Contor inteligent bazat pe evaluarea semnăturii energetice

Raport ştiinţific şi tehnic - Etapa II
Proiectarea componentelor hardware şi software

Cuprins

1. Introducere

Scopul documentului

Scopul acestui document este de a detalia proiectul sistemului SigMET prin cele două componente principale:

- Componenta software
- Componenta hardware

Documentul se adresează liderilor şi inginerilor din echipele de lucru şi de asemenea reprezentanților autorității contractante.

Rezumatul etapei

Obiectivul central al proiectului SigMET îl reprezintă realizarea modelului experimental al unui sistem de tip contor inteligent, capabil să furnizeze informații privitoare la consumul de energie electrică al fiecărei clase de consumatori întâlniți în mod curent la utilizatorii rezidențiali. Pentru atingerea acestui obiectiv, în cadrul etapei curente au fost utilizate rezultatele obținute în etapa I, în care a fost definită arhitectura generală şi funcționalitățile sistemului. Au fost detaliate proiectele principalelor componente hardware şi software.

Diseminarea rezultatelor obținute a fost realizată prin intermediul a 9 articole ştiințifice dintre care unul aflat în stadiul de evaluare. A fost actualizată pagina web a proiectului, care poate fi accesată la adresa: www.ee.tuiasi.ro/~SigMET

2. Proiectarea componentei software a contorului cu algoritm de recunoaştere a consumatorului

In vederea compatibilizării arhitecturii **SigMET** cu sistemul de distribuție al energiei la nivel de locuință individuală s-a stabilit proiectarea componente software cu următoarele date de intrare:

- Semnal de tensiune U (forma de undă);
- Semnal de curent I_1 corespondent circuitului de iluminare al locuinței;
- Semnal de curent I_2 corespondent circuitului de alimentare prin prize.

Arhitectura generală de proiectare utilizează un set de rutine prin care se monitorizează continuu valoarea puterii (pentru întreaga locuință) active P_1 si P_2 si reactive Q_1 și Q_2 (corespunzătoare curenților circuitelor de iluminat și prize) în scopul identificării apariției evenimentelor specifice conectării sau deconectării unui consumator. Se consideră apariția unui eveniment dacă variația unei puteri depășește o valoare impusă de prag (T).

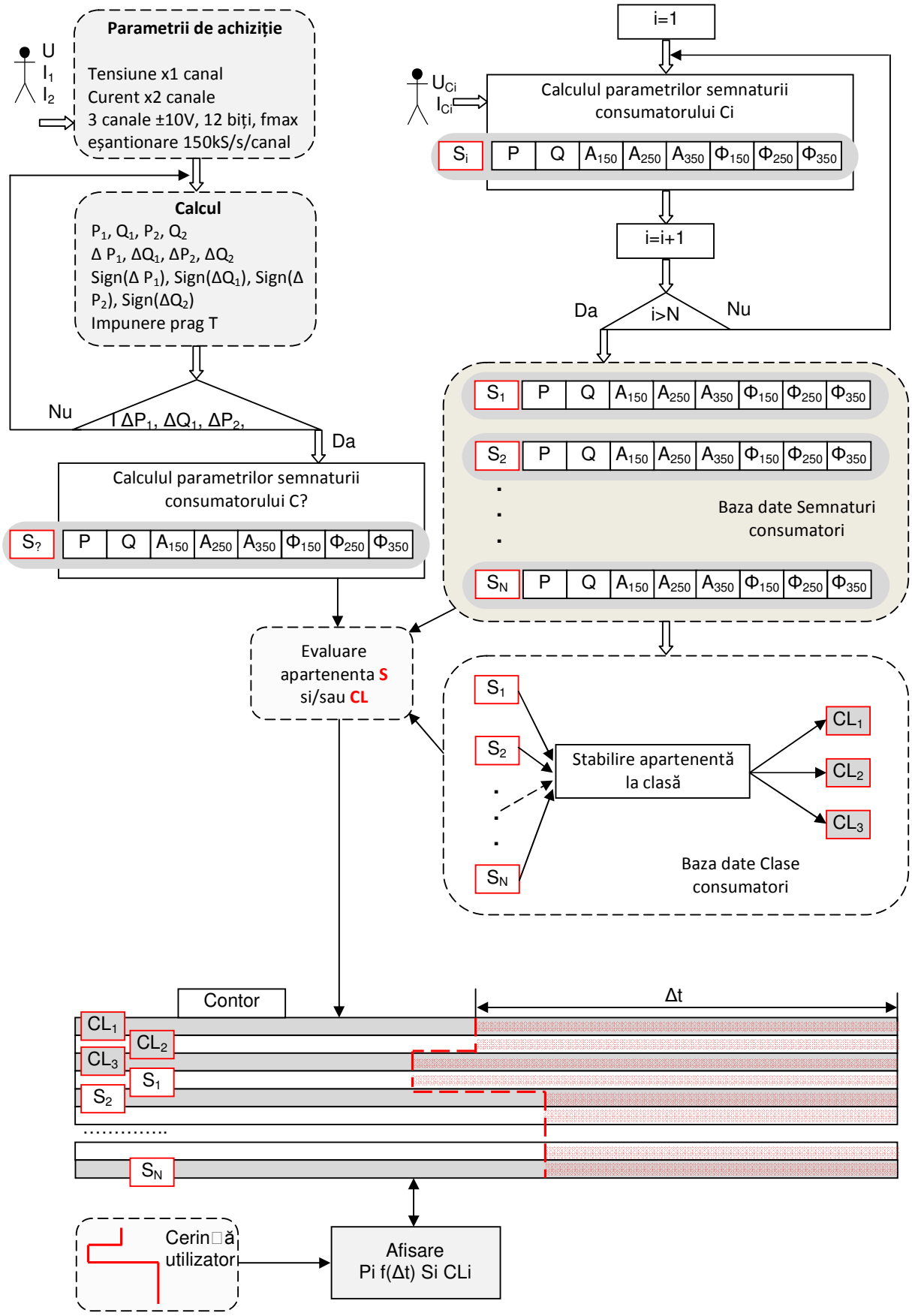
Apariția unui eveniment impune determinarea formei de undă a curentului și tensiunii consumatorului care a determinat evenimentul, pe baza diferenței formelor de undă considerate la momentele de dinainte și de după cel al producerii evenimentului. Pe baza formelor de undă de curent și tensiune se calculează (S) - ***parametrii semnăturii consumatorului*** supus identificării (C?).

