcev omogoča merjenje porabe različnih virov, kot sta na primer voda ter plin. Pametni števec je števec, ki odčitava porabo energije vsaj vsako uro in vsaj enkrat na dan pošlje prebrane podatke napravi za nadzor in zaračunavanje. Pametni števci omogočajo obojestransko komunikacijo z nadzorno napravo. Za razliko od običajnih števcev omogočajo odčitavanje porabe na daljavo. Podobni števci, imenovani intervalni ali števci porabe v času, so obstajali že vrsto let, vendar z nekaj ključnimi razlikami. Pametni števci omogočajo odčitavanje v realnem času ali pa skoraj realnem času, omogočajo pošiljanje raznih opozoril ter nadzor kvalitete energije. Stari intervalni oziroma števci porabe v času tudi niso nikoli bili namenjeni splošni uporabi. Intervalni oziroma števci porabe so se vgrajevali le za komercialne in industrijske uporabnike. Proizvodnja pametnih števcev je cenejša in zato omogoča uporabo tudi v privatnih domovih. Tradicionalni števci ne omogočajo pregleda nad tem kdaj je bila energija porabljena. S pametnimi števci lahko proizvajalci vpeljejo več različnih tarif oziroma

cen električne energije, ki so določene glede na količino porabe električne energije v nekem času. Zaradi stalne komunikacije z nadzorno napravo lahko proizvajalec tudi nastavlja tarife kadar želi. Pametni števci pa so tudi veliko bolj ekološki, kot je videti na prvi pogled. Z različnimi tarifami ter s prihajajočimi funkcionalnostmi, ki omogočajo uporabniku, da pogleda trenutno porabo, bodo proizvajalci električne energije mogoče lahko pripravili uporabnike do tega, da zmanjšajo porabo električne energije oziroma, da večino energijsko potrošnih naprav priklapljajo v času manjše uporabe. S tem bi zmanjšali potrebo po izgradnji novih elektrarn in proizvodnji velike količine energije, ki bi slabo vplivala na okolje. Medtem ko sami pametni števci tega ne omogočajo, je mogoče z dobro nadzorno napravo s katero števci komunicirajo, vzpostaviti tudi pametno mrežo.

Pametna mreža je razvijajoča tehnologija, katere namen je optimizirati porabo ter proizvodnjo električne energije [3]. Pametna mreža je mreža pametnih števcev, ki komunicira z neko nadzorno napravo. Ta nadzorna naprava lahko preverja porabo energije na določenem območju. S tem lahko prepreči preobremenitve električnega omrežja z izklopom nekaterih uporabnikov, kar bi lahko v nasprotnem primeru pustilo trajne posledice nanjo. S stalnim nadzorom bo omogočala tudi druge napredne storitve. Že pri pametnih števcih je bilo omenjeno, da bi proizvajalci radi imeli bolj enakomerno porazdeljeno porabo energije skozi dan. V ta namen se že uporabljajo različne tarife, ki določajo ceno energije glede na čas dneva. Tarife pa še vedno zahtevajo, da potrošniki skrbimo za smotrno porabo elektrike. S pametno mrežo pa bo mogoče narediti sistem, ki bi sam, glede na trenutne zaloge proste energije, priklapljal oziroma izklapljal naprave po domovih, ki nimajo omejitev s časom izvajanja (npr. pralni stroj, pomivalni stroj, itd.).

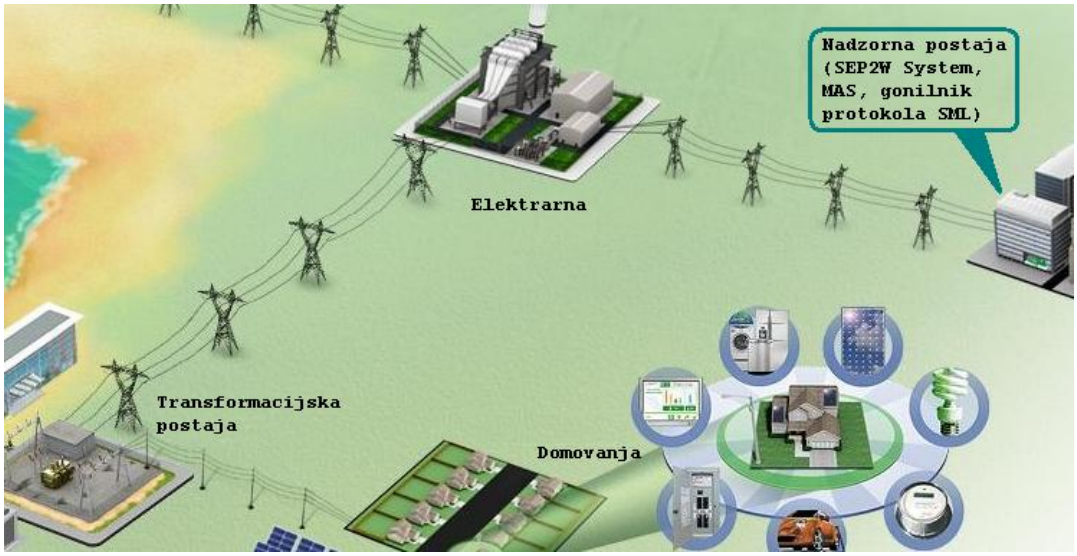
Za uresničitev ideje o pametni mreži pa je potrebno najprej rešiti probleme s pametnimi števci [2]. Sama izdelava števca se lahko od proizvajalca do proizvajalca razlikuje. Zato mora za pravilnost branja podatkov ter pravilnost shranjevanja skrbeti proizvajalec sam. Problemi pa nastopajo pri komunikaciji. Števec mora biti zmožen zanesljivo ter varno posredovati prebrane informacije neki centralni enoti. Prvi problem komunikacije leži v izbiri skupnega prenosnega medija. Predlagani mediji so telefonsko omrežje, omrežje osebnega klicalnika (`Pager`), satelit, radijsko omrežje in komunikacija preko same električne napeljave. Vsi prenosni mediji imajo prednosti ter slabosti. Na primer v slabo poseljenih in goratih območjih imajo brezžične tehnologije večje probleme, v gosto poseljenih urbanih območjih pa je veliko lažje zagotoviti brezžično povezavo, kot pa napeljati nove fiksne prenosne medije. Najbolj privlačna vrsta komunikacijskega protokola je protokol TCP/IP, a nekateri bi raje videli, da bi se standardiziral univerzalni priključek, ki bi ločil napravo od komunikacijskega modula (s tem tudi prenosnega medija). S tem bi bilo mogoče začeti z masovno proizvodnjo pametnih naprav preden se določi standardni način transportne poti. Tudi opisani protokol SML vsebuje definicijo svojega transportnega protokola.

Pravi namen protokola SML pa je reševanje drugega problema komunikacije in sicer razumevanja med števcem ter nadzorno enoto. Protokol SML definira pravila oziroma jezik komunikacije med strežnikom ter odjemalcem. Prilagojen je za komunikacijo z merilniki energije. Sam protokol je še vedno v razvoju in zadnja prosto dostopna verzija protokola SML 1.03 je izšla konec leta 2008. Protokol, ki je bil razvit v Nemčiji, naj bi bil en in edini protokol za komunikacijo s pametnimi števci po Nemčiji. Iz tega razloga mora podjetje Iskraemeco d.d., podjetje, ki se že več kot 60 let ukvarja z razvojem števcev ter zadnjih nekaj let z aplikacijo za upravljanje s pametnimi števci, podpreti tudi komunikacijski protokol SML. Z izdelavo števca ter podporo protokola se lahko Iskraemeco še naprej obdrži na nemškem tržišču.

Dokument se nanaša na odjemalski gonilnik protokola SML, ki je sposoben komunicirati s strežniškim protokolom SML, ki ga uporabljajo števci. V drugem poglavju je prikazano mesto oziroma vlogo gonilnika protokola SML. Na kratko so opisani odjemalci ter strežniki, ki nastopajo pri komunikaciji. Opisane so tudi oblike oziroma formati podatkov s katerimi bo gonilnik protokola SML upravljal pri opravljanju svojih nalog. V tretjem poglavju je opisana aplikacija ter orodja, ki so se uporabljala v razvoju gonilnika protokola SML. S tem imam v mislih tako aplikacije, ki so se uporabljale za sam razvoj, kot tudi odjemalce ter strežnike. Sledila bo specifikacija knjižnice, ki bo vsebovala razredni diagram ter nastavitve. V četrtem poglavju je predstavljen končni rezultat razvoja gonilnika protokola SML, prikazan primer uporabe ter analizira delovanja knjižnice.

2. Namen knjižnice

Namen gonilnika protokola SML je omogočati komunikacijo med strežnikom in odjemalcem preko protokola SML. Strežnik s katerim bo knjižnica komunicirala bo najverjetneje števec stranke, lahko pa bo tudi kakšna druga pametna naprava, ki bo uporabljala protokol SML (npr. koncentrator). Odjemalec je aplikacija, ki želi brati oziroma nastavljati števce stranke. Ta aplikacija bi bila lahko zelo preprosta, kot na primer prosto dostopna spletna aplikacija *Google PowerMeter*, ki omogočala splošnim uporabnikom brati svoje števce določenih dobaviteljev električne energije. Lahko pa bi bila bolj kompleksna aplikacija v rokah dobaviteljev, ki omogoča večji nadzor nad pretokom energije, kot na primer *SEP2W System*. Takšna aplikacija bi omogočala z avtomatiziranjem branj ter odklopov, že neko preprosto obliko pametne mreže. Na sliki 1 lahko vidimo najbolj verjetno lokacijo gonilnika protokola SML, in sicer v nadzorni postaji. Nadzorna aplikacija ki bo uporabljala knjižnico, lahko teče na neki oddaljeni nadzorni postaji in kljub temu nadzoruje pretok energije po električnem omrežju.



Slika 1: Gonilnik protokola SML uporablja aplikacija, ki nadzoruje pretok električne energijo s pomočjo branja ter nastavljanja števcev uporabnikov [4].

2.1. Podrobnejša umestitev knjižnice

Gonilnik protokola SML bi bil mogoče bolje poimenovan kot razčlenjevalnik protokola SML. Knjižnica je vmesnik, ki odjemalcu omogoča prevajanje zahtev v binarne datoteke SML, ki jih strežnik lahko razume in nanje smiselno odgovarja. Gonilnik protokola SML nato vrne podatke zapisane znotraj prejetih odgovorov teh zahtev. Glede na njegov položaj v toku komunikacije, mora gonilnik protokola SML znati le sestavljati zahteve protokola SML, ter

brati odgovore protokola SML. Gonilniku protokola SML ni potrebno sestavljati odgovorov na zahteve, saj nikoli ne nastopa v vlogi strežnika. Knjižnica je vseeno zgrajena tako, da bi bila sprememba njene vloge prav tako mogoča brez večjih problemov, če bi bilo le to potrebno. Na sliki 2 vidimo bolj posplošen prikaz vloge oziroma lokacije knjižnice. Odjemalec je vedno strežniška aplikacija MAS. Strežnik je lahko katerakoli aplikacija oziroma naprava, ki je sposobna odgovarjati na zahteve protokola SML s smiselnimi odgovori.

Slika 2: Posplošen prikaz vloge knjižnice gonilnika protokola SML.

2.2. Odjemalec

Odjemalec gonilnika protokola SML je vedno strežniška aplikacija MAS. Le ta omogoča drugim aplikacijam dostop do različnih naprav preko različnih prenosnih medijev in protokolov. Z gonilnikom protokola SML bo strežniška aplikacija MAS pridobila nov gonilnik protokola, ki naj bi postal standard za komunikacijo s števci po vsej Nemčiji. Glavni uporabnik strežniške aplikacije MAS je `SEP2W System`, uporablja pa jo tudi aplikacija `MAS Client`.

`SEP2W System` vsebuje široko področje funkcionalnosti. Omogoča branje števcev, nastavljanje števcev, nadgradnjo števcev, združevanje podatkov, overjanje pravilnosti podatkov, poročanje, avtomatično izvajanje nalog in še mnogo več. Strežniška aplikacija MAS je le ena od njenih komponent. Zahteve se v tej aplikaciji generirajo s pomočjo uporabniškega vmesnika.

Aplikacija `MAS Client` je preprostejša aplikacija ustvarjena le z namenom upravljanja s strežniško aplikacijo MAS. Aplikacija prepušča definicijo ukazov namenjenih strežniški aplikaciji MAS kar uporabniku. Zahteve se definirajo v obliki dokumenta XML. Tudi odgovori so v obliki dokumenta XML.

Da strežniška aplikacija MAS lahko uporablja nek gonilnik protokola mora ta gonilnik razširiti razred `BaseDriver`. Gonilnik protokola s tem dobi nekaj abstraktnih metod, ki jih strežniška aplikacija potrebuje za komunikacijo in metode, ki mu omogočajo komunikacijo preko različnih prenosnih medijev.

Komunikacijo preko prenosnih medijev zagotavljajo povezovalni gonilniki in ne gonilniki protokolov. Na sliki 3 lahko vidimo razširjen prikaz komunikacije odjemalca s strežnikom.

Slika je simbolična, saj gonilnik protokola ne komunicira direktno s povezovalnim gonilnikom. Komunicira le s strežniško aplikacijo MAS.

Slika 3: Razširjen (simbolični) potek komunikacije med odjemalcem ter strežnikom preko gonilnika protokola SML.

Komunikacija z odjemalcem

Za komunikacijo gonilnika protokola SML s strežnikom poskrbi pet metod, ki jih mora vsak gonilnik protokola podpreti ob razširitvi razreda `BaseDriver`. Parametri metod so razširjeni razredi iz razreda `MasData`. Gonilnik protokola SML mora zato omogočati pretvorbo svojih objektov v/iz razreda `MasData`. Metode, ki jih mora gonilnik protokola vsebovati, so sledeče:

- `OptimizeOperationsImpl` – Gre za prvo metodo, ki jo strežniška aplikacija MAS pokliče, ko želi komunicirati s strežnikom. Metoda kot parameter prejme zahteve, ki jih odjemalec želi posredovati strežniku. Namen metode je, da glede na nastavitve gonilnika protokola sestavi zahteve v obliko, ki bi bila že primerna za komunikacijo s strežnikom. Gonilnik protokola SML v tej metodi prebere nastavitve, sestavi zahteve v format protokola SML in jih združi v večje skupine imenovane datoteka SML.

- `BeforeConnect` – Metoda vsebuje kodo, ki se mora izvesti pred povezavo gonilnika protokola s strežnikom. Metoda lahko preko vhodnega parametra tudi prejme nastavitve, če so le te potrebne. Metoda se v gonilniku protokola SML ne uporablja, saj protokol SML nima nikakršnih vzpostavitev ali rušenj sej.

- `ConnectImpl` – Naloga metode je vzpostaviti povezavo s strežnikom. Metoda lahko preko vhodnega parametra tudi prejme nastavitve, če so le te potrebne. Metoda v gonilniku protokola SML le poizkusi prebrati javni ključ, če le ta ni bil podan v nastavitvah.

- `ExecuteInvokeImpl` – Namen te metode je izvajanje ene same zahteve. Ker se zahteve sestavijo v datoteke SML že v metodi `OptimizeOperationsImpl`, se v tej metodi samo pošlje datoteka, v kateri se nahaja ta zahteva. S pomočjo vhodnega parametra metoda poišče datoteko, ki vsebuje trenutno zahtevo. Večinoma metoda datoteko pošlje, vendar ker datoteke SML lahko vsebujejo več kot eno zahtevo je mogoče, da je bila ta datoteka že poslana strežniku. Ob prvem pošiljanju datoteke se vanjo zapišejo vsi prispeli odgovori. Tudi če datoteke ni bilo potrebno poslati strežniku, datoteka sedaj vsebuje odgovore na zahteve. Metoda poišče želene rezultate ter jih vrne.

- `DisconnectImpl` – Metoda `DisconnectImpl` je zadnja metoda, ki jo strežniška aplikacija MAS pokliče. Metoda mora zaključiti komunikacijo odjemalca s strežnikom. Metoda se v gonilniku protokola SML ne uporablja, saj protokol SML nima vzpostavitev ali rušenj sej.

2.3. Strežnik

Strežnik je lahko katerakoli strežniška enota do katere strežniška aplikacija MAS omogoča povezavo preko povezovalnega gonilnika, ki omogoča strežbo preko protokola SML. V začetku izdelovanja gonilnika protokola na primer je bila strežniška enota kar preprosta aplikacija. Aplikacija je simulirala napravo, ki uporablja za komunikacijo protokol SML.

Komunikacija s strežnikom

Za komunikacijo s strežnikom mora gonilnik protokola posredovati pravilen format podatkov po določenem prenosnem mediju. Prenos podatkov po prenosnem mediju pa je delo povezovalnega gonilnika. Razširitev razreda `BaseDriver` omogoča gonilniku protokola uporabo preprostih metod za komunikacijo preko prenosnega medija:

- `Send` – Metoda pošlje tabelo bajtov preko prenosnega medija.

- `Receive` – Metoda sprejme parameter, ki predstavlja število bajtov, ki jih bo metoda prebrala in vrnila v tabeli.