Parametrii semnăturii unui consumator sunt următorii:

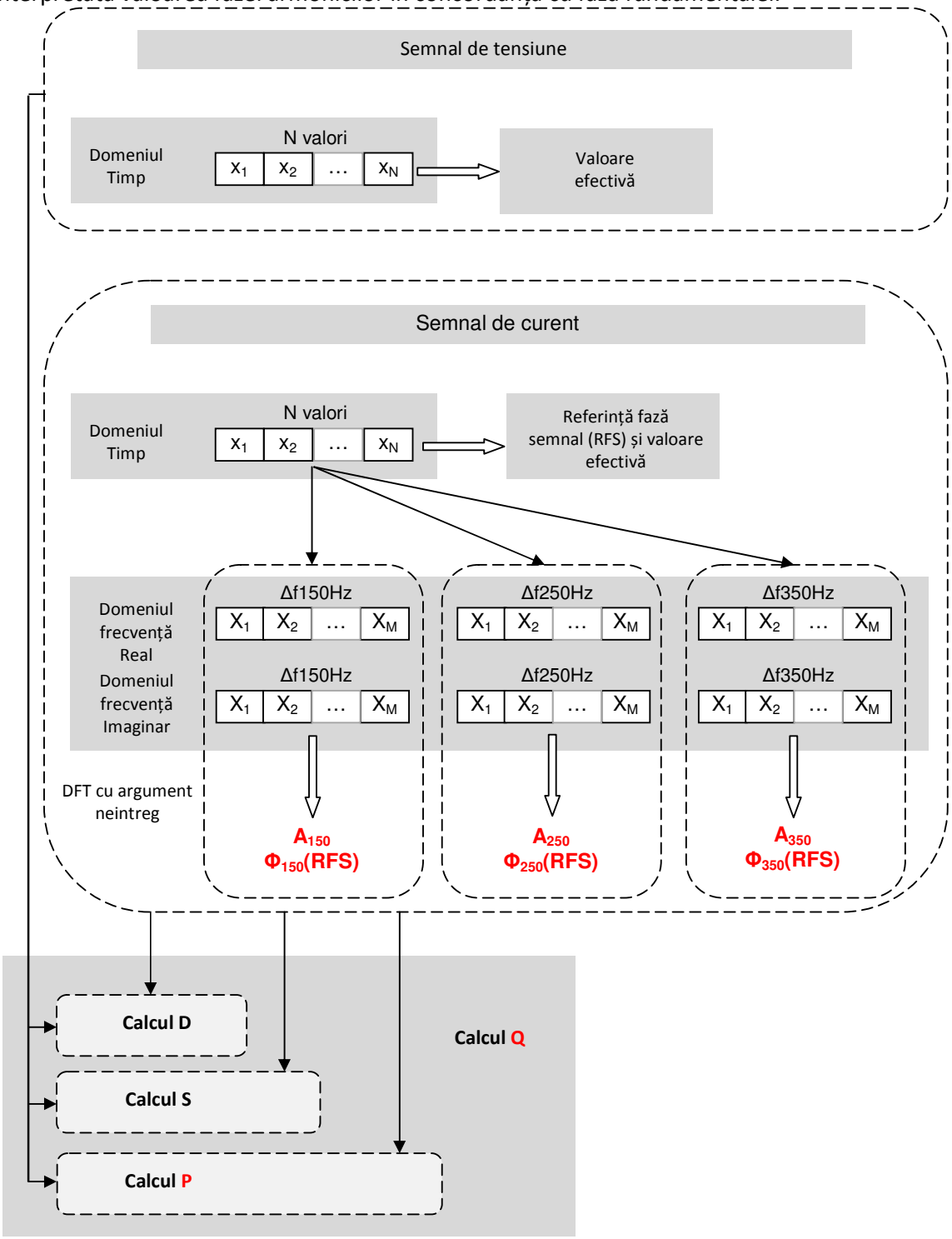
- P puterea activa, calculată ca fiind media produsului dintre semnalul de tensiune și de curent;
- Q puterea reactivă calculată pe baza $Q^2=S^2-P^2-D^2$, in care S este puterea aparentă calculată ca produs al valorii efective de curent și tensiune;
- A_{150} amplitudinea armonicii de 150Hz;
- A_{250} amplitudinea armonicii de 250Hz;
- A_{350} amplitudinea armonicii de 350Hz;
- Φ_{150} faza armonicii de 150Hz față de fundamentală, cu reținerea fazei fundamentalei ca referință;
- Φ_{250} faza armonicii de 250Hz față de fundamentală, cu reținerea fazei fundamentalei ca referință;
- Φ_{350} faza armonicii de 350Hz față de fundamentală, cu reținerea fazei fundamentalei ca referință.

Parametrii calculați ai consumatorului (C?) supus identificării sunt comparați cu parametrii consumatorilor stocați în prealabil in baza de date a semnăturilor consumatorilor. Evaluarea identificării numelui sau clasei consumatorului supus identificării se realizează pe baza Scorului de potrivire SCor. Acesta se calculează pe baza ponderii influenței abaterii fiecărui parametru al semnăturii consumatorului supus identificării față de parametrii consumatorilor aflați în baza de date. **SCor** = $C_P(P_i-P_?)/P_i + C_Q(Q_i-Q_?)/Q_i + C_{A150}(A_{150\ i} - A_{150\ ?})/A_{150\ i} + C_{A250}(A_{250\ i} - A_{250\ ?})/A_{250\ i} + C_{A350}(A_{350\ i} - A_{350?})/A_{350\ i} + C_{\Phi150}(\Phi_{150\ i} - \Phi_{150\ ?})/\Phi_{150\ i} + C_{\Phi250}(\Phi_{250\ i} - \Phi_{250\ ?})/\Phi_{250\ i} + C_{\Phi350}(\Phi_{350\ i} - \Phi_{350\ ?})/\Phi_{350\ i}$. Coeficienții de ponderare C au valori cuprinse intre 0 și 1.

Se acceptă ca apartenență determinată, situația în care SCor este mai mic decât sensibilitatea de detecție impusă.


Parametrii de achiziție
U
I1
I2
Tensiune x1 canal
Curent x2 canale
3 canale ±10V, 12 biți, fmax eșantionare 150kS/s/canal
Calcul
P1, Q1, P2, Q2
Δ P1, ΔQ1, ΔP2, ΔQ2
Sign(Δ P1), Sign(ΔQ1), Sign(Δ P2), Sign(ΔQ2)
Impunere prag T
Nu
Da
Calculul parametrilor semnaturii consumatorului C?
S?
P Q A150 A250 A350 Φ150 Φ250 Φ350
i=1
UCi
ICi
Calculul parametrilor semnaturii consumatorului Ci
Si
i=i+1
Da
i>N
Nu
S1
S2
SN
Baza date Semnaturi consumatori
Evaluare apartenenta S si/sau CL
Stabilire apartenentă la clasă
CL1
CL2
CL3
Baza date Clase consumatori
Contor
Δt
Cerință utilizator
Afisare Pi f(Δt) Si CLi

Arhitectura de calcul a parametrilor semnăturii consumatorilor se bazează pe utilizarea Transformatei Fourier Discretă aplicată centrat doar pe frecvențele de interes 150Hz, 250Hz și 350Hz. Utilizarea valorilor neîntregi pentru argumentul funcției permite creșterea rezoluției bins-urilor din domeniul frecvență și implicit înlăturarea efectului picket fence. In această situați poate fi interpretată valoarea fazei armonicilor în concordanță cu faza fundamentalei.