2.4. Protokol SML

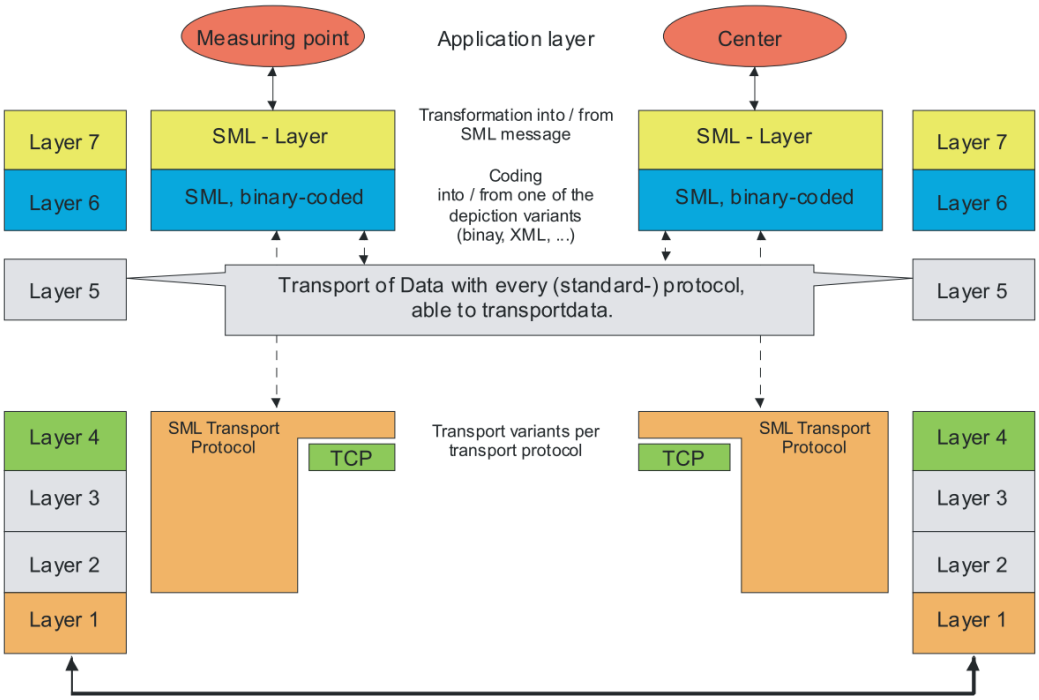
Zasnova protokola SML sega v leto 2007 [5, 6]. Ustvarjen je bil z namenom uporabe v nizko energijskih vdelanih sistemih. Protokol naj bi se uporabljal za branje podatkov na dolge razdalje. Zato mora biti protokol SML čim bolj preprost. Da naprava, ki uporablja ta protokol ne potrebuje velike količine spomina, so elementi sporočil postavljeni v zaporedju, s katerim napravi ni treba pomniti vseh predhodnih podatkov in je mogoče obdelovati zahtevo kar med

prejemanjem podatkov. Dober primer tega sta sporočili *SML Open Request* in *SML Close Request*, ki bosta podrobneje opisani kasneje. Sporočilo *SML Open Request* vsebuje element, ki napravi pove s kakšno kodno stranjo naj »spremeni« bajte v črke, oziroma s katero črko naj interpretira nek bajt. Sporočilo *SML Close Request* prekine vse nastavitve, ki so začele veljati z zadnjim sporočilom *SML Open Request*. Naprava bi lahko v trenutku sprejema shranila kodno stran v nek svoj register, ter zavrgla vse podatke pred tem, saj jih ne bo več potrebovala.

Protokol SML lahko delimo na dva dela:

- datotečna struktura SML – struktura podatkov zgrajena z določenimi pravili, ki je lahko predstavljena kot XML dokument ali binarna datoteka,
- transportni protokol SML – preprost transportni protokol, ki potrebuje serijsko povezavo od točke do točke in omogoča razbijanje in ponovno sestavljanje blokov podatkov ter preverjanje pravilnosti prenosa.

Kot je vidno na sliki 4, protokol ne zahteva, da se za komunikacijo preko protokola SML uporablja le transportni protokol SML. Datoteka SML v binarni ali XML obliki je lahko zapisana na trdem disku, ali pa prenesena preko katerega koli transportnega protokola.



Slika 4: Protokol SML [5, 6].

2.4.1. Datotečna struktura SML

Osnovni objekt komunikacije preko protokola SML je datoteka SML. Datoteka SML je sestavljena iz več sporočil SML. Datoteke SML so lahko zahtevne datoteke, datoteke z odgovori ali pa mešane datoteke. Vsaka zahtevna datoteka SML vsebuje na začetku natančno eno sporočilo *SML Open Request* in na koncu natančno eno sporočilo *SML Close Request*. Vsaka datoteka SML z odgovori vsebuje na začetku natančno eno sporočilo *SML Open Response* ali *SML Attention Response* in na koncu natančno eno sporočilo *SML Close Response* ali *SML Attention Response*. Med začetnimi in končnimi sporočili pa je lahko poljubno število drugih sporočil. Datoteka SML ima v sebi lahko tudi kombinacijo zahtev in odgovorov. Večinoma pa datoteke SML vsebujejo le eno vrsto sporočil. Gonilnik protokola SML deluje izključno z binarno obliko datoteke, saj je namen knjižnice prenos in ne prikazovanje oz. hranjenje podatkov.

2.4.2. Osnovni tipi SML

Protokol SML v osnovi uporablja pet tipov:

- *Octet String* – zaporedje bajtov spremenljive dolžine,
- *Boolean* – Boolov operator,
- *Integer* – celo število v 8, 16, 32, ali 64 bitni obliki,
- *Unsigned* – nepredznačeno celo število v 8, 16, 32, ali 64 bitni obliki,
- *ListOf* – seznam osnovnih tipov (vsak element je lahko drugačnega tipa).

Osnovni tipi so sestavljeni iz glave in telesa. Glava tipa SML vsebuje informacijo o vrsti tipa in dolžini telesa. Glava tipa *ListOf*, za razliko od ostalih osnovnih tipov, namesto dolžine telesa v bajtih vsebuje število osnovnih tipov, ki se nahajajo v telesu seznama.

Tip	b8	b7	b6	b5	b4	b3	b2	b1
Octet String	Zadnji?	0	0	0	Dolžina			
Boolean		1	0	0				
Integer8/16/32/64		1	0	1				
Unsigned8/16/32/64		1	1	0				

ListOf		1	1	1

Tabela 1: Sestava prvega bajta glave tipov binarnega zapisa SML tipa.

Prvi bit v glavi pove, če je to tudi zadnji bajt glave. Če je vrednost bita b8 enaka 1, to pomeni, da trenutnemu bajtu sledi še en. S tem je mogoče tipu določiti poljubno dolžino. Za vse tipe razen tip *ListOf* je dejanska dolžina za ena manjša od podane.

Šestnajstiški zapis	Dvojiški zapis	Dolžina
0x83 0x04	1000 0011 0000 0100	Ponazarja zaporedje dolgo 50 bajtov. Ker tip ni *ListOf*, se vsakemu bajtu dolžine odšteje ena: ((3 * 16) – 1) + (4 - 1).
0x76	0111 0110	Ponazarja seznam 6 elementov.

Tabela 2: Primer glave zaporedja bajtov, ki presega 15 bajtov in seznama 6 elementov binarnega zapisa SML tipa.

Ker je dolžina vseh tipov razen tipa *ListOf* za vsak bajt glave za eno manjša, obstaja posebna vrednost 0x01 (šestnajstiško). Ta vrednost predstavlja zaporedje znakov dolžine nič. V protokolu SML se ta vrednost uporablja za opcijske elemente, ki niso prisotni. Binarni zapis v protokolu SML poteka po velikosti od večje vrednosti proti manjši (*Little Endian*).

2.4.3. Razširjeni tipi SML

Večina razširjenih tipov, je razširjenih iz tipa *ListOf*. Zato so izpeljani tipi samo določeno zaporedje osnovnih tipov. Izjeme so večinoma samo drugače poimenovani osnovni tipi. Obstajajo še implicitno podani tipi, ki so v bistvu osnovni tip le, da je vrsta tipa podana ob komunikaciji sami in ne v specifikaciji protokola (v sporočilo lahko namestimo katerikoli osnovni tip). Na kratko bom obrazložil razširjene tipe, ki se v trenutni verziji gonilnika protokola SML uporabljajo.

1.1.1.1. SML Message

SML Message je osnovni izpeljani tip. Vsa komunikacija protokola SML teče preko tega tipa. Tip je generična lupina za vsa sporočila, ki se prenašajo med odjemalcem in strežnikom. Ko govorimo o več vrstah SML sporočil govorimo le o vrednosti telesa sporočila, ki pa je sestavni del tega tipa.

V nadaljevanju sledi psevdokoda strukture tipa *SML Message* in tabela 3, ki vsebuje podrobnejši opis posameznih elementov.

SML Message – *ListOf*(6)
{
Identifikacija transakcije – *Octet String*
Številka skupine – *Unsigned8*
Prekinitev ob napaki – *Boolean*
Telo sporočila – *Message Body*
CRC16 – *Unsigned16*
Konec sporočila SML – Konec sporočila SML
}

Polje	Pomen
Identifikacija transakcije	Polje, ki vsebuje identifikacijsko številko, ki jo poljubno generira odjemalec. Slednji uporabi to polje, da lahko poveže zahteve z odgovori.
Številka skupine	Polje, ki omogoča odjemalcu določiti katere zahteve se morajo izvesti skupaj, oz. v kakšnem vrstnem redu.
Prekinitev ob napaki	S tem poljem odjemalec strežniku naroči kakšne korake naj uporabi v primeru napake. Strežnik lahko nadaljuje izvajanje, nadaljuje izvajanje od naslednje skupine, izvede samo še naslednjo skupino, ali pa takoj prekine izvajanje.
Telo sporočila	Polje je razširjeni tip, v katerem se nahaja dejansko sporočilo.
CRC16	Polje vsebuje 16 bitni CRC, ki je izračunan po standardu DIN EN 62056-46. Za računanje CRC se uporablja celotno sporočilo SML brez zadnjih štirih bajtov (CRC16 in konec sporočila SML).
Konec sporočila SML	Konec sporočila SML je vedno predstavljeno kot bajt z vrednostjo 0x00 (šestnajstiško).

Tabela 3: Opis elementov tipa *SML Message*.

2.4.3.1. Message Body

Message Body je razširjeni tip, ki določa dejansko vrsto sporočila SML. Sam tip je seznam dveh elementov. Prvi element določa vrsto drugega elementa, ki pa je eno od različnih sporočil. *Message Body* je tudi izjema, ko pride do izbirnih tipov (seznami dveh elementov), saj je edini izbirni tip, pri katerem prvi element ni tipa Unsigned8.

V nadaljevanju sledi psevdokoda strukture tipa *SML Message* in tabela 4, ki vsebuje možne vrednosti elementa »Vrsta vsebine«.

Message Body – *ListOf*(2)
{
    Vrsta vsebine – *Unsigned16*/*Unsigned32*
    Vsebina – Vsebina (razširjeni tip glede na vrsto vsebine)
}

Vrsta vsebine (šestnajstiško)	Vsebina
0x00000100	*Open Request*
0x00000101	*Open Response*
0x00000200	*Close Request*
0x00000201	*Close Response*
0x00000300	*Get Profile Pack Request*
0x00000301	*Get Profile Pack Response*
0x00000400	*Get Profile List Request*
0x00000401	*Get Profile List Response*
0x00000500	*Get Proc Parameter Request*
0x00000501	*Get Proc Parameter Response*
0x00000600	*Set Proc Parameter Request*
0x00000601	*Set Proc Parameter Response*
0x00000700	*Get List Request*

0x00000701	*Get List Response*
0x0000FF01	*Attention Response*

Tabela 4: Seznam vseh možnih vrst vsebine (teles) v verziji protokola SML 1.4.

2.4.3.2. Open Request

Open Request se mora pojaviti na začetku vsake datoteke SML, ki vsebuje zahteve. Čeprav protokol SML nima sej, se to sporočilo obnaša kot začetek seje. Sporočilo poda določene podatke, ki veljajo za vse zahteve, ki sledijo temu sporočilu do prvega sporočila *Close Request*.

V nadaljevanju sledi psevdokoda strukture tipa *Open Request* in tabela 5, ki vsebuje podrobnejši opis posameznih elementov.

Open Request – *ListOf*(7)
{
Kodna stran – *Octet String* (opcijsko)
Identifikacija odjemalca – *Octet String*
Identifikacija zahtevane datoteke – *Octet String*
Identifikacija strežnika – *Octet String* (opcijsko)
Uporabniško ime – *Octet String* (opcijsko)
Geslo – *Octet String* (opcijsko)
Verzija SML – *Unsigned8* (opcijsko)
}

Polje	Pomen
Kodna stran	Polje, ki pove s kakšno kodno stranjo naj bodo predstavljeni nizi znakov. Privzeta kodna stran je specificirana v standardu ISO 8859-15.
Identifikacija odjemalca	Polje vsebuje identifikacijsko številko odjemalca, ki se uporablja za povezovanje zahtev z odgovori.
Identifikacija zahtevane datoteke	Polje, ki vsebuje še ena identifikacijska številka za povezovanje zahtev z odgovori.
Identifikacija strežnika	Polje, ki vsebuje identifikacijska številka strežnika. Če to polje ni prisotno, potem je sporočilo namenjeno

	kot razpršeno oddajanje.
Uporabniško ime	Polje, ki vsebuje uporabniško ime za overitev uporabnika.
Geslo	Polje, ki vsebuje geslo za overitev uporabnika.
Verzija SML	Polje vsebuje verzijo protokola SML, ki je trenutno podprta. Če polje ni prisotno, potem je privzeta verzija 1.

Tabela 5: Opis elementov tipa *`Open Request`*.

2.4.3.3. Open Response

`Open Response` se mora pojaviti na začetku vsake datoteke SML, ki vsebuje odgovore. Čeprav protokol SML nima sej, se to sporočilo obnaša kot začetek seje. Sporočilo poda določene podatke, ki veljajo za vse odgovore, ki sledijo temu sporočilu do prvega *`Close Response`*.

V nadaljevanju sledi psevdokoda strukture tipa *`Open Response`* in tabela 6, ki vsebuje podrobnejši opis posameznih elementov.

`Open Response` – *`ListOf`*(6)
{
Kodna stran – *`Octet String`* (opcijsko)
Identifikacija odjemalca – *`Octet String`* (opcijsko)
Identifikacija zahtevane datoteke – *`Octet String`*
Identifikacija strežnika – *`Octet String`*
Referenčni čas – *`SML Time`* (opcijsko)
Verzija SML – *`Unsigned8`* (opcijsko)
}

Polje	Pomen
Kodna stran	Polje pove s kakšno kodno stranjo naj bodo predstavljeni nizi znakov. Privzeta kodna stran je specificirana v standardu ISO 8859-15.
Identifikacija odjemalca	Polje, ki vsebuje identifikacijsko številko odjemalca.

	Če to polje ni prisotno, potem je sporočilo namenjeno kot razpršeno oddajanje.
Identifikacija zahtevane datoteke	Polje, ki vsebuje še ena identifikacijska številka za povezovanje zahtev z odgovori.
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika.
Referenčni čas	Polje, ki vsebuje čas kreacije odgovora. To polje je prazno, če strežnik nima prisotne enote s časom.
Verzija SML	Polje vsebuje verzijo protokola SML, ki je trenutno podprta. Če polje ni prisotno, potem je privzeta verzija 1.

Tabela 6: Opis elementov tipa *Open Response*.

2.4.3.4. Close Request

Close Request se mora pojaviti na koncu vsake datoteke SML, ki vsebuje zahteve. Čeprav protokol SML nima sej, se to sporočilo obnaša kot konec seje. Vsa pravila, ki so začela veljati s prvim *Open Request*, so s tem sporočilom nehala veljati.

V nadaljevanju sledi psevdokoda strukture tipa *Close Request* in tabela 7, ki vsebuje podrobnejši opis posameznih elementov.

Close Request – ListOf(1)
{
    Globalni podpis – SML Signature (opcijsko)
}

Polje	Pomen
Globalni podpis	Polje, ki vsebuje podpis za overjanje sogovornika. Ta tip je identičen tipu *Octet String*.

Tabela 7: Opis elementov tipa *Close Request*.

2.4.3.5. Close Response

Close Response se mora pojaviti na koncu vsake odgovorne datoteke SML. Čeprav protokol SML nima sej, se to sporočilo obnaša kot konec seje. Vsa pravila, ki so začela veljati s prvim *Open Response* so s tem sporočilom nehala veljati.

V nadaljevanju sledi psevdokoda strukture tipa *Close Response* in tabela 8, ki vsebuje podrobnejši opis posameznih elementov.

Close Response – ListOf(1)
{
    Globalni podpis – SML Signature (opcijsko)
}

Polje	Pomen
Globalni podpis	Polje, ki vsebuje podpis za overjanje sogovornika. Ta tip je identičen tipu *Octet String*.

Tabela 8: Opis elementov tipa *Close Response*.

2.4.3.6. Get Profile List Request

Get Profile List Request je edina zahteva, ki lahko prejme več odgovorov. Namen zahteve je branje preprostih seznamov podatkov. Zaradi preprostosti so odgovori malo daljši. V gonilniku protokola SML se ta zahteva uporablja za branje dogodkov števca.

V nadaljevanju sledi psevdokoda strukture tipa *Get Profile List Request* in tabela 9, ki vsebuje podrobnejši opis posameznih elementov.

Get Profile List Request – ListOf(9)
{
    Identifikacija strežnika – *Octet String* (opcijsko)
    Uporabniško ime – *Octet String* (opcijsko)
    Geslo – *Octet String* (opcijsko)
    S surovimi podatki – *Boolean* (opcijsko)
    Začetni čas – *SML Time* (opcijsko)
    Končni čas – *SML Time* (opcijsko)
    Drevesna pot parametrov – *SML Tree Path*
    Seznam objektov – *List of SML ObjReqEntry* (opcijsko)

Podrobnosti – *SML Tree* (opcijsko)
}

Polje	Pomen
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika. Če to polje ni prisotno, potem je sporočilo namenjeno kot razpršeno oddajanje.
Uporabniško ime	Polje, ki vsebuje uporabniško ime za overjanje sogovornika.
Geslo	Polje, ki vsebuje geslo za overjanje sogovornika.
S surovimi podatki	V protokolu SML to polje nima opisa. Gonilnik protokola SML tega polja ne uporablja.
Začetni čas	Polje, ki vsebuje začetni (najstarejši) čas podatkov, ki jih zahtevamo.
Končni čas	Polje, ki vsebuje končni (najnovejši) čas podatkov, ki jih zahtevamo.
Drevesna pot parametrov	Polje je razširjeni tip, ki je v bistvu samo seznam vrednosti *Octet String*. V seznam je treba namestiti pot do podatkov, ki jih želimo prebrati.
Seznam objektov	Polje je razširjeni tip, ki je v bistvu samo seznam vrednosti *Octet String*. Vrednost tega polja se uporablja kot filter za vrsto objektov, ki jih zahteva vrne. Gonilnik protokola SML tega polja ne uporablja.
Podrobnosti	Polje je razširjeni tip, ki je zelo prilagodljiv. V to polje bi lahko gonilnik protokola SML vstavil poljubno podatkovno strukturo, ki bi jo strežnik želel.

Tabela 9: Opis elementov tipa *Get Profile List Request*.

2.4.3.7. Get Profile List Response

Get Profile List Response je edini odgovor, ki se lahko pojavi večkrat za eno samo zahtevo. Namen odgovora je vračanje preprostih seznamov podatkov. A zaradi preprostosti so odgovori malo večji. Gonilnik protokola SML iz tega odgovora izlušči dogodke strežnika.

V nadaljevanju sledi psevdokoda strukture tipa *Get Profile List Response* in tabela 10, ki vsebuje podrobnejši opis posameznih elementov.

Get Profile List Response – *ListOf*(9)
{
Identifikacija strežnika – *Octet String*
Aktivacijski čas – *SML Time*
Perioda zajemanja – *Unsigned32*
Drevesna pot parametrov – *SML Tree Path*
Čas vrednosti – *SML Time*
Status – *Unsigned64*
Periodni seznam – *List of SML Period Entry*
S surovimi podatki – *Boolean* (opcijsko)
Periodni podpis – *SML Signature* (opcijsko)
}

Polje	Pomen
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika.
Aktivacijski čas	Polje, ki v protokolu SML nima opisa. Polje se v gonilniku protokola SML ne uporablja.
Perioda zajemanja	Polje, ki vsebuje periodo zajemanja podatkov.
Drevesna pot parametrov	Polje je razširjeni tip, ki je v bistvu samo seznam vrednosti *Octet String*. V seznamu se nahaja pot do podatkov, ki smo jih prejeli.
Čas vrednosti	Polje, ki vsebuje čas zapisa dogodka v knjigo dogodkov.
Status	Polje, ki vsebuje statusno besedo dogodka.
Periodni seznam	Polje je seznam objektov tipa *Period Entry*. Več zapisov iz tega seznama skupaj sestavlja en dogodek.

S surovimi podatki	V protokolu SML to polje nima opisa. Gonilnik protokola SML tega polja ne uporablja.
Periodni podpis	Podpis, ki se pri določenih dogodkih uporabi za overjanje sogovornika. Ta tip je identičen tipu *Octet String*.

Tabela 10: Opis elementov tipa *Get Profile List Response*.

2.4.3.8. Get Proc Parameter Request

Namen zahteve *Get Proc Parameter Request* je branje parametrov delovanja naprave (npr. branje ure, branje identifikacijske številke naprave, itd.).

V nadaljevanju sledi psevdokoda strukture tipa *Get Proc Parameter Request* in tabela 11, ki vsebuje podrobnejši opis posameznih elementov.

Get Proc Parameter Request – *ListOf*(5)
{
Identifikacija strežnika – *Octet String* (opcijsko)
Uporabniško ime – *Octet String* (opcijsko)
Geslo – *Octet String* (opcijsko)
Drevesna pot parametrov – *SML Tree Path*
Atribut – *Octet String* (opcijsko)
}

Polje	Pomen
Identifikacija strežnika	Polje vsebuje identifikacijska številko strežnika. Če to polje ni prisotno, potem je sporočilo namenjeno kot razpršeno oddajanje.
Uporabniško ime	Polje, ki vsebuje uporabniško ime in se uporablja za overjanje sogovornika.
Geslo	Polje, ki vsebuje geslo in se uporablja za overjanje sogovornika.
Drevesna pot parametrov	Polje je razširjeni tip, ki je v bistvu samo seznam vrednosti *Octet String*. V seznam je treba namestiti pot do podatkov, ki jih želimo prebrati.

Atribut	Polje je atribut, ki se ga lahko uporabi kot dodaten filter za podatke, ki jih zahtevamo.

Tabela 11: Opis elementov tipa *`Get Proc Parameter Request`*.

2.4.3.9. Get Proc Parameter Response

Namen odgovora *`Get Proc Parameter Response`* je vračanje parametrov delovanja naprave (npr. ure, identifikacijske številke naprave, itd.).

V nadaljevanju sledi psevdokoda strukture tipa *`Get Proc Parameter Response`* in tabela 12, ki vsebuje podrobnejši opis posameznih elementov.

`Get Proc Parameter Response` – *`ListOf`*(3)
{
Identifikacija strežnika – *`Octet String`*
Drevesna pot parametrov – *`SML Tree Path`*
Drevo parametrov – *`SML Tree`*
}

Polje	Pomen
Identifikacija strežnika	Polje, ki vsebuje identifikacijska številka strežnika.
Drevesna pot parametrov	Polje je razširjeni tip, ki je v bistvu samo seznam vrednosti *`Octet String`*. V seznam je treba namestiti pot do podatkov, ki jih želimo prebrati.
Drevo parametrov	Polje vsebuje razširjeni tip, ki je zelo prilagodljiv. V to polje lahko strežnik vstavi poljubno podatkovno strukturo.

Tabela 12: Opis elementov tipa *`Get Proc Parameter Response`*.

2.4.3.10. Set Proc Parameter Request

Namen zahteve *`Set Proc Parameter Request`* je nastavljanje parametrov delovanja naprave (npr. ure, itd.).

V nadaljevanju sledi psevdokoda strukture tipa *Set Proc Parameter Request* in tabela 13, ki vsebuje podrobnejši opis posameznih elementov.

Set Proc Parameter Request – *ListOf*(5)
{
    Identifikacija strežnika – *Octet String* (opcijsko)
    Uporabniško ime – *Octet String* (opcijsko)
    Geslo – *Octet String* (opcijsko)
    Drevesna pot parametrov – *SML Tree Path*
    Drevo parametrov – *SML Tree*
}

Polje	Pomen
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika. Če to polje ni prisotno, potem je sporočilo namenjeno kot razpršeno oddajanje.
Uporabniško ime	Polje, ki vsebuje uporabniško ime in se uporablja za overjanje sogovornika.
Geslo	Polje, ki vsebuje geslo in se uporablja za overjanje sogovornika.
Drevesna pot parametrov	Polje je razširjeni tip in je v bistvu samo seznam vrednosti *Octet String*. V seznam je treba namestiti pot do podatkov, ki jih želimo prebrati.
Drevo parametrov	Polje je razširjeni tip, ki je zelo prilagodljiv. V to polje bi lahko gonilnik protokola SML vstavil poljubno podatkovno strukturo, ki bi jo strežnik želel. Na žalost to tudi pomeni, da je način nastavljanja parametra lahko zelo drugačen od strežnika do strežnika.

Tabela 13: Opis elementov tipa *Set Proc Parameter Request*.

2.4.3.11. Attention Response

Namen odgovora *Attention Response* je odgovarjanje na zahteve *Set Proc Parameter Request* in odgovarjanje na napake. *Attention Response* je edini odgovor, ki se lahko nanaša na katero koli zahtevo.

V nadaljevanju sledi psevdokoda strukture tipa *Attention Response* in tabela 14, ki vsebuje podrobnejši opis posameznih elementov.

Attention Response – *ListOf*(4)
{
    Identifikacija strežnika – *Octet String*
    Številka opozorila – *Octet String*
    Sporočilo opozorila – *Octet String* (opcijsko)
    Drevesna pot parametrov – *SML Tree* (opcijsko)
}

Polje	Pomen
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika.
Številka opozorila	Polje, ki vsebuje številko opozorila in pove za kakšno opozorilo gre.
Sporočilo opozorila	Polje, ki lahko vsebuje dodatno sporočilo strežnika odjemalcu.
Detajli opozorila	Polje je razširjeni tip, ki je zelo prilagodljiv. V to polje bi lahko strežnik vstavil poljubno podatkovno strukturo.

Tabela 14: Opis elementov tipa *Attention Response*.

Sledi tabela 15, ki prikazuje različne vrednosti ter opise elementa »Število opozorila« do verzije protokola SML 1.04.

Število opozorila (šestnajstiško)	Pomen
…	Vse te vrednosti so trenutno rezervirane.
0x81 0x81 0xC7 0xC7 0xE0 0x00	S to vrednostjo se začnejo aplikacijske napake.

…	Vse te vrednosti so rezervirane za aplikacijske napake.
0x81 0x81 0xC7 0xC7 0xFC 0xFF	S to vrednostjo se končajo aplikacijske napake.
0x81 0x81 0xC7 0xC7 0xFD 0x00	Vrednost predstavlja OK/potrditev. To je opozorilo, ki ga prejmemo, če je nastavljanje parametra uspelo. S to vrednostjo se tudi začnejo pozitivna opozorila (niso napake).
0x81 0x81 0xC7 0xC7 0xFD 0x01	Vrednost pove, da bo zahteva izvedena kasneje in da bo odgovor odposlan kot odgovor brez zahteve.
…	Vse te vrednosti so trenutno rezervirane.
0x81 0x81 0xC7 0xC7 0xFD 0xFF	S to vednostjo se končajo pozitivna opozorila.
0x81 0x81 0xC7 0xC7 0xFE 0x00	Vrednost pomeni, da napake ni mogoče dodeliti nobeni drugi napaki definirani spodaj.
0x81 0x81 0xC7 0xC7 0xFE 0x01	Vrednost pove, da sogovornik ne prepozna oznake v prejetem sporočilu.
0x81 0x81 0xC7 0xC7 0xFE 0x02	Vrednost predstavlja nezadostno overitev. Kombinacija uporabniškega imena in gesla ni pravilna.
0x81 0x81 0xC7 0xC7 0xFE 0x03	Vrednost pove, da ciljni naslov (identifikacija strežnika) ni na voljo.
0x81 0x81 0xC7 0xC7 0xFE 0x04	Vrednost pove, da zahteva (identifikacija zahtevane datoteke) ni na voljo.
0x81 0x81 0xC7 0xC7 0xFE 0x05	Vrednost pove, da na enega ali več ciljnih naslovov ni mogoče pisati.
0x81 0x81 0xC7 0xC7 0xFE 0x06	Vrednost pove, da enega ali več ciljnih naslovov ni mogoče brati.
0x81 0x81 0xC7 0xC7 0xFE 0x07	Vrednost pove, da je bila komunikacija z merilno točko zmotena.
0x81 0x81 0xC7 0xC7 0xFE 0x08	Vrednost pove, da surovih podatkov ni mogoče

	razbrati.
0x81 0x81 0xC7 0xC7 0xFE 0x09	Vrednost pove, da je podana vrednost v prejetem sporočilu izven dovoljene zaloge vrednosti.
0x81 0x81 0xC7 0xC7 0xFE 0x0A	Vrednost pove, da zahteva ni bila izvedena (na primer, ker drevesna pot parametrov kaže na manjkajoč element).
0x81 0x81 0xC7 0xC7 0xFE 0x0B	Vrednost pove, da je bila kontrolna vsota napačna.
0x81 0x81 0xC7 0xC7 0xFE 0x0C	Vrednost pove, da razpršeno oddajanje ni podprto.
0x81 0x81 0xC7 0xC7 0xFE 0x0D	Vrednost pove, da je sogovornik prejel nepričakovano sporočilo SML (na primer datoteka SML brez zahteve *Open Request*).
0x81 0x81 0xC7 0xC7 0xFE 0x0E	Vrednost pove, da je objekt v profilu neznan (OBIS koda v zahtevi za profil kaže na vir podatkov, ki ni zapisan v profilu).
0x81 0x81 0xC7 0xC7 0xFE 0x0F	Vrednost pove, da tip v prejetem sporočilu ni podprt (na primer tip v zahtevi *Set Proc Parameter Request* je drugačen od pričakovanega).
0x81 0x81 0xC7 0xC7 0xFE 0x10	Vrednost pove, da opcijski element ni podprt (element definiran opcijsko v protokolu SML, ki je bil prejet, ni podprt).
0x81 0x81 0xC7 0xC7 0xFE 0x11	Vrednost pove, da zahtevani profil nima vnosa.
0x81 0x81 0xC7 0xC7 0xFE 0x12	Vrednost pove, da se je profil končal pred zahtevanim začetkom.
0x81 0x81 0xC7 0xC7 0xFE 0x13	Vrednost pove, da v območju profila ni nobenih vnosov. V drugih območjih leži vsaj en vnos.
0x81 0x81 0xC7 0xC7 0xFE 0x14	Vrednost pove, da se je datoteka SML zaključila brez zahteve *Close Request*.
0x81 0x81 0xC7 0xC7 0xFE 0x15	Vrednost pove, da profil trenutno ni na voljo (na primer, ker se v času zahteve profil preureja ali pa je vpis vnosa še potrebno izračunati).

…	Vse te vrednosti so trenutno rezervirane.

Tabela 15: Pomen različnih vrednosti elementa "Številka opozorila" .

2.4.3.12. Get List Request

Get List Request je zahteva, ki od strežnika zahteva že sestavljen seznam podatkov. Merilna naprava, ki sem jo uporabljal za testiranje gonilnika protokola SML, je odgovor na to sporočilo sama oddajala vsake tri sekunde brez vsakršne zahteve.

V nadaljevanju sledi psevdokoda strukture tipa *Get List Request* in tabela 16, ki vsebuje podrobnejši opis posameznih elementov.

Get List Request – *ListOf*(5)
{
Identifikacija odjemalca – *Octet String*
Identifikacija strežnika – *Octet String* (opcijsko)
Uporabniško ime – *Octet String* (opcijsko)
Geslo – *Octet String* (opcijsko)
Ime seznama – *Octet String* (opcijsko)
}

Polje	Pomen
Identifikacija odjemalca	Polje, ki vsebuje identifikacijsko številko odjemalca in se uporablja za povezovanje zahtev z odgovori.
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika. Če to polje ni prisotno, potem je sporočilo namenjeno kot razpršeno oddajanje.
Uporabniško ime	Polje, ki vsebuje uporabniško ime in se uporablja za overitev sogovornika.
Geslo	Polje, ki vsebuje geslo in se uporablja za overitev sogovornika.
Ime seznama	Polje, ki vsebuje ime seznama, ki ga želimo prebrati (OBIS koda).

Tabela 16: Opis elementov tipa *Get List Request*.

2.4.3.13. Get List Response

Get List Response je odgovor, ki vsebuje že sestavljen seznam podatkov. Merilna naprava, ki sem jo uporabljal za testiranje gonilnika protokola SML, je to sporočilo sama oddajala vsake tri sekunde.

V nadaljevanju sledi psevdokoda strukture tipa *Get List Response* in tabela 17, ki vsebuje podrobnejši opis posameznih elementov.

Get List Response – *ListOf*(7)
{
Identifikacija odjemalca – *Octet String* (opcijsko)
Identifikacija strežnika – *Octet String*
Ime seznama – *Octet String* (opcijsko)
Aktivacijski čas senzorja – *SML Time* (opcijsko)
Seznam vrednosti – *SML List*
Podpis seznama – *SML Signature* (opcijsko)
Aktivacijski čas vrat – *SML Time* (opcijsko)
}

Polje	Pomen
Identifikacija odjemalca	Polje, ki vsebuje identifikacijsko številko odjemalca, ki se uporablja za povezovanje zahtev z odgovori. Če to polje ni prisotno, potem je sporočilo namenjeno kot razpršeno oddajanje.
Identifikacija strežnika	Polje, ki vsebuje identifikacijsko številko strežnika.
Ime seznama	Polje, ki vsebuje ime seznama, ki ga odgovor vsebuje (OBIS koda).
Aktivacijski čas senzorja	Polje, ki vsebuje čas strežnika ob kreaciji tega sporočila.
Seznam vrednosti	Polje je razširjeni tip, ki vsebuje elemente tipa *List Entry*.
Podpis seznama	Polje, ki vsebuje podpis in se lahko uporabi za overjanje sogovornika. Ta tip je identičen tipu *Octet String*.
Aktivacijski čas vrat	Polje je opcijski atribut, ki lahko vsebuje aktivacijski čas senzorja. Vmesna naprava ali vrata lahko v to polje SML

odgovora vstavijo informacijo o svojem trenutnem času.

Tabela 17: Opis elementov tipa *Get List Response*.

2.4.3.14. SML Time

SML Time je najpogosteje uporabljen razširjeni tip. Zgrajen je tako, da omogoča zapis časa v treh različnih oblika.