Calculul parametrilor semnăturii consumatorilor se aplică atât in etapa de constituire a bazei de date când se introduc individual în circuit consumatorii vizați dar și în etapa de monitorizare a consumului, după detecția unui eveniment.

Calculul fazei armonicilor, în etapa constituirii bazei de date, se realizează cu reținerea fazei in care se afla fundamentala în momentul investigării (RFS). Acest parametru deși nu caracterizează semnătura consumatorului, se stochează ca aparținând acestuia, fiind necesar în calculul parametrilor consumatorului din etapa de monitorizare, când este necesară refacerea condițiilor din etapa de constituire a bazei de date, referitoare la faza fundamentalei.

Datele de ieșire ale blocului de evaluare a apartenenței sunt transmise către contor în formatul (S_i, CL_i,P_i),reprezentând variația puterii la apariția unui eveniment, pentru un consumator sau o clasă identificată.

Cerințele utilizatorilor sunt transmise către blocul contor sub formă matricială, compatibilă cu structura de stocare a contorului, cuprinzând pe linii, consumatorii (S_1 … S_N,) sau clasele de apartenență (CL_1, CL_2, CL_3) iar pe coloane perioada de timp pentru care se solicită calcularea energiei active consumată.

3. Proiectarea componentei software de stocare a datelor de măsurare.

Sunt patru tipuri de operații pe care utilizatorul le poate efectua prin utilizarea bazei de date:

1. **Definirea datelor**: definirea unor noi structuri de date pentru o bază de date, eliminând structurilor de date sau modificarea acestora.
2. **Actualizarea datelor**: introducerea, modificarea și ștergerea datelor.
3. **Citirea datelor**: obținerea de informații brute sau rapoarte complexe obținute prin prelucrarea datelor primare.
4. **Administrare**: înregistrarea și monitorizarea utilizatorilor, aplicarea regulilor de securitate a datelor, monitorizarea performanțelor, menținerea integrității datelor, recuperarea informațiilor în caz de avarie.

3.1. Definirea scopului bazei de date

In urma analizării structurilor de date cu care operează sistemul SigMet putem nota următoarele:

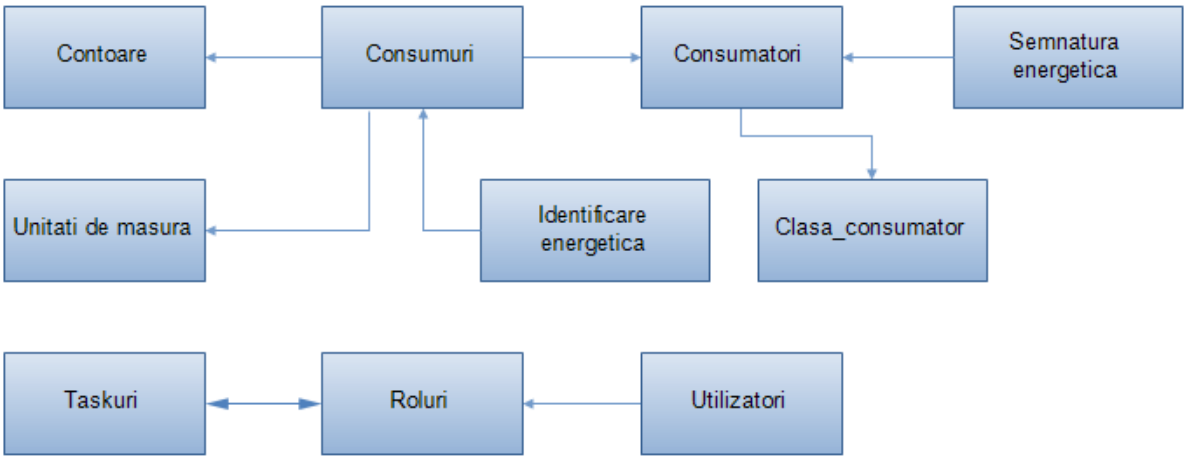
- Sistemul poate funcționa cu diverse tipuri de contoare, pentru fiecare contor fiind necesar a se salva următoarele informații: Id, serie, cod, descriere, an fabricatie, marcă, clasă de precizie, putere nominală.
- Fiecare contor în urma analizei curbei de semnal trimite periodic valorile de consum pentru fiecare tip de consumator predefinit.
- Sistemul trebuie sa furnizeze rapoarte despre consumurile inregistrate de contor pentru fiecare tip de consumator, putere medie, putere de varf, putere minimă, grafic orar, săptămânal, lunar.
- Accesul la resursele sistemului trebuie acordat doar pe bază de cont şi parolă, fiecare cont având drepturi se administrare specifice.

3.2. Diagrama entitate-asociere

Toate informatiile culese în timpul analizei se transpun într-o diagrama de tip "entitate-asociere" (EA) în care se modelează grafic structura bazei de date.
Acest model lucrează cu două concepte:

- "Entitate" - este un concept ce modelează clasele de obiecte pentru care se colectează informațiile: contor, marcă, utilizator, consum, etc. O entitate conține mai multe atribute, atributul fiind o informație atomică ce se dorește a fi salvată în baza de date relativ la acea entitate. De exemplu, pentru un contor se pot defini următoarele atribute: cod, tip, descriere, marcă, putere nominală, et,
- "Asociere" - modelează relațiile, interdependențele dintre entități. De exemplu, între contoare şi consumuri se poate stabili asocierea "consum înregistrat de" care stabilește modul de asociere a consumurilor la contoarele aferente.

Figura următoare prezintă diagrama EA pentru baza de date SigMET:



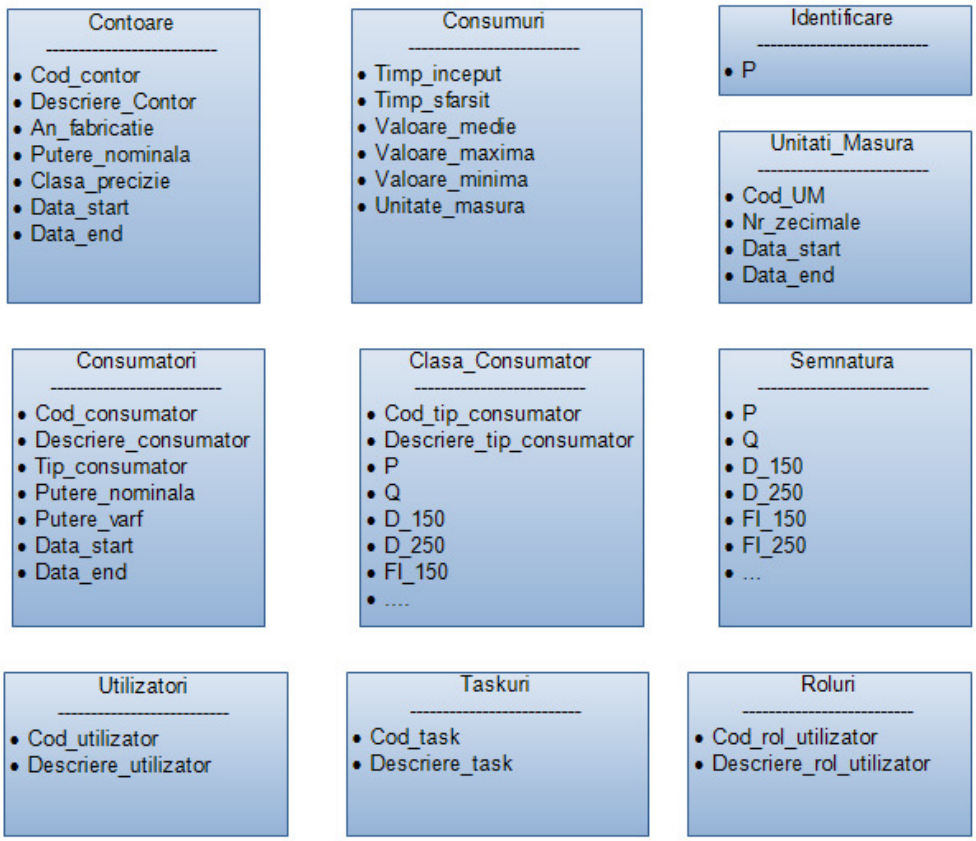
Pentru sistemul SigMET au fost identificate următoarele entități de informație:

- Contoare: stochează datele specifice contoarelor din sistem
- Consumuri: stochează consumurile periodice ale fiecarui contor in parte si fiecare consumator in parte
- Unitati_de_masura: gestioneaza unitățile de măsură cu care se lucrează in sistem
- Consumatori: gestionează consumatorii din sistem
- Clasa_consumator: consumatorii sunt impartiti pe clase, fiecare tip avand anumite caracteristici energetice
- Semnatura energetica: reprezintă amprenta energetică introdusă în sistem la conectarea unui consumator
- Identificare energetica: este rezultatul identificării energetice ce se calculează în urma apariției unui eveniment dat de conectarea unui consumator
- Utilizatori: gestioneaza utilizatorii care pot accesa sistemul