V nadaljevanju sledi psevdokoda strukture tipa *SML Time* in tabela 18, ki vsebuje podrobnejši opis posameznih elementov.

SML Time – *ListOf*(2)
{
    Vrsta vsebine – *Unsigned8*
    Vsebina – Vsebina (razširjeni tip glede na vrsto vsebine)
}

Vrsta vsebine (šestnajstiško)	Vsebina
0x01	*Unsigned32* – Vsebina predstavlja sekundni indeks trajanja od nekega dogodka dalje.
0x02	*SML Timestamp* – Razširjeni tip, ki pa je drugače tipa *Unsigned32*. Vsebina predstavlja sekundni odmik od datuma 1.1.1970 00:00:00 v UTC formatu.
0x03	*SML Timestamp Local* – Razširjeni tip, ki poleg časa poda tudi časovni odmik lokalne cone in poletnega časa.

Tabela 18: Opis elementov tipa *SML Time*.

2.4.3.15. SML Timestamp Local

SML Timestamp Local je eden od možnih zapisov časa v tipu *SML Time*. Tip poleg časovne značke v UTC formatu poda tudi odmik lokalne cone in poletnega časa.

V nadaljevanju sledi psevdokoda strukture tipa *SML Timestamp Local* in tabela 19, ki vsebuje podrobnejši opis posameznih elementov.

SML Timestamp Local – ListOf(3)
{
Časovna značka – *SML Timestamp*
Lokalni odmik – *Integer16*
Sezonski odmik – *Integer16*
}

Polje	Pomen
Časovna značka	Razširjeni tip, ki pa je drugače tipa *Unsigned32*. Vsebina predstavlja sekundni odmik od datuma 1.1.1970 00:00:00 v UTC formatu.
Lokalni odmik	Vsebuje odmik lokalne cone od časa UTC v minutah.
Sezonski odmik	Vsebuje odmik poletnega časa od časa UTC v minutah.

Tabela 19: Opis elementov tipa *SML Timestamp Local*.

2.4.3.16. SML Period Entry

SML Period Entry je tip, ki je prilagojen za vračanje podatkov meritev. Zgrajen je tako, da vsebuje ime (OBIS kodo) elementa, enoto meritve, potenco meritve, vrednost in v nekaterih primerih tudi podpis za overitev strežnika.

V nadaljevanju sledi psevdokoda strukture tipa *SML Period Entry* in tabela 20, ki vsebuje podrobnejši opis posameznih elementov.

SML Period Entry – ListOf(5)s
{
Ime objekta – *Octet String*
Enota – *SML Unit*
Potenca – *Integer8*
Vrednost – *SML Value*
Podpis vrednosti – *SML Signature* (opcijsko)
}

Polje	Pomen
Ime objekta	Polje vsebuje OBIS kodo objekta, ki smo ga prejeli.
Enota	Polje vsebuje enota vrednosti, ki smo jo prejeli.
Potenca	Polje vsebuje potenco vrednosti, ki smo jo prejeli. Končna vrednost = vrednost * (10^potenca).
Vrednost	Polje je edinstveni razširjeni tip, saj je lahko katerega koli osnovnega tipa. Razlikuje se od drugih izbirnih tipov, ki so seznami dveh vrednosti, kjer prva vrednost napove drugo. Tip je implicitna izbira tipa, kar pomeni, da bo glava tipa `SML Value` enaka glavi nekega osnovnega tipa.
Podpis vrednosti	Podpis, ki se lahko uporabi za overjanje sogovornika. Ta tip je identičen tipu `Octet String`.

Tabela 20: Opis elementov tipa *`SML Period Entry`*.

2.4.3.17. SML Tree

`SML Tree` je zelo prilagodljiv tip. S tem tipom je mogoče predstaviti različne informacije na več načinov, kar pa je tudi njegova šibka točka, saj je nastavljanje parametrov strežnika lahko od implementacije do implementacije drugačno.

V nadaljevanju sledi psevdokoda strukture tipa `SML Tree` in tabela 21, ki vsebuje podrobnejši opis posameznih elementov.

`SML Tree` – `ListOf`(3)
{
    Ime parametra – `Octet String`
    Vrednost parametra – `SML Proc Par Value` (opcijsko)
    Seznam otrok – `List of SML Tree` (opcijsko)
}

Polje	Pomen
Ime objekta	Polje vsebuje OBIS kodo objekta, ki smo ga prejeli.
Vrednost parametra	Polje je razširjeni tip, ki omogoča več vrst vrednosti.
Seznam otrok	Polje je razširjeni tip, ki je v bistvu samo seznam elementov tipa *SML Tree*.

Tabela 21: Opis elementov tipa *SML Tree*.

2.4.3.18. SML Proc Par Value

SML Proc Par Value je izbirni tip, ki omogoča večji izbor vrednosti, kot sam tip *SML Value*.

V nadaljevanju sledi psevdokoda strukture tipa *SML Proc Par Value* in tabela 22, ki vsebuje podrobnejši opis posameznih elementov.

SML Proc Par Value – *ListOf*(2)
{
    Vrsta vsebine – *Unsigned8*
    Vsebina – Vsebina (razširjeni tip glede na vrsto vsebine)
}

Vrsta vsebine (šestnajstiško)	Vsebina
0x01	*SML Value* je edinstveni razširjeni tip, saj je lahko katerega koli osnovnega tipa. Razlikuje se od drugih izbirnih tipov, ki so seznami dveh vrednosti, kjer prva vrednost napove drugo. Tip je implicitna izbira tipa, kar pomeni, da bo glava tipa *SML Value* enaka glavi nekega osnovnega tipa.
0x02	*SML Period Entry* je razširjeni tip, ki vsebuje vrednost nekega prebranega registra.
0x03	*SML Tupel Entry* je razširjeni tip, ki je ostanek nekega starejšega protokola in se ne uporablja več.

0x04	*SML Time* je razširjeni tip v katerem so zapisani različni tipi časa v protokolu SML.
0x05	*SML List Entry* je razširjeni tip podoben tipu *SML Period Entry* z nekaterimi dodatnimi polji.

Tabela 22: Opis elementov tipa *SML Proc Par Value*.

2.4.3.19. SML List Entry

Tip *SML List Entry* je zelo podoben tipu *SML Period Entry* z nekaterimi dodatnimi polji. Poleg elementov, ki si jih deli s tipom *SML Period Entry*, vsebuje še status in čas vrednosti. Gre za bolj preprost tip, saj je večina polj opcijskih.

V nadaljevanju sledi psevdokoda strukture tipa *SML List Entry* in tabela 23, ki vsebuje podrobnejši opis posameznih elementov.

SML List Entry – *ListOf*(7)
{
Ime objekta – *Octet String*
Status – *SML Status* (opcijsko)
Čas vrednosti – *SML Time* (opcijsko)
Enota – *SML Unit* (opcijsko)
Potenca – *Integer8* (opcijsko)
Vrednost – *SML Value*
Podpis vrednosti – *SML Signature* (opcijsko)
}

Polje	Pomen
Ime objekta	Polje vsebuje OBIS kodo objekta, ki smo ga prejeli.
Status	Polje je razširjeni tip, ki je podoben tipu *SML Value*. Tudi ta tip ni običajni izbirni tip (seznam dveh elementov kjer prvi določa drugega) ampak je katerikoli od tipov *Unsigned8/16/32/64* podan implicitno (glava direktno poda vrsto sledečega tipa).
Čas vrednosti	Polje, ki vsebuje čas vrednosti.

Enota	Polje, ki vsebuje enoto prejete vrednosti.
Potenca	Polje, ki vsebuje potenco prejete vrednosti. Končna vrednost = vrednost * (10^potenca).
Vrednost	Polje je edinstveni razširjeni tip, saj je lahko katerega koli osnovnega tipa. Razlikuje se od drugih izbirnih tipov, ki so seznami dveh vrednosti, kjer prva vrednost napove drugo. Tip je implicitna izbira tipa, kar pomeni, da bo glava tipa `SML Value` enaka glavi nekega osnovnega tipa.
Podpis vrednosti	Polje vsebuje podpis, ki se lahko uporabi za overjanje sogovornika. Ta tip je identičen tipu `Octet String`.

Tabela 23: Opis elementov tipa *`SML List Entry`*.

2.5. Transportni protokol SML

Specifikacija protokola SML vsebuje tudi definicijo transportnega dela protokola SML. Transportni protokol se uporablja za nezavarovane povezave od točke do točke (npr. direktno branje preko optične sonde, preko PSTN modemov, GSM modemov, itd.) in tudi za zavarovane, kot je na primer TCP. Omogoča razbijanje in sestavljanje podatkov v bloke ter preprosti nadzor pravilnosti s CRC. Transportni protokol uporablja ubežno zaporedje, da napove, da sledi ukaz transportnega protokola SML. Pred datoteko SML postavi ukaz za normalen prenos ali pa bločni prenos. Na konec datoteke SML doda toliko bajtov z vrednostjo 0x00 (šestnajstiško), da je skupno število bajtov datoteke SML deljivo s štiri, na samem koncu pa pošlje še ukaz za zaključek prenosa. Ukaz za zaključek prenosa vsebuje informacijo o številu dodanih bajtov in kodo CRC. V tabeli 24 je mogoče videti ubežna zaporedja do specifikacije protokola SML 1.04.

Ukaz (šestnajstiško)	Pomen
0x1B 0x1B 0x1B 0x1B	Gre za ubežno zaporedje, ki napoveduje sledeče ukaze. Če se ubežno zaporedje pojavi v datoteki SML sami, ga transportni protokol podvoji.
0x01 0x01 0x01 0x01	Ukaz, ki ponazarja začetek normalnega prenosa.

0x02 0xTT 0xUU 0xVV	Ukaz, ki ponazarja začetek bločnega prenosa. Skrajno levi bit bajta 0xTT pove smer prenosa bloka (0 – zahteva, 1 – odgovor oz. potrditev), njegov sosed pa pove, če je to zadnji blok (1 – zadnji blok). Vsi ostali biti skupaj sestavljajo zaporedno številko bloka. Če številka bloka preseže zalogo vrednosti, začne ponovno z ena. Bajt 0xUU pove, po koliko različnih poteh se je sporočilo že preneslo. Bajt 0xVV je trenutno rezerviran za prihodnjo uporabo. Bajt ima vrednost vedno enako 0x01 (šestnajstiško).
0x03 0x00 0xRR 0xRR	Ukaz, ki pove prejemniku, kolikšna je časovna omejitev za odgovor.
0x04 0x00 0xSS 0xSS	Ukaz, ki pove prejemniku, kakšna je predlagana največja velikost bloka.
0x1A 0xXX 0xYY 0xZZ	Ukaz, ki ponazarja zaključek prenosa. Bajt 0xXX pove, koliko bajtov smo dodali na konec datoteke SML, da je bila celotna dolžina deljiva s štiri. Veljavne vrednosti so od 0x00 do 0x03 (šestnajstiško). Bajta 0xYY in 0xZZ sta 32 bitni CRC.

Tabela 24: Tabela ubežnih zaporedij transportnega protokola SML.

3. Izdelava knjižnice

Knjižnica je narejena za operacijski sistem *Windows* in zahteva *.Net Framework 4.0*. Napisana je v razvojnem okolju *Visual Studio 2008/2010* v jeziku C#. Knjižnica je že kar nekaj časa v izdelavi, saj se strojna programska oprema strežnika (števca) skupaj s protokolom SML še vedno spreminja.

3.1. Orodja

Orodja, ki sem jih uporabil za razvoj gonilnika protokola SML, bi lahko razdelili na dve različni skupini. V prvo skupino spadajo orodja, ki so mi omogočala oziroma olajšala razvoj programske opreme. To sta aplikaciji *Visual Studio 2008* in kasneje *Visual Studio 2010*. V drugo skupino bi spadala orodja, ki so mi pomagala pri popravljanju napak. V največjo pomoč mi je bil seveda dejanski strežnik protokola SML (števec). Pomagala mi je tudi aplikacija, ki je simulirala strežnik protokola SML v času, ko strežnika protokola SML še nisem imel. Pomagal pa sem si tudi z aplikacijo *Port Monitor*, ki mi je omogočala pregled prenosa podatkov preko zaporednih vrat. Seveda pa bilo samo testiranje nemogoče brez strežniške aplikacije MAS ter aplikacije *MAS Client* in kasneje *SEP2W System*, s pomočjo katerih sem lahko pošiljal dejanske zahteve knjižnici.

3.1.1. Visual Studio 2008

Visual Studio je brez dvoma eno bolj prijaznih okolij za razvoj aplikacij v operacijskem sistemu Windows [7]. Aplikacija spada v skupino integriranih razvojnih okolij (IDE). Podpira prevajanje in urejevanje več jezikov za razvoj programske opreme (C++, C#, Visual Basic, itd.), a je mogoče dodati še druge. Urejevalnik zelo olajša razvoj z orodji, kot so samodejno izpolnjevanje kode, označevanje sintakse, ustvarjanje zaznamkov, iskanje s pomočjo regularnih izrazov, itd. Čeprav ga sam v tej knjižnici nisem potreboval, moram omeniti tudi orodje za vizualni dizajn aplikacij, s pomočjo katerega je mogoče v trenutku narediti uporabniške vmesnike. Vsebuje tudi vgrajeni razhroščevalnik z veliko uporabnimi funkcijami, kot so pregled vrednosti spremenljivke, pogled vrednosti v različnih številskih sestavih, pregled sklada, itd. Vsebuje tudi zelo uporabno in poučno orodje, ki omogoča analizo kode. Orodje prikaže nevarnosti v kodi in predlaga rešitve. *Visual Studio* lahko tudi vsebuje vgrajeno aplikacijo *Team System*, ki omogoča nadzor nad verzijami. Poleg tega pa omogoča tudi izdelovanje oz. dodajanje orodij tretjih oseb. Razvoj gonilnika protokola SML je potekalo v jeziku C#.

3.1.2. Visual Studio 2010

Visual Studio 2010 je novejša različica orodja *Visual Studio 2008* [6]. Najbolj očitna sprememba je uporabniški vmesnik aplikacije. Tokrat je dizajn veliko bolj čist in preprost. Poleg tega pa kodni urejevalnik olajša delo z več dokumenti, saj omogoča zelo preprosto kombiniranje pogledov na odprte dokumente. Krasen je tudi novi urejevalnik dodatkov, ki je sedaj kar vgrajen v samo aplikacijo. Menim, da aplikacija tudi deluje malo hitreje.

3.1.3. Strežnik protokola SML (števec)

Števec, ki je narejen v podjetju Iskraemeco, je narejen v večini po načrtih tujega števca. Števec je pričel z razvojem malo za razvojem gonilnika protokola SML. Strojno programsko opremo števca razvija druga skupina. S skupnim razvojem v različnih razvojnih skupinah je mogoče medsebojno preizkušanje programske opreme. Ker se protokol SML še vedno razvija in ker se še vedno dodajajo funkcionalnosti števcu, je potrebno gonilnik protokola SML še vedno popravljati oziroma razvijati.

3.1.4. Simulator strežnika protokola SML

Ker se razvoj števca ni pričel z razvojem gonilnika protokola SML, sem moral v začetku programirati brez preizkušanja. Vse kar sem dobil v vpogled o protokolu SML razen specifikacije protokola je bila binarna datoteka nekaterih zahtev. Mentor v podjetju je zato naredil preprost simulator strežnika SML, da sem lahko preizkušal knjižnico. Ker je simulator strežnika SML naredila druga oseba, sem že v začetku našel veliko domnevnih napak zaradi različnih interpretacij specifikacij protokola. Razvoj simulatorja se je ustavil s prihodom prve verzije strežnika protokola SML.

3.1.5. Port Monitor

Port Monitor je aplikacija, s pomočjo katere sem lahko nadzoroval pretok podatkov na zaporednih vratih. Z aplikacijo sem lahko preverjal na kateri strani komunikacije ležijo napake v protokolu.

3.1.6. Strežniška aplikacija MAS

Strežniška aplikacija MAS omogoča drugim aplikacijam, kot sta *MAS Client* ter *SEP2W System*, komunikacijo s strežniki (npr. števci, koncentratorji, itd.). Strežniška aplikacija sprejema zahteve drugih aplikacij preko dokumentov XML. Dokumente nato posreduje transportnim gonilnikom ter gonilnikom protokolov v obliki *MasData*.