- Roluri: utilizatorii sunt impartiti pe diverse roluri, fiecare rol cu anumite drepturi
- Taskuri: lista de activitati ce pot fi operate asupra sistemului de catre diversi utilizatori

3.3. Conversia elementelor informaționale în coloane de tabele

In baza de date, informația este stocată sub formă de tabele, pentru fiecare entitate din modelul EA se creează un tabel având câte o coloană pentru fiecare atribut al entității. Conform observațiilor de mai sus, ar rezulta următoarea structură de tabele necesară pentru stocarea informațiilor specifice:

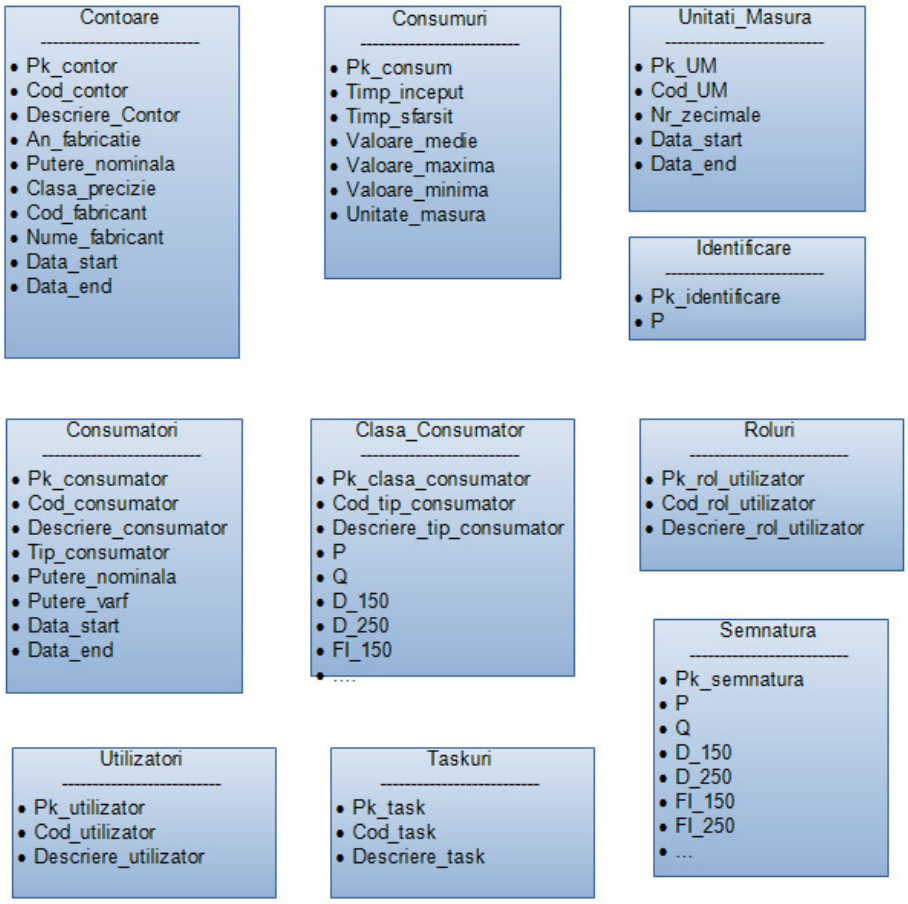


Reguli care trebuie respectate în crearea coloanelor de tabele:

- Informația dintr-o coloană trebuie să fie atomică, în sensul că nu mai poate fi despărțită în alte elemente componente
- Nu se pun coloane pentru valori calculate: de exemplu, nu este indicat să fie o coloană cu consumul specific. Aceasta se calculează printr-un apel de funcție în momentul când este necesar, în caz contrar, ar trebui recalculată la orice modificare a tabelelor, ceea ce poate constitui o sursă de erori (dacă se uită de exemplu să se recalculeze la un moment dat, sau se termină calculul printr-o eroare).