Opisal bom približek dokumenta XML, ki ga prejme strežniška aplikacija MAS. Dokument XML je sestavljen iz treh logičnih delov. Element *CommunicationPool* vsebuje nastavitve komunikacijskega protokola. V Elementu *Operations* se nahajajo definicije operacij, ki jih lahko strežniška aplikacija MAS posreduje gonilniku protokola. Element drži le definicije teh operacij in ne zahteva od strežniške aplikacije MAS podatke o tem katere od njih se morajo izvesti. V zadnjem elementu *MeteringDevices* se nahajajo nastavitve gonilnika protokola in operacije, ki se bodo izvedle. Ista operacija se lahko pojavi več kot enkrat. Operacije se izvajajo v vrstnem redu v katerem so navedene. Sledi primer (psevdokoda) zahteve v obliki dokumenta XML.

```
<MASRequest ID="1">
	<CommunicationPool>
		<CommunicationDevice Name="Komunikacijski protokol" />
		<Property Name="Nastavitev 1" Value="Vrednost" />
		...
		<Property Name="Nastavitev N" Value="Vrednost" />
	</CommunicationPool>
	<Operations>
		<Invoke ID="1" ClassID="Kategorija funkcije" InstanceID="OBIS koda"
MethodID="Funkcija">
			<Parameters>
				<Tip_parametra>Vrednost</Tip_parametra>
				...
				<Tip_parametra>Vrednost</Tip_parametra>
			</Parameters>
		</Invoke>
		...
		<Invoke ID="N" ClassID="Kategorija funkcije" InstanceID="OBIS koda"
MethodID="Funkcija" />
	</Operations>
	<MeteringDevices>
		<MeteringDevice ID="1" Driver="MeterSML">
			<Property Name="MsgIdentSize" Value="16" />
			...
			<Property Name="ErrorOnInvalidSignature" Value="true" />
			<Operation ID="1" />
			...
			<Operation ID="N" />
		</MeteringDevice>
	</MeteringDevices>
</MASRequest>
```

3.1.7. MAS Client

Aplikacija je v bistvu zelo preprost uporabniški vmesnik za strežniško aplikacijo MAS. Zmožna je povezovanja s strežniško aplikacijo MAS, pošiljanja zahtev v obliki XML dokumenta (zahteve mora napisati uporabnik sam), lovljenja ter prikazovanja dogodkov

strežniške aplikacije MAS, ki se zgodijo med izvajanjem neke zahteve, ter prikazovanje rezultatov izvedene zahteve v surovi obliki XML dokumenta.

3.1.8. SEP2W System

`SEP2W System` je večje število aplikacij in je glavni uporabnik strežniške aplikacije MAS. Strežniška aplikacija MAS je le ena od mnogih strežniških aplikacij, ki jih `SEP2W System` uporablja. Poleg komunikacije s pametnimi napravami, ki jo omogoča strežniška aplikacija MAS, omogoča še agregacijo podatkov, pomnjenje podatkov, poročanje o podatkih, avtomatizacijo raznih opravil in še mnogo več.

3.2. Arhitektura knjižnice

Gonilnik protokola SML uporablja tri različne vrste razredov. *Delovni razredi* so razredi, ki se ne navezujejo na protokol SML, ampak bolj na logiko komunikacije gonilnika protokola SML s strežniško aplikacijo MAS ter strežnikom protokola SML. *Osnovni tipi* so razredi, ki se zelo navezujejo na protokol SML. Razredi vsebujejo poleg vrednosti tudi metode za pretvorbo iz/v tabele bajtov ter razširjene razrede razreda `MasData`. *Razširjeni tipi* so v gonilniku protokola SML vedno razširjeni tipi tipa `ListOf`. Njihov namen je bolj protokolu SML podobno in preprosto programiranje, kot pa s samo osnovnimi tipi. Čeprav protokol SML vsebuje tudi razširjene tipe, ki niso razširjeni iz razreda `ListOf`, jih v svojo knjižnico nisem vključil, saj se mi je zdelo to potratno. Uporabil sem kar osnovne razrede, če so bili razširjeni tipi enaki osnovnim. Kjer pa je tip omogočal več različnih osnovnih tipov (implicitno podani tipi), sem uporabil generični razred `SmlData`.

3.2.1. Delovni razredi

Delovanje gonilnika protokola SML poteka preko naslednjih razredov:

- `SmlDriver`,
- `SignatureCalculator`,
- `SmlFileCollection`,
- `SmlFile`,
- `SmlMessageCreator`.

Glavni razred se imenuje `SmlDriver`, ki razširja razred `BaseDriver`. Med njegove naloge spada prejemanje ter pošiljanje podatkov, pretvorba zahtev strežniške aplikacije MAS v enakovredne datoteke SML, poročanje poteka komunikacije ter preverjanje pravilnosti

podpisa prejetih sporočil. Gonilnik protokola SML vključuje v komunikacijo še transportni protokol SML. Transportni protokol SML bi moral biti ločen od gonilnika protokola SML, ker pa ni potrebe po drugem transportnem protokolu, se zaradi preprostosti kar vedno vključuje v komunikacijo.

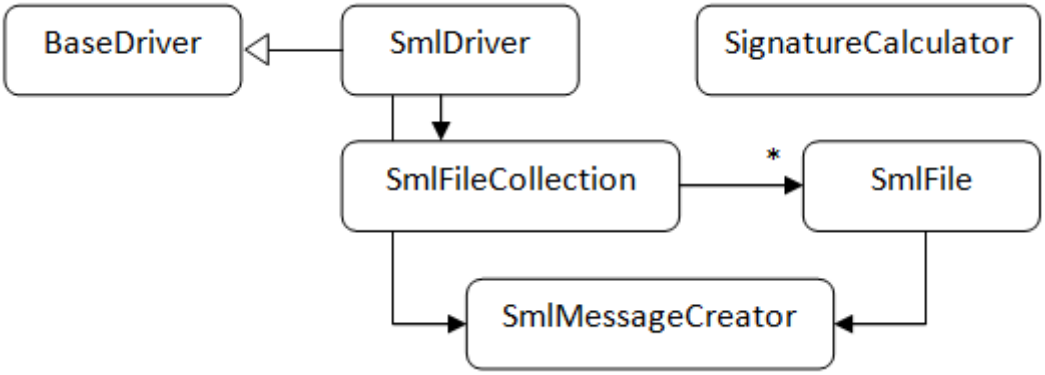
SmlFileCollection je razred, ki vsebuje seznam vseh datotek SML tipa *SmlFile*, ki smo jih kreirali za zahteve strežniške aplikacije MAS ter metode, ki omogočajo povezavo zahteve strežniške aplikacije MAS z datoteko SML. Namen razreda je omogočati razredu *SmlDriver* tem lažjo povezavo med rezultati zahteve ter zahtevo samo, medtem ko skriva podrobnosti upravljanja z zahtevami ter datotekami.

SmlFile je razred, ki na zunaj izgleda zelo podobno, kot datoteka SML sama. *SmlFile* razred vsebuje metode, ki omogočajo preprosto dodajanje/odstranjevanje sporočil SML ter iskanje rezultatov zahteve znotraj datoteke (spomnimo se, da lahko ena datoteka SML vsebuje več kot eno zahtevo/odgovor). *SmlFile* razred tudi sam poskrbi za zahtevi *SML Open Request* ter *SML Close Request*. Namen razreda je omogočati razredu *SmlFileCollection* preprosto iskanje rezultatov zahteve znotraj datoteke, medtem ko skriva podrobnosti upravljanja z zahtevami znotraj datoteke.

SmlMessageCreator je razred katerega enak primerek vsebujeta tako *SmlDriver*, kot *SmlFile*. Namen razreda je združiti kreacijo sporočil SML v enem razredu ter odstraniti potrebo drugih razredov po sledenju edinstvenih vrednosti elementov sporočil SML, kot na primer transakcijsko številko sporočila SML.

SignatureCalculator je statični razred, ki vsebuje metode za preverjanje pravilnosti podpisov sporočil SML. Razred uporablja prosto dostopno knjižnico za računanje kriptografije (dostopno na http://www.bouncycastle.org/). *SmlDriver* s pomočjo tega razreda preveri podpis ter vrne napako v primeru nepravilnega podpisa. Preverjanje se izvaja le, če nastavitve to zahtevajo.

Medsebojni odnosi delovnih razredov so lepše vidni na sliki 5. Celotna slika razrednega diagrama je vidna na sliki 7.



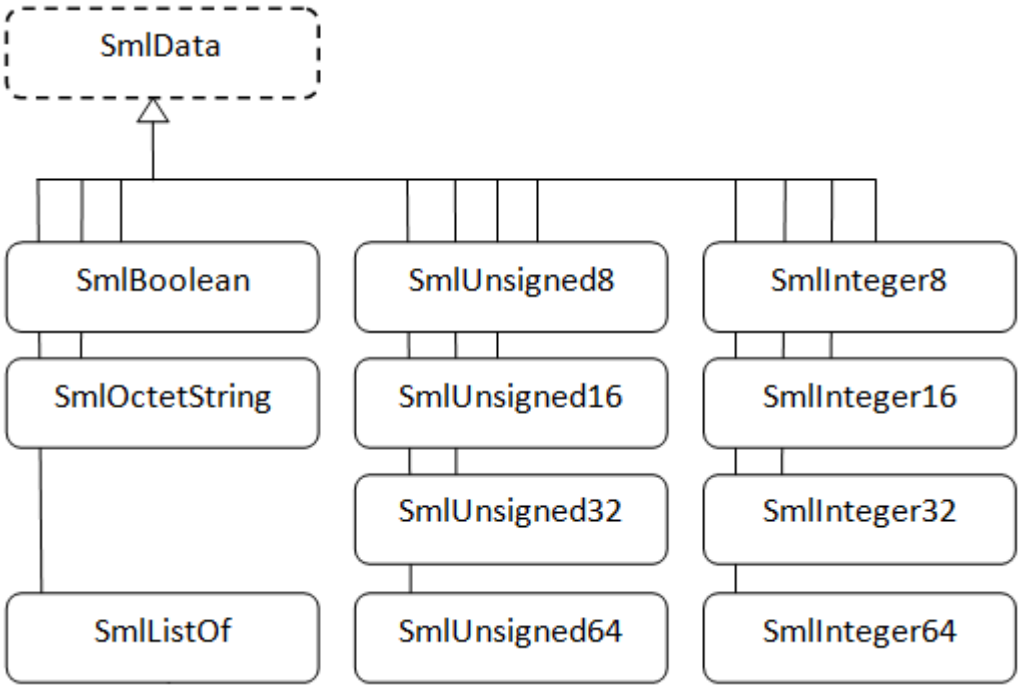
Slika 5: Preprost razredni diagram delovnih razredov.

3.2.2. Osnovni tipi

Osnovni tipi vsebujejo vse osnovne tipe protokola SML, ter generični abstraktni razred `SmlData`.

`SmlData` je abstraktni razred, ki ga vsi ostali osnovni tipi razširjajo. Vsebuje nekaj metod, ki so skupne vsem osnovnim tipom, kot na primer pregled dolžine tipa iz prebrane glave v tabeli bajtov ali iz tipa `MasData`. Vsebuje pa tudi nekaj abstraktnih metod, ki jih morajo osnovni tipi podpreti, kot na primer vračanje vrednosti tipa. Vsebuje tudi statične metode, ki se uporabljajo za razbijanje podatkov v obliko, ki jo potrebujemo za preverjanje podpisa.

Osnovni tipi in njihovi odnosi s tipom `SmlData` so vidni na sliki 6. Celotna slika razrednega diagrama je vidna na sliki 7.

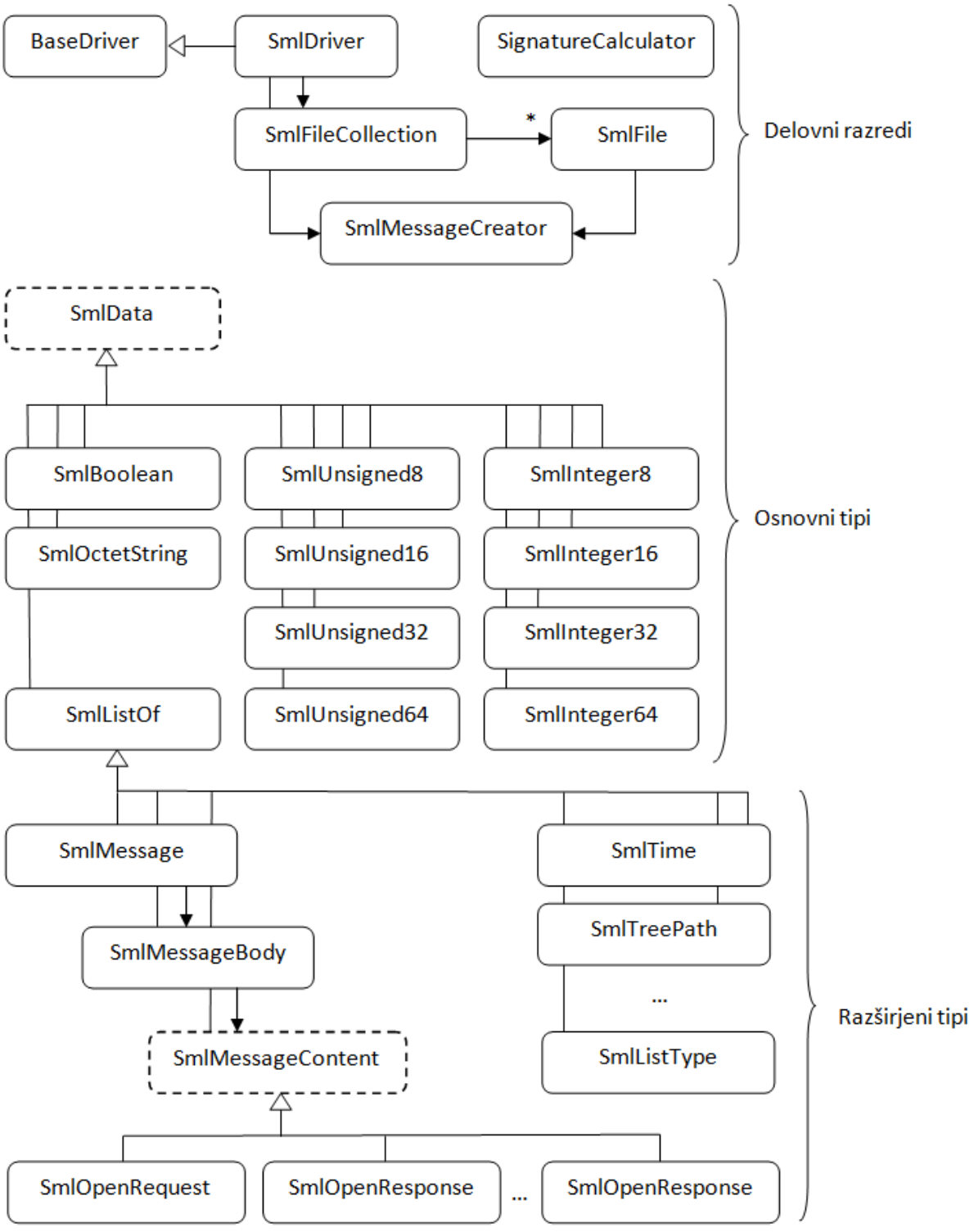


Slika 6: Preprost razredni diagram osnovnih tipov.

3.2.3. Razširjeni tipi

Razširjeni tipi so v gonilniku protokola SML vsi razredi razširjeni iz razreda `ListOf`. Tipi omogočajo malo lažje - direktno dostopanje do svojih elementov in jih vračajo v pravem tipu (v tipu `ListOf` so vsi otroci tipa `SmlData`). Med razširjenimi tipi obstaja razred, ki v protokolu SML ne obstaja. To je razred `SmlMessageContent`. Razred se uporablja kot element razreda `SmlMessageBody`. Vsa sporočila (npr. `SML Open Request`) razširjajo ta razred. Namen tega razreda je, da je mogoče v `SmlMessageBody` namestiti katero koli sporočilo, a ga še vedno bolje omejiti, kot z elementom tipa `SmlData`.

Slika razširjenih tipov ter njihovih odnosov z ostalimi tipi je mogoče videti na sliki 7.



Slika 7: Preprost celoten razredni diagram gonilnika protokola SML.

3.2.4. Nastavitve gonilnika protokola SML

Gonilnik protokola SML vsebuje tudi nekaj nastavitev. Nekatere med njimi lahko z napačnimi vrednostmi preprečujejo uspešno komunikacijo, nekatere pa le olajšajo dela z gonilnikom. Gonilnik protokola SML podpira naslednje nastavitve:

- *NumOfMsgPerFile*,
- *MsgIdentSize*,
- *WaitReceiveBeforeSend*,
- *ClientID*,
- *ServerID*,
- *Username*,
- *Password*,
- *PublicKey*,
- *AutoEDL40Mode*,
- *ErrorOnInvalidSignature*.

3.2.4.1. NumOfMsgPerFile

Nastavitev pove gonilniku protokola SML kolikšno je največje število sporočil, ki jih lahko vstavi v eno samo datoteko SML. Nastavitev ima lahko hude posledice na strežniški strani komunikacije. Strežniki oziroma števci za katere je ta protokol namenjen so preproste naprave z omejeno količino virov. Prevelika količina sporočil v eni sami datoteki SML lahko povzroči napako na strežniku. Privzeta vrednost nastavitve je 1 (v tem primeru bodo v datoteki le sporočila *SML Open Request*, trenutna zahteva in *SML Close Request*).

3.2.4.2. MsgIdentSize

Čeprav so v specifikaciji protokola SML izbirne vrednosti različnih sporočil SML prikazane kot 32 bitne vrednosti, obstaja neko nenapisano pravilo. Vrednosti, ki so v specifikaciji zapisane z večjim tipom kot pa je za predstavitev vseh vrednosti potreben, je mogoče zapisati z manjšim tipom. Razen v primeru izbire tipa sporočila nisem zasledil primera, kjer bi se to

pravilo še uporabljalo. Privzeta vrednost te nastavitve je 16 bitov, kar pomeni, da bo vrsta sporočila zapisana s tipom *Unsigned16* in ne s tipom *Unsigned32*.

3.2.4.3. WaitReceiveBeforeSend

Nastavitev pove gonilniku protokola SML, če mora pred pošiljanjem podatkov pričakovati tok podatkov strežnika. Glede na komunikacijsko pot s strežnikom je mogoče dobiti brez zahteve sporočilo na vsake tri sekunde. Ta nastavitev preprečuje možnost, da bi gonilnik protokola SML pričel s pošiljanjem ob istem času, ko pošilja podatke že strežnik. Gonilnik bo počakal dokler ne prejme celotne datoteke SML pred prvim pošiljanjem. Strežnik je narejen tako, da kadar prejme zahtevo odjemalca ponovno nastavi čas do naslednjega toka podatkov na tri sekunde. Čakanje je potrebno le ob prvi zahtevi. Po privzeti vrednosti te nastavitve se na pošiljanje ne čaka.

3.2.4.4. ClientID

ClientID je podatek, ki se uporablja le za povezovanje zahtev z odgovori. Privzeta vrednost nastavitve je naključna vrednost, ki se generira ob zagonu gonilnika protokola SML.

3.2.4.5. ServerID

ServerID je podatek, ki je na strani zahtev opcijski. Če podatka ne vključimo, bo gonilnik protokola SML na mesto elementa strežniške identifikacije vstavil prazno polje, ki predstavlja razpršeno oddajanje.

3.2.4.6. Username in Password

Uporabniško ime in geslo v nasprotju s pričakovanjem nista potrebna podatka. Večino branja je mogoče narediti brez overitve. Za pisanje pa sta podatka potrebna. Privzeta vrednost polij je prazno polje.

3.2.4.7. PublicKey

Nastavitev omogoča uporabniku vnos javnega ključa strežnika. Javni ključ je potreben za overitev sogovornika. Če ta nastavitev ni podana, se bo javni ključ prebral sam ob zagonu gonilnika protokola SML.

3.2.4.8. AutoEDL40Mode

Ta nastavitev olajša uporabnikovo delo s strežnikom, saj omogoča avtomatsko nastavljanje parametra števca, ki drugače preprečuje nastavljanje ure. Nastavitev ne bo nastavljala tega parametra vsakih nekaj minut kot bi bilo potrebno, da se nastavljena ura ne bi zbrisala. Po privzeti vrednosti te nastavitve mora uporabnik sam nastaviti parameter števca za pisanje ure.

3.2.4.9. ErrorOnInvalidSignature

Če je ta nastavitev nastavljena, potem bo gonilnik protokola SML ob neveljavnem podpisu sporočila vrnil napako. Po privzeti vrednosti te nastavitve se preverjanje podpisa ne izvaja.

3.2.5. Funkcije gonilnika protokola SML

Gonilnik protokola SML omogoča komunikacijo z nekaj vnaprej določenimi funkcijami. Te funkcije se razdelijo na štiri kategorije:

- *SML Raw Data* – funkcije te kategorije predstavijo svoje rezultate v surovi obliki protokola SML (rezultat teh funkcij je notranjost sporočila npr. objekt tipa *SML Get Proc Parameter Response*),
- *SEP2 Clock* – funkcije te kategorije predstavijo svoje rezultate v standardni obliki, ki si jo delijo z ostalimi gonilniki protokolov,
- *SEP2 Register* – funkcije te kategorije predstavijo svoje rezultate v standardni obliki, ki si jo delijo z ostalimi gonilniki protokolov,
- *SEP2 Event Profile* – funkcije te kategorije predstavijo svoje rezultate v standardni obliki, ki si jo delijo z ostalimi gonilniki protokolov.

3.2.5.1. SML Raw Data

V kategorijo *SML Raw Data* spada največje število funkcij. Vse funkcije, ki spadajo v ostale kategorije so samo drugačna predstavitev rezultatov teh funkcij. V to kategorijo spadajo funkcije:

- *Read Parameter*,
- *Write Parameter*,
- *Get Profile List*,

- *Read Parameter List*,
- *Read Telegram*.

3.2.5.1.1. Read Parameter

Read Parameter je drugo ime za zahtevo *SML Get Proc Parameter Request*. Funkcija potrebuje kot vhodne parametre le OBIS kodo objekta, ki ga želimo prebrati. Vsi ostali podatki razen polje atributa so prisotni že v nastavitvah gonilnika protokola SML. Polje atribut se pri trenutnemu strežniku ne uporablja.

3.2.5.1.2. Write Parameter

Write Parameter predstavlja zahtevo *SML Set Proc Parameter Request*. Funkcija potrebuje poleg OBIS kode objekta, ki ga želimo nastaviti, še vrednost, ki jo bomo zapisali v objekt. Vrednost je podana v obliki razširjenega razreda *MasData*. To vrednost je potrebno pretvoriti v enega od osnovnih tipov SML in zapakirati v drevo parametrov. Vsi ostali podatki so prisotni že v nastavitvah gonilnika protokola SML.

3.2.5.1.3. Get Profile List

Get Profile List funkcija uporablja zahtevo *SML Get Profile List Request*. Za delovanje potrebuje OBIS kodo objekta, ki ga želimo prebrati in začetni ter končni čas branja. Polja s surovimi podatki, seznam objektov ter detajli se ne uporabljajo. Vsi ostali podatki so prisotni že v nastavitvah gonilnika protokola SML.

3.2.5.1.4. Read Parameter List

Read Parameter List funkcija uporablja zahtevo *SML Get List Request*. To sporočilo je zelo preprosto. Kot vhodni parameter je potrebno podati le OBIS kodo objekta. Vsi ostali podatki so prisotni že v nastavitvah gonilnika protokola SML.

3.2.5.1.5. Read Telegram

Read Telegram je posebna funkcija, ki ne uporablja nobene zahteve. Ne sprejme nobenih parametrov, saj samo počaka na naslednjo samodejno oddajo sporočila strežnika.

3.2.5.2. SEP2 Clock

Funkcije te kategorije imajo opravka z uro. Te funkcije ne potrebujejo OBIS kode, saj je OBIS koda za objekt ure nespremenljiva v strežniku. Lahko pa jo podamo, če želimo oziroma, če določen strežnik vsebuje več kot eno uro. Funkcije, ki spadajo v to kategorijo:

- *Read*,
- *Synchronize*,
- *Set*,
- *Get Time Difference*.

3.2.5.2.1. Read

Read funkcija prebere s pomočjo zahteve *SML Get Proc Parameter* objekt ure. Funkcija sama po sebi ne zahteva OBIS kode objekta, saj je le ta nespremenljiva. Lahko pa jo podamo. Strežnik v primeru, da ura ni nastavljena vrne napako. Funkcija mora tudi vrniti rezultate v bolj pregledni obliki. Rezultati se morajo nahajati v posebni strukturi SEP2 Register.

3.2.5.2.2. Synchronize

Synchronize funkcija poskusi prebrati uro, pogledati razliko med strežnikom in odjemalcem in nastaviti uro na odjemalčevo, če se ura razlikuje za več ali manj kot interval podan v parametrih funkcije. Problemi nastanejo s strežnikovim načinom upravljanja z uro. Strežnik vsebuje objekt, ki strežniku določa ali je uro mogoče nastaviti. Poleg tega se ta objekt vsakih neka minut nastavlja na privzeto vrednost »nedovoljeno«. Ko se objekt nastavi nazaj na privzeto vrednost se izbriše tudi ura. Zato gonilnik protokola SML vsebuje nastavitev s pomočjo katere se objekt nastavi samodejno za tiste objekte, za katere je to potrebno. V primeru, da strežnik vrne napako, se ura vedno nastavlja. Funkcija potrebuje dva parametra, ki določata začetek in konec intervala za sinhronizacijo (prvi parameter pove vsaj kako velika mora biti razlika v uri, da je sinhronizacija smiselna, medtem ko drugi pove koliko je največja razlika do katere še želimo sinhronizacijo). Če želimo, lahko podamo tudi OBIS kodo objekta, ki ga želimo nastaviti. *Synchronize* vrne kot rezultat razliko med časom odjemalca in strežnika.

3.2.5.2.3. Set

`Set` funkcija nastavi uro na želeno vrednost. Tako kot pri sinhronizaciji, tudi tukaj lahko pride do problemov z nastavljanjem zaradi objekta, ki omogočati nastavljanje. Ampak gonilnik protokola SML omogoča nastavitev, ki bo zagotovila, da se bo ta objekt nastavil samodejno. Če želimo lahko podamo tudi OBIS kodo objekta, ki ga želimo nastaviti. `Set` ne vrne rezultata, če ni prišlo do napake (čeprav sam `Set` ne vrne rezultata bo strežniška aplikacija MAS sama zabeležila uspešno komunikacijo).

3.2.5.2.4. Get Time Difference

`Get Time Difference` funkcija prebere uro strežnika in jo nato odšteje od ure odjemalca. Da ta funkcija uspe, mora biti ura števca nastavljena, kar pomeni da moramo vsakih nekaj minut v strežniku nastavljati parameter, ki določa, če lahko nastavljamo uro. Če želimo lahko podamo tudi OBIS kodo objekta, ki ga želimo prebrati. Rezultati se morajo nahajati v posebni strukturi `SEP2 Register`.

3.2.5.3. SEP2 Register

`SEP2 Register` vsebuje le funkcijo `Read`. Ta funkcije deluje identično funkciji `Read Parameter` iz kategorije `SML Raw Data`. Edina razlika leži v predstavitvi podatkov. Pri tej funkciji so rezultati vrnjeni v obliki posebne strukture `SEP2 Register`.

3.2.5.4. SEP2 Event Profile

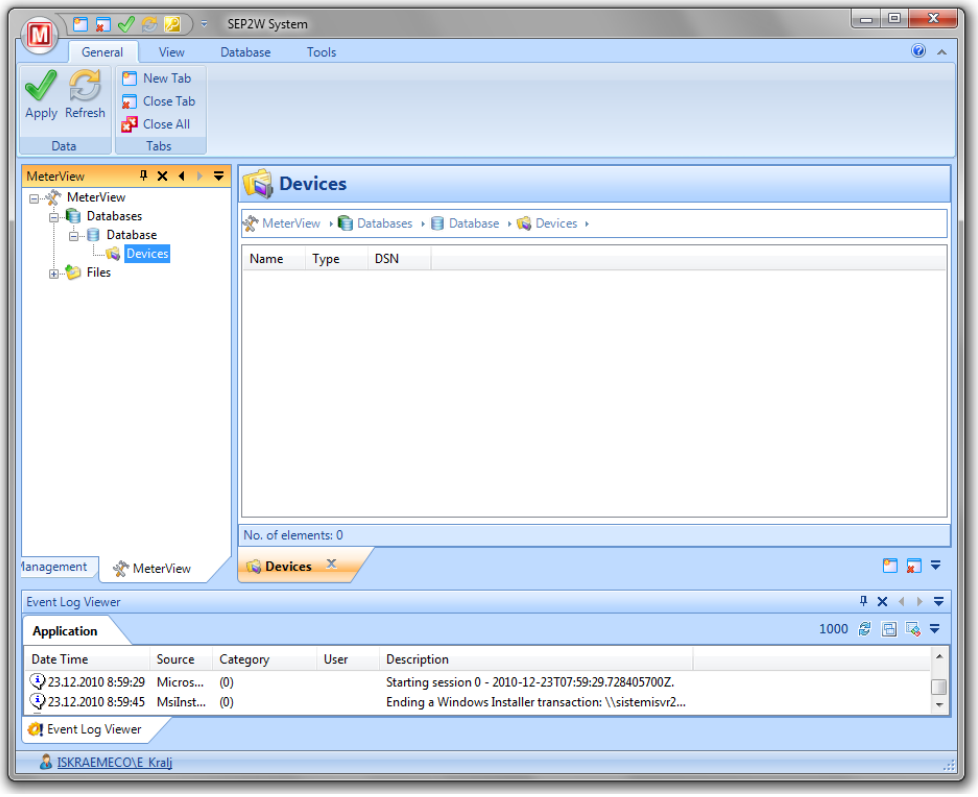
`SEP2 Event Profile` vsebuje le funkcijo `Read`. Ta funkcija deluje identično funkciji `Get Profile List` iz kategorije `SML Raw Data`. Edina razlika leži v predstavitvi podatkov. Pri tej funkciji so rezultati vrnjeni v obliki posebne strukture `SEP2 Event`.

4. Predstavitev in analiza delovanja knjižnice

Rezultat razvoja je knjižnica, ki omogoča strežniški aplikaciji MAS komunikacijo s pametnimi napravami, ki uporabljajo protokol SML. V tem poglavju bom predstavil primer komunikacije s števcem s pomočjo *SEP2W System*. Pogledali pa bomo tudi prostorsko in časovno zahtevnost knjižnice. Predstavil bom tudi nekaj drugih kandidatov za prenosne medije, ter kakšne prednosti oziroma slabosti bi prinesli.

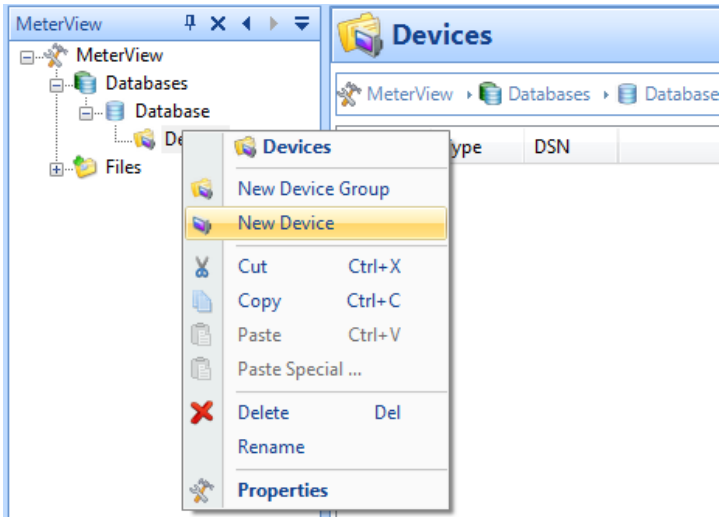
4.1. Predstavitev aplikacije

Za branje števca v *SEP2W System* moramo najprej ustvariti novo napravo s pravilnimi nastavitvami. Naša naprava bo priključena na računalnik neposredno preko serijskih vrat. Napravo najlažje naredimo v enemu od vtičnikov *SEP2W System*, ki ga vidimo na sliki 8. Vtičnik se imenuje *MeterView*.



Slika 8: *Meter View* je vtičnik za neposredno branje/pisanje števcev, ki med drugim vsebuje tudi čarovnika za ustvarjanje novih naprav.

V bazi podatkov izberemo želeno mapo. Z desnim klikom na mapo se nam odpre kontekstni meni iz slike 9. V meniju izberemo *New Device*.

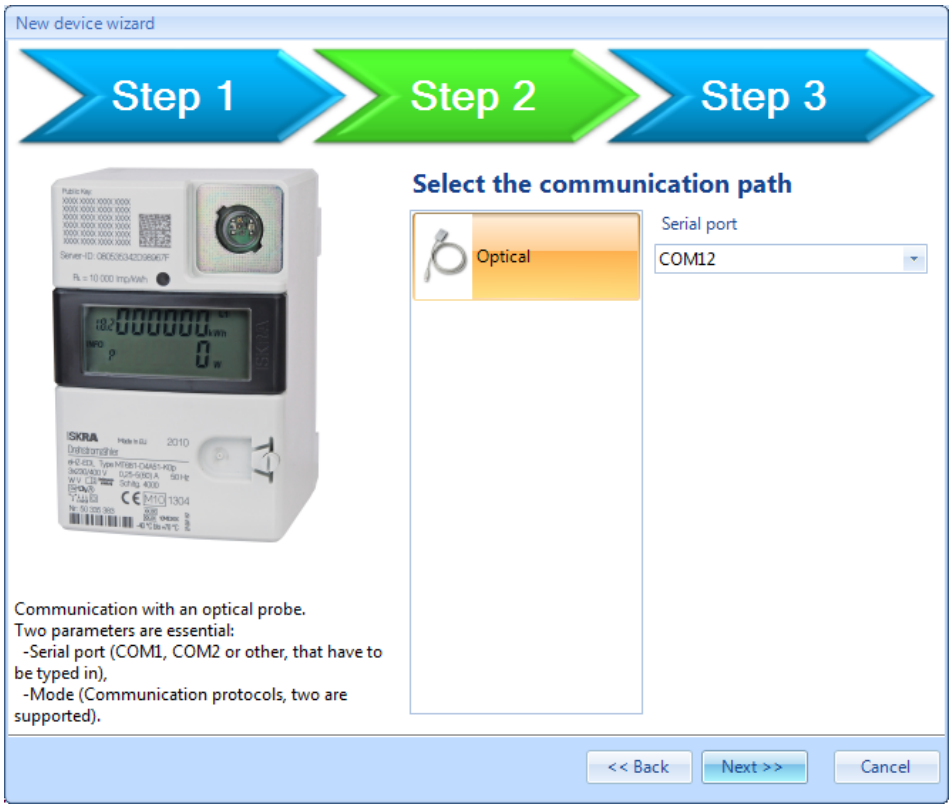


Slika 9: Kontekstni meni s pomočjo katerega lahko med drugim ustvarimo novo napravo.

Na sliki 10 je prikazan čarovnik, ki se nam odpre za pomoč pri kreaciji nove naprave. V čarovniku izberemo želeni tip števca (v našem primeru je to števec MT681). Vnesemo tudi geslo, ki ga bomo potrebovali za zapisovanje vrednosti. Nastavitev `Parameters map` pustimo na prevzeti vrednosti.

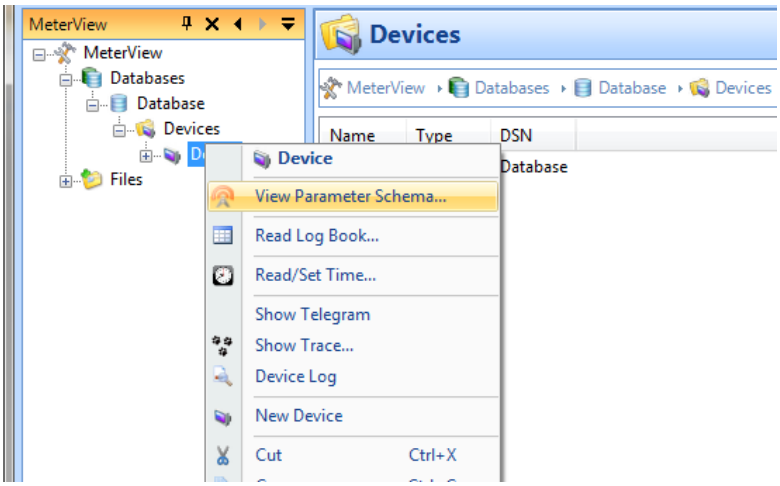
Slika 10: Prva stran čarovnika za vnos nove naprave. Na strani lahko izberemo tip naprave, vnesemo ime in geslo, določimo komunikacijski protokol (MT681 omogoča le protokol SML) in `Parameters map` (nastavitev določa lokacije – OBIS kode elementov znotraj števca).

Na sliki 11 vidimo naslednji korak pri kreaciji nove naprave. V tem oknu lahko izberemo komunikacijsko pot in njene nastavitve. Ker je trenutno edini način za branje števca preko optične sonde, moramo na tem oknu čarovnika le nastaviti serijska vrata. Za tem oknom sledi le potrditev podatkov.



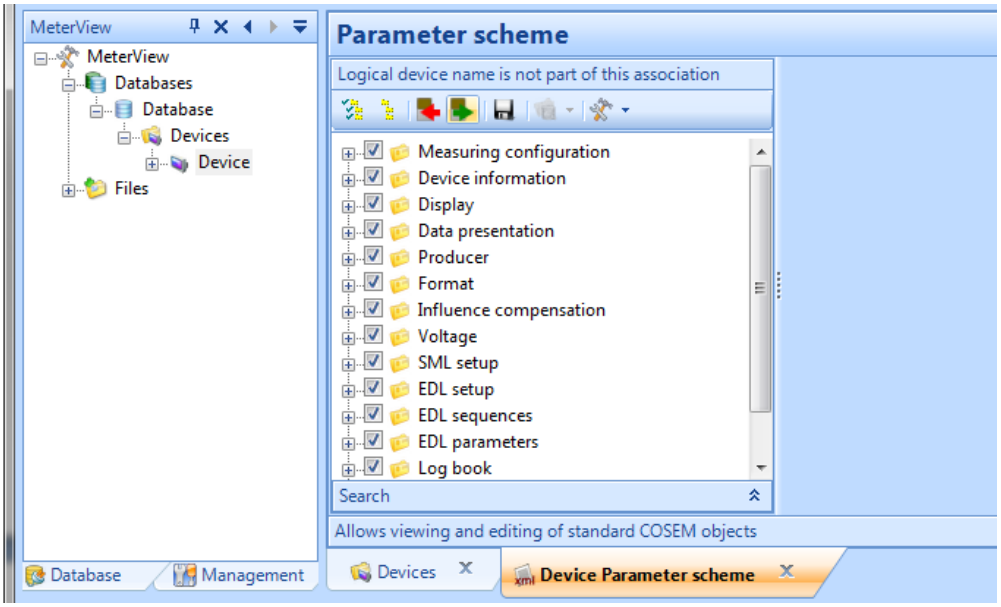
Slika 11: Nastavljanje komunikacijskih poti nove naprave.

Uspešno smo ustvarili novo napravo. Na napravi lahko sedaj z desnim klikom odpremo meni, ki vsebuje nekaj akcij ki jih lahko nad napravo opravimo. Primer menija za trenutno vrsto naprave lahko vidimo na sliki 12. Za prvi test izberemo `View Parameter Schema`.



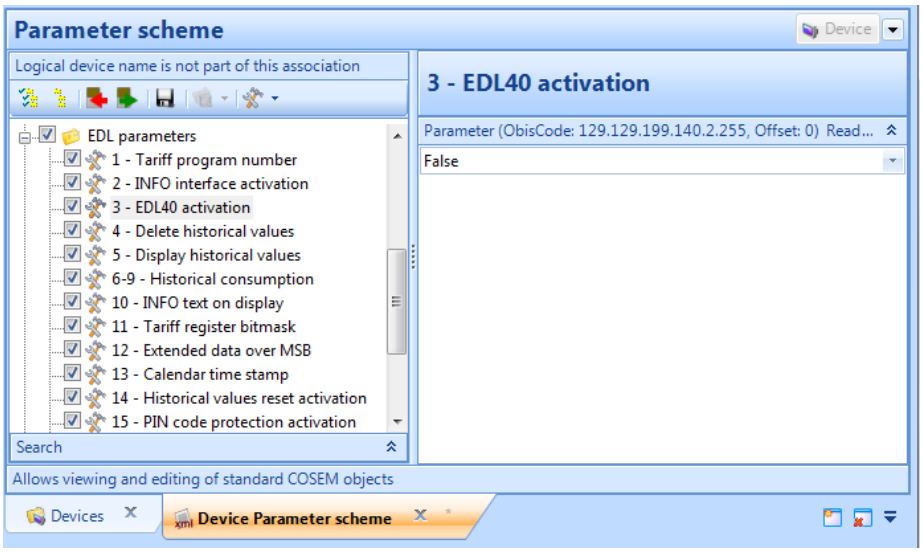
Slika 12: Različne akcije, ki jih lahko opravimo nad trenutno napravo.

Parametrska shema je neki fiksni dokument, ki vsebuje nekakšen zemljevid števca SML. Med kreacijo naprave smo imeli možnost izbire različnih parametrskih shem. Sedaj lahko začnemo z branjem in nastavljanjem števca. Najprej bomo prebrali celotno parametrsko shemo predstavljeno na sliki 13. Parametrsko shemo preberemo tako, da označimo polja, ki jih želimo prebrati (v našem primeru vsa) in pritisnemo zelen gumb nad parametrsko shemo. Vtičnik `MeterView` bo strežniški aplikaciji MAS poslal zahteve za branje izbranih parametrov, ta pa bo zahteve posredovala gonilniku protokola SML. Gonilnik bo za branje teh parametrov uporabil sporočilo `Get Proc Parameter Request`.



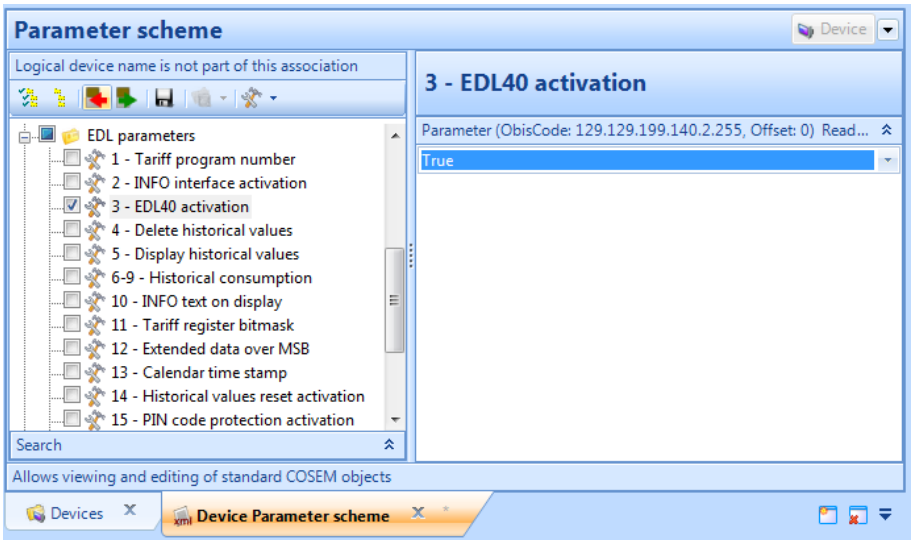
Slika 13: Parametrska shema števca.

Prebrane podatke lahko dobimo s klikom na posamezni element parametrske sheme. Na sliki 14 vidimo prikaz vrednosti polja EDL40, ki mora biti nastavljen na `True`, da števec dovoli nastavljati uro.



Slika 14: Primer prebranega Boolovega elementa.

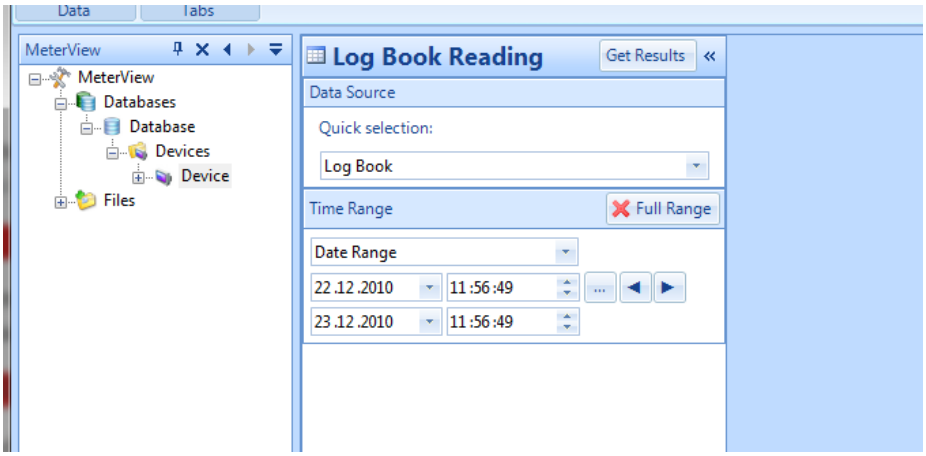
Podatke lahko tudi spremenimo in vpišemo v števec. V števec se vpisuje z rdečim gumbom nad parametrsko shemo. Vpisali bomo le polje EDL40 izbrano na sliki 15. Za vpisovanje atributa bo strežniška aplikacija MAS posredovala gonilniku protokola SML zahtevo po nastavljanju parametra. Gonilnik protokola SML bo za to nalogo uporabil sporočilo *`Set Proc Parameter Request`*.



Slika 15: Spremenjen parameter EDL40 pripravljen za vpis v števec.

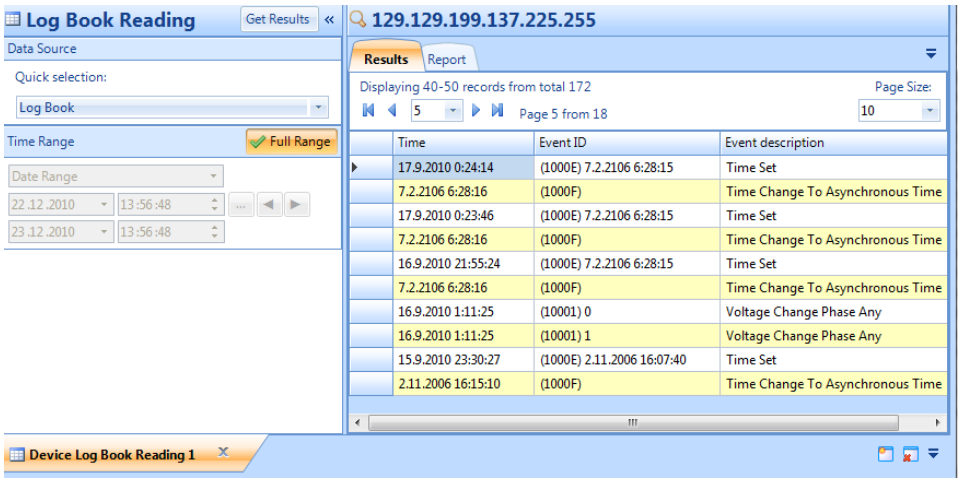
Branje in nastavljanje parametrov je bolj ali manj podobno po celi parametrski shemi. Parametrsko shemo zapremo ter ponovno kliknemo z desnim gumbom miške na napravo.

Tokrat v kontekstnem meniju izberemo možnost *Read Log Book*. Odpre se nam okno iz slike 16.



Slika 16: Okno za branje dogodkov naprave.

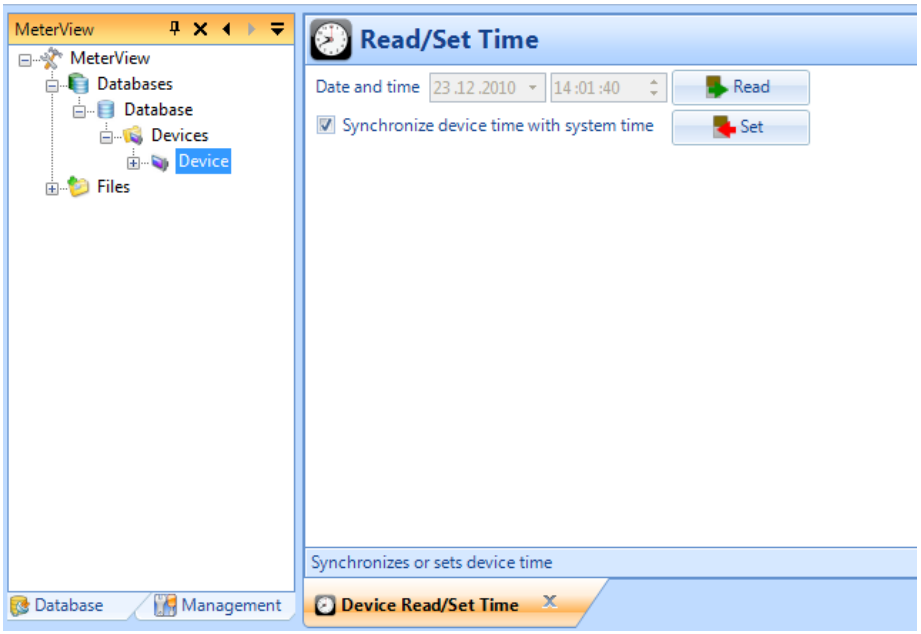
V tem oknu lahko beremo knjigo dogodkov naprave na določenem intervalu. Ker je knjiga dogodkov zelo obsežena, ta akcija vzame kar nekaj časa. S klikom na gumb *Get Results* sprožimo branje knjige dogodkov. Strežniška aplikacija MAS bo gonilniku protokola SML posredovala zahtevo za branje knjige dogodkov. Gonilnik bo za to nalogo uporabil sporočilo *Get Profile List Request*. Rezultati branja so vidni na sliki 17.



Slika 17: Rezultati branja knjige dogodkov.

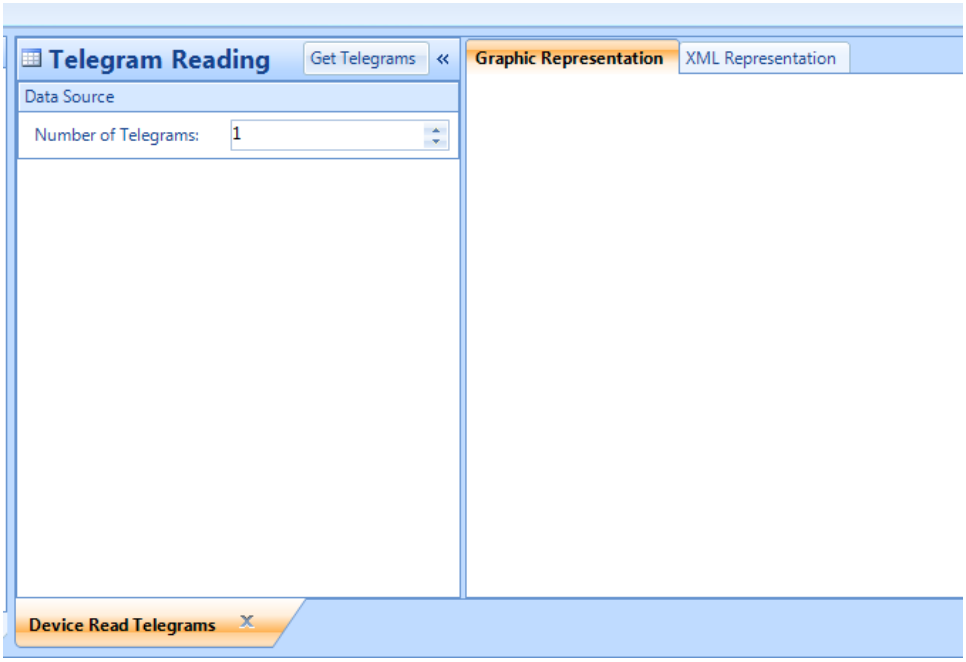
Ponovno odpremo kontekstni meni naprave in izberemo možnost *Read/Set Time*. V novo odprtem oknu, ki ga vidimo na sliki 18, imamo možnost izbire med ročnim nastavljanjem ure, sinhronizacijo in branjem ure. Gonilnik protokola SML za te naloge uporablja *Get Proc Parameter Request* in *Set Proc Parameter Request* v različnih kombinacijah.

Za sinhronizacijo ure bo na primer gonilnik najprej prebral uro, jo primerjal z uro računalnika, nastavil parameter, ki dovoljuje nastavljanje ure in na koncu še nastavil samo uro.



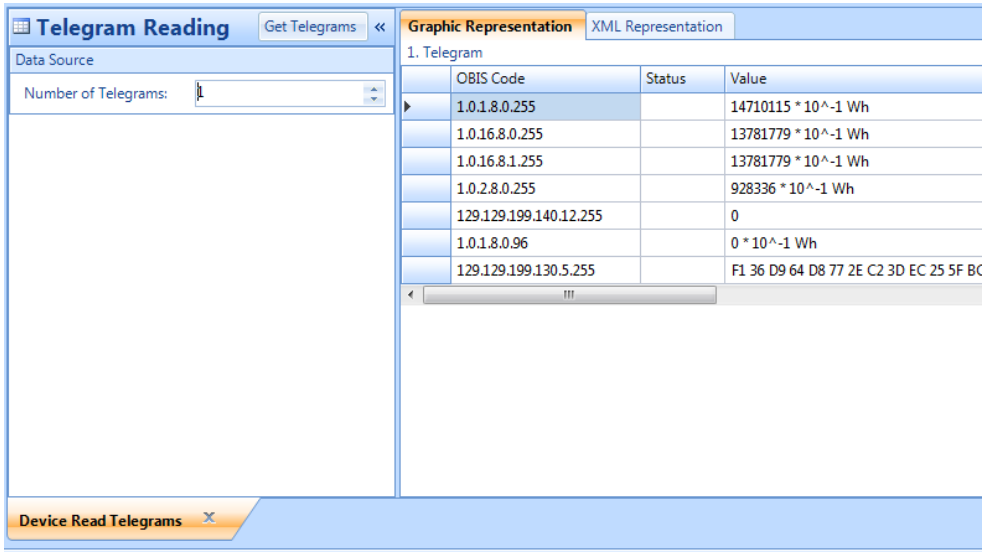
Slika 18: Okno za nastavljanje in branje ure.

Spet odpremo kontekstni meni naprave. Izberemo še možnost *`Show Telegram`*. Odpre se nam okno slike 19.



Slika 19: Okno za branj telegramov.

`Read Telegram` prebere podatke, ki jih števec samodejno vrača vsakih nekaj sekund. Te podatke števec vrača samo na eni optični sondi, ki je namenjena nadzorni napravi (koncentratorju). Podatke, ki jih števec vrača, je mogoče tudi spremeniti z zapisom spremenjenega seznama v števec (preko parametrske sheme). S pritiskom na gumb `Get Telegrams`, bo gonilnik protokola SML v bistvu le počakal na naslednjih nekaj telegramov. Telegrami so v obliki sporočila `Get List Response`. Na sliki 20 vidimo primer prebranega telegrama.



Slika 20: Prebrani telegram števca. Ker so telegrami oblike `Get List Response` en sam telegram vsebuje več različnih podatkov.

4.2. Analiza delovanja

V temu podpoglavju se bom spustil malce v samo delovanje knjižnice. Preveril bom časovno zahtevnost delovanja in porabo spomina. Pogledal bom tudi potencialne prenosne medije, ki bi lahko sledili in njihove prednosti ter slabosti.

4.2.1. Hitrost

Knjižnica trenutno ne trpi iz vidika hitrosti. Edina prava oviro pri hitrosti na katero bi lahko naleteli, zakrivi stalno vračanje podatkov. Trenutna verzija števca namreč na približno vsake tri sekunde na enih od svojih vrat odda odgovor brez predhodne zahteve. Števec bo prekinil z stalnim vračanjem podatkov za nekaj časa, če sprejme novo zahtevo. V tabeli 25 lahko vidimo nekaj časov različnih zahtev za branje/nastavljanje števca. Pri zbiranju podatkov sem vsako zahtevo poslal v svoji datoteki SML (za vsako zahtevo se je odposlalo sporočilo `Open`

Request, zahteva in sporočilo *Close Request*). Med časom priprave zahtev knjižnica potuje čez seznam zahtev ter sestavi ustrezna sporočila, nato, če nastavitve zahtevajo, počaka na konec vračanja podatkov in prebere javni ključ, če ta ni podan med nastavitvami. Čas pošiljanja zahtev je celoten čas pošiljanja sporočila SML od začetka do konca. Odvisen je od prenosnega medija in njegovih nastavitev (števec SML lahko trenutno komunicira le preko optične sonde). V primeru optične sonde priključene na serijska vrata igra veliko vlogo število bitov na sekundo (*baud rate*). Vsi testi so imeli nastavitev bitov na sekundo nastavljenih na 9600. Pri času prejemanja odgovorov gre za čas, ki ga števec porabi za razumevanje zahteve ter izgradnjo odgovora, čas priprave rezultatov pa je čas, ki ga knjižnica porabi, da rezultate vrne v primerni obliki glede na zahtevo (surovo sporočilo za metode *SML Raw Data* in strukture v obliki *MAS Data* za vse ostale SEP2 skupine). Če se v celici tabele nahaja več kot en čas to pomeni, da je akcija sestavljena iz več korakov ob različnem času. Primer tega je *SEP2 Clock – Synchronize*, ki najprej prebere uro, nato mora nastaviti parameter števca, ki omogoča nastavljanje ure in šele na koncu pride do dejanskega vpisovanja novega časa.

Akcije (zahteve)	Čas priprave zahtev [ms]	Čas pošiljanja zahtev [ms]	Čas prejemanja odgovorov [ms]	Čas priprave rezultatov [ms]
SML Raw Data - Read Parameter	1233	150	208	1
SML Raw Data - Write Parameter	1002	161	147	2
SML Raw Data - Get Profile List	973	170	87195	2
SML Raw Data - Read Parameter List	482	156	384	7
SML Raw Data - Read Telegram	5935	/	380	1
SEP2 Clock - Read	444	153	155	1
SEP2 Clock - Synchronize	638	151 + 162 + 181	178 + 146 + 147	1 + 2 + 1
SEP2 Clock - Set	1173	161 + 181	146 + 145	1 + 1

SEP2 Clock - Get Time Difference	871	150	169	6
SEP2 Register - Read	705	151	157	6
SEP2 Event Profile - Read	1273	170	89603	68

Tabela 25: Časi različnih zahtev pri branju/pisanju podatkov števca. Časi, ki so predstavljeni kot seštevek dveh časov, predstavljajo več različnih korakov istih akcij ob različnih časih. Nastavljanje ure na primer mora najprej nastaviti parameter, ki omogoča nastavljanje ure. To pomeni, da bomo strežniku poslali dve različni zahtevi.

Izmerjene hitrosti kažejo, da pri optični sondi problemov s hitrostjo vsaj za trenutne podatke ni. Pri drugačnem prenosnem mediju pa bi se lahko zapletlo. Če medij ne bi bil sposoben prenesti podatkov od odjemalca do strežnika v treh sekundah komunikacija ne bi bila mogoča. Seveda bi bilo v takem primeru mogoče povečati interval stalnega vračanja podatkov.

4.2.2. Pomnilnik

Knjižnica sama zahteva za današnje čase zelo majhno količino prostora na disku. Zahteva le 108 KB prostora. Ampak ker knjižnica sama ne more delovati moramo naložiti še strežniško aplikacijo MAS ter *SEP2W System* ali *MAS Client*. Strežniška aplikacija MAS potrebuje 2,082 MB prostora. V najbolj skromni obliki *SEP2W System* potrebuje 61 MB prostora, *MAS Client* pa le 2,19 MB prostora. Poleg tega vse aplikacije vključno s samo knjižnico potrebujejo *.NET Framework 4.0* za delovanje. Seveda aplikacija ne bo vedno tekla na najnovejših sistemih, vendar je za današnje čase velikost aplikacije še vedno zelo majhna.

Glede delovnega pomnilnika aplikacija nima jasno določene omejitve. Knjižnica do zaključka branja vse zahteve ter njihove odgovore hrani v delovnem pomnilniku. Takšno opravljanje s spominom lahko prinese na drugih sistemih probleme, saj cene delovnega pomnilnika v primerjavi s trdim diskom niso nizke in je zelo verjetno pričakovati sisteme z manjšimi količinami. Seveda je treba tudi v tem primeru v potrebno količino prišteti tudi vse potrebne aplikacije. Strežniška aplikacija MAS skupaj s knjižnico v mirovanju potrebuje 55,5 MB spomina, *SEP2W System* in *MAS Client* pa v mirovanju potrebujeta 107 MB in 54 MB spomina. V tabeli 26 lahko vidimo najvišjo mejo pomnilnika, ki ga knjižnica zahteva med izvajanjem določene zahteve v primerjavi z dejansko velikostjo sporočil, ki so vpleteni v akcijo. Največja zasedenost spomina lahko zelo niha in ni najbolj prepričljiva meritev, vendar bo vseeno dala idejo o razmerjih med surovimi pretečenimi podatki in porabo spomina. Za

zbiranje podatkov sem bral/nastavljal elemente, ki so najpogosteje v uporabi. Zahteve ter odgovore sem ločil s poševnico. Zahteve ter odgovori vneseni kot vsota predstavljajo več kot eno zahtevo/odgovor poslano ob različnih časih akcije.

Akcije (zahteve)	Velikost sporočila (zahteva / odgovor) [bajti]	Največja količina zasedenega spomina (100 branj) [MB]
SML Raw Data - Read Parameter	132 / 160	60,928
SML Raw Data - Write Parameter	144 / 128	60.932
SML Raw Data - Get Profile List	152 / 31352	61,79*
SML Raw Data - Read Parameter List	140 / 340	61,976
SML Raw Data - Read Telegram	- / 308	62,436
SEP2 Clock - Read	132 / 128	60,952
SEP2 Clock -Synchronize	132 + 144 + 160 / 128 + 128 + 128	60,864
SEP2 Clock - Set	144 + 160 / 128 + 128	62,040
SEP2 Clock - Get Time Difference	132 / 152	61,888
SEP2 Register -Read	132 / 140	60,912
SEP2 Event Profile - Read	152 / 31352	62,676*

Tabela 26: Največja velikost pomnilnika, ki ga knjižnica zasede pri izvajanju neke akcije (v mirovanju 56,5 MB) in dejanska velikost sporočil. Vrednosti največje porabe spomina, ki imajo zvezdico, niso bile brane s stotimi ponovitvami zaradi zelo dolgega časa branja.

4.2.3. Prenosni mediji

Trenutna verzija števca ima dva komunikacijska vmesnika. Na obeh vmesnikih je mogoče priključiti optično sondo. Zadnji vmesnik, ki ga je pri vgrajenem števcu nemogoče doseči, je namenjen komunikaciji števca s fiksnimi kontrolnimi napravami (koncentratorji). Ta vmesnik vsakih nekaj sekund vrača podatke, katerih vsebino je mogoče nastaviti, ki jih kontrolne naprave uporabljajo za nadzor nad števci. Sprednji vmesnik je namenjen neposrednemu branju števca na mestu priklopa. Na prednjem vmesniku števec ne vrača podatkov samodejno po nekem časovnem intervalu. Mogoče je, da bodo vrsto katerega od teh dveh vmesnikov kasneje zamenjali. Med primerne prenosne medije spadajo DLC, Network in GSM. V tabeli 27 lahko vidimo primerjave med različnimi prenosnimi mediji, preko katerih bo mogoče nekoč mogoče brati števec SML.

Prenosni medij	Opis	Prednosti	Slabosti
Optična sonda	Omogoča neposredna povezava s števcem. Do nadaljnjega je to edini način priklopa na števec.	Omogoča hiter prenos podatkov.	Za branje oziroma priklop števcev preko optičnih sond bo potrebno napeljati veliko število novih kablov.
DLC	Omogoča prenos podatkov kar preko električne napeljave.	Žica po kateri merimo porabo igra tudi vlogo samega prenosnega medija.	Omogoča zelo počasen prenos in ima branje omejeno na naprave priključene na isto transformacijsko postajo.
Network	Predstavlja prenos preko TCP/IP protokola po standardnem UTP kablu.	Tak način prenosa podatkov je že zelo razširjen, je prisoten v skoraj vsakem gospodinjstvu in omogoča relativno hiter prenos podatkov.	Še vseeno pa takšen prenos podatkov ni prisoten v čisto vsakem gospodinjstvu in zasede pasovne širine za druge namene (zanemarljivo malo).

GSM	Prenos podatkov po zraku, ki ga uporabljajo mobilni telefoni.	Zračni prenos ne zahteva vgradnje nobenih novih kablov.	Prenos signalov po zraku je lahko zelo problematično v goratih območjih. Velik problem je tudi strošek in dejstvo, da še dodatno obremeni GSM omrežje.

Tabela 27: Primerjava prenosnih medijev.

5. Zaključek

Največje probleme pri razvoju gonilnika protokola SML sem imel zaradi drugačnega delovanja v primerjavi z ostalimi gonilniki protokolov razvitih v podjetju Iskraemeco d.d.. Protokol SML omogoča pošiljanje večjega števila zahtev v eni sami datoteki SML. Strežniška aplikacija pričakuje, da se bo za vsako podano zahtevo odposlal blok podatkov, na katerega bo kmalu za tem prišel odgovor. Zaradi pošiljanja več zahtev v eni sami datoteki SML, mora gonilnik protokola SML vsebovati neko svoje pomnjenje, kjer shrani odgovore na zahteve, ki smo jih že odposlali skupaj s trenutno datoteko SML. Metoda *OptimizeOperationsImpl* pa se s protokolom SML zelo dobro ujame. Skupen pogled nad vsemi zahtevami strežniške enote MAS omogoča izgradnjo datotek SML brez večjih problemov. Če bi zahteve prišle samo preko metode *ExecuteInvokeImpl*, ne bi bilo nemogoče izkoristiti možnost protokola SML za pošiljanje večjega števila zahtev v eni datoteki, saj bi bilo potrebno na vsako zahtevo takoj odgovoriti.

Ko sem pričel z razvojem aplikacije, nisem pogosto uporabljal vmesnikov, razredov ter dedovanja. Menil sem, da so programi veliko bolj preprosti in pregledni, če je večina kode v istem razredu. Zato so bili edini razredi v gonilniku protokola SML *SmlDriver*, generični razred *SmlData*, ter osnovni tipi izpeljani iz tipa *SmlData*. Osnovne tipe sem razbil na več razredov le zato, ker sem delal po vzoru razreda *MasData*. Kasneje sem iz osnovnih tipov izpeljal še razširjene tipe. Na koncu sem razdelil še vloge razreda *SmlDriver* na razrede *SmlMessageCreator*, *SmlFile* in *SmlFileCollection*. Z razvojem gonilnika protokola SML se je tudi spreminjal moj pogled na objektno programiranje. Čeprav je bila implementacija izpeljanih tipov protokola SML popolnoma moja odločitev, je ideja za to še vedno izvirala iz implementacije razreda *MasData*. Ideja za razbitje vloge razreda *SmlDriver* na več razredov pa je bila lastna. Sedaj vidim, da ravno razbitje kode na več razredov naredi kodo bolj pregledno. Z lepo ločenimi vlogami različnih razredov je spreminjanje kode veliko bolj preprosto, kar se je lepo pokazalo pri dodajanju preverjanja podpisa števca. Z razbitimi razredi je bila implementacija te nove funkcionalnosti rešena v hipu.

Sam protokol SML mi je bil v večini všeč. Ima probleme s tem, da v nekaterih delih ni dovolj prilagodljiv. Ta problem je najbolj opazen pri branju dogodkov, ki je rešeno tako, da se en dogodek zapiše v več struktur *SML Period Entry* enega sporočila *SML Get Profile List Response*. Druga stvar, ki se mi zdi nepotrebna je, da so osnovni tipi razbiti na več enakih tipov. V protokolu SML tipa *Unsigned8* ter *Unsigned16* nista enaka. Menim, da bi bilo veliko bolj preprosto, če bi se nepredznačena števila označilo le z enim nazivom (na primer *Unsigned)*, ki je spremenljive dolžine, kot tipa *Octet String* ter *ListOf*. S takim poimenovanjem tipov tudi ne bi bilo potrebno imeti izjem, kot v tipu *SML Message*

Body kjer je izbirno polje v specifikaciji definirano kot *Unsigned32*, v samem strežniku protokola SML pa kot *Unsigned16*.

Menim, da bi bilo zelo zanimivo spisati gonilnik za protokol SML, ki bi deloval malo bolj generično, kjer edini odjemalec ne bi bila le strežniška aplikacija MAS. Veliko razredov v trenutni knjižnici bi bilo mogoče razbiti na še več logičnih enot ter mogoče celo implementirati osnovne tipe brez določene dolžine in se vseeno držati specifikacije protokola. Ker je protokol SML sestavljen tako, da informacija o tipih pride pred vsebino samo, bi bilo zelo zanimivo videti aplikacijo, ki bi izkoristila še to prednost protokola SML ter omogočala komunikacijo brez vsakršnega oziroma najmanjšega možnega pomnjenja prejetih podatkov.

Viri

[1] Opis električnih števcev. Dostopno na:

http://en.wikipedia.org/wiki/Electrical_meter

[2] Opis pametnih števcev. Dostopno na:

http://en.wikipedia.org/wiki/Smart_meter

[3] Opis pametne mreže. Dostopno na:

http://en.wikipedia.org/wiki/Smart_grid

[4] Slika pametne mreže. Dostopna na:

http://www.hawaiisenergyfuture.com/Articles/Smart_Grid.html

[5] (2008) *SML, Smart Message Language 1.02*. Dostopno na:

http://www.sym2.de/i_sml-spezi.html (Dokument ni več dostopen. Dostopna je le verzija 1.03, ki pa je v nemškem jeziku)

[6] (2010) *SML, Smart Message Language 1.04*. Dostopno na:

http://www.sym2.de/i_sml-spezi.html (Specifikacija protokola SML 1.04 še ni prosto dostopna. Dostopna je le verzija 1.03, ki pa je v nemškem jeziku)

[7] Opis aplikacije *Microsoft Visual Studio*. Dostopno na:

http://en.wikipedia.org/wiki/Microsoft_Visual_Studio