3.4. Specificarea cheilor primare

Orice tabel trebuie să conțină un identificator unic al liniilor cu informații. Numai în acest fel se poate regăsi în mod unic și corect o anumită linie conform unor criterii de căutare. Acest identificator unic se numește cheie primară și poate fi formată din una sau mai multe coloane a căror combinație este unică.
De exemplu, pentru tabela Contoare am putea alege ca identificator unic seria contorului. Chiar dacă am fi tentați să folosim ca cheie primară seria unui contor, această opțiune nu este bună, deoarece oricând se pot găsi două contoare de la firme diferite care sa aibă aceeaşi serie.
Soluția cea mai sigură din acest punct de vedere este să adăugăm câte o coloană distinctă la fiecare tabel care să conțină valori numerice unice pentru fiecare linie din tabel. Cunoscând acel număr, putem identifica în mod unic linia din tabela de interes. Bazele de date pun la dispoziție mecanisme de generare a numerelor unice, astfel încât, aceste coloane să se completeze automat de către baza de date la inserarea unei linii noi în tabelă.
Ne conformăm acestor reguli de identificare a liniilor din tabele și adăugăm la toate tabelele câte o coloană numită "pk_...", unde punctele din denumire vor fi înlocuite cu numele tabelului în cauză.
In acest fel, schema logică a tabelelor bazei de date evoluează spre următoarea configurație:



3.5. Crearea relațiilor dintre tabele

Toate asocierile definite în diagrama EA trebuie să se regăsească în structura tabelelor, deci următorul pas în crearea bazei de date îl reprezintă definirea relațiilor dintre tabele. Relațiile de legătură între tabele se creează prin folosirea de coloane comune, de joncțiune. Cele două tabele asociate au fiecare câte o coloană ce conțin date identice. Plecând de la valoarea dată într-un tabel, se face joncțiunea cu cel de-al doilea tabel prin selectarea liniilor care au aceeași valoare în coloana comună. De exemplu, în tabela "Consumuri" se introduce o coloană numită "pk_contor" ca va conține identificatorul contorului de care aparține acel consum. Având un consum dat, se regăsește valoarea "pk_contor" pentru acel consum și cu acea valoare se merge în tabela "Contoare" unde vom găsi mai multe informații specifice contorului care a înregistrat acel consum.
De reținut că asocierile pot fi de mai multe feluri:

- Unu la unu: o instanță din prima entitate este asociată la o singură instanță din a doua entitate și reciproc
- Unu la mai multi: o linie dintr-o entitate este asociată cu mai multe linii din cealaltă entitate. Exemplu: un contor este asociat cu mai multe linii din tabela de consumuri (acele consumuri se adauga periodic pentru fiecare contor din sistem).
- Mai mulți la mai mulți: un rol poate fi asociat cu mai multe taskuri (activități), dar şi invers, o activitate poate fi asociată la mai multe roluri.

Convenție: in diagramele EA ramurile asocierilor de tip "unu" vor fi reprezentate sub formă de săgeată.

Crearea relațiilor de tipul unu-la-mai-mulți

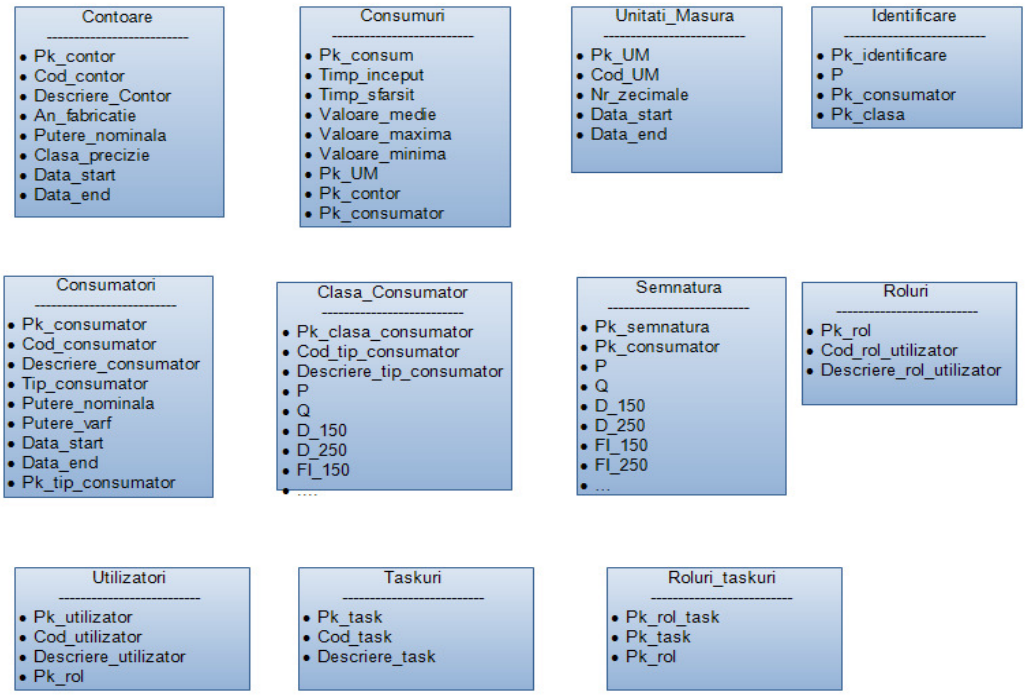
Este relația întâlnită cel mai des, exprimă în general apartenența unei entități la o clasificare mai largă. Din analiza diagramei EA, observăm că trebuie create următoarele relații:

- consumuri-contoare: fiecare consum apartine de un contor, la fiecare contor se asociaza mai multe consumuri
- consumuri-consumatori: fiecare consum apartine de un consumator, la fiecare consumator se asociaza mai multe consumuri
- consumuri-unități de masură
- consumator-tip consumator
- utilizatori-roluri: un rol poate include mai multi utilizatori

Crearea relațiilor de tipul mai-mulți-la-mai-mulți

Relația între taskuri si roluri este mai complexă deoarece un rol este asociat cu mai multe taskuri si un task poate fi acordat mai multor roluri. Aceasta relație complexă se rezolvăprin adăugarea unui tabel intermediar între cele două şi care va face legătura între roluri si taskuri.
In acest capitol intră şi relația intre contoare si consumatori. Ea s-a rezolvat prin tabela intermediară Consumuri ce conține toate consumurile asciate unui contor pe un anumit consumator.

In urma etapei de implementare a relațiilor dintre tabele, rezultă următoarea structură logică a bazei de date:



3.6. Aplicarea regulilorde normalizare

Sunt câteva reguli numite "de normalizare" prin care se poate verifica dacă o bază de date este sau nu structurată corect. Normalizarea bazei de date reprezintă procesul prin care se verifică dacă structura bazei este conformă cu regulile de normalizare și ajustarea acesteia în cazul identificării unor neconcordanțe.

Sunt definite cinci reguli de normalizare, dar în majoritatea cazurilor ne oprim doar la verificarea primelor trei forme normale.

Prima formă normală verifică ca la intersecția dintre o linie și o coloană dintr-un tabel să fie o singură informație, nu o listă. De exemplu, nu pot fi mai multe coduri într-o singură celulă a tabelului Contoare. Această regulă este destul de evidentă și simplu de implementat. Toate tabelele din baza noastră verifică prima formă normală.

A doua formă normală se aplică tabelelor care au cheia primară formată din mai multe coloane. Ea specifică că în acest caz, toate coloanele din tabel trebuie să fie dependente de întreaga cheie primară, nu numai de o coloană din această cheie.

Problema formei normale numărul doi apare la implementarea incorectă a relațiilor de tipul mai-multi-la-mai-multi. Intotdeauna aceast tip de relație se rezolvă prin adăugarea unui tabel suplimentar de legătură între cele două entități legate multiplu. Dacă aceste relații sunt corect implementate, atunci forma normală doi se verifică implicit.

Forma normală trei verifică dependențele tranzitive. Spunem că o coloană este dependentă tranzitiv de cheia primară dacă în această dependență se interpune o altă coloană. Forma normală trei nu admite dependențe tranzitive, adică o coloană dintr-o tabelă trebuie să fie dependentă numai de cheia primară, nu și de alte coloane din tabelă.

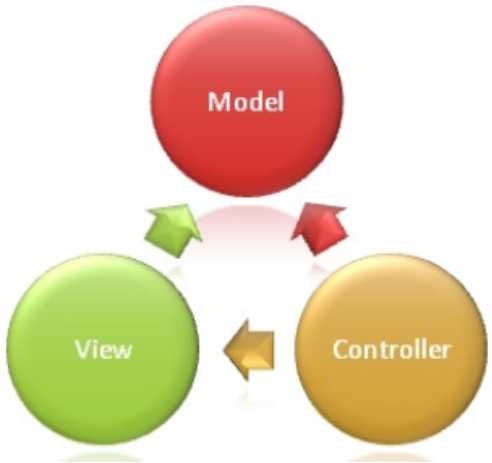
In urma analizei se poate constata ca baza de date proiectată satisface toate cele trei forme normale, deci este optimizată din acest punct de vedere.

4. Proiectarea componentei de interfață cu utilizatorul

Interfața cu utilizatorul reprezintă factorul cheie al sistemului SigMET. Chiar dacă sistemul implementează funcționalități complexe şi foarte utile, dacă interfața cu beneficiarul final nu este optimă, atunci sunt foarte mari şansele ca sistemul să nu fie accepatat de către acesta.
De aceea, în realizarea interfeței trebuie implementate cele mai moderne instrumente de design a paginilor Web care să prezinte într-un mod atractiv şi uşor de folosit informația dată de sistem.

4.1. Tehnologia ASP MVC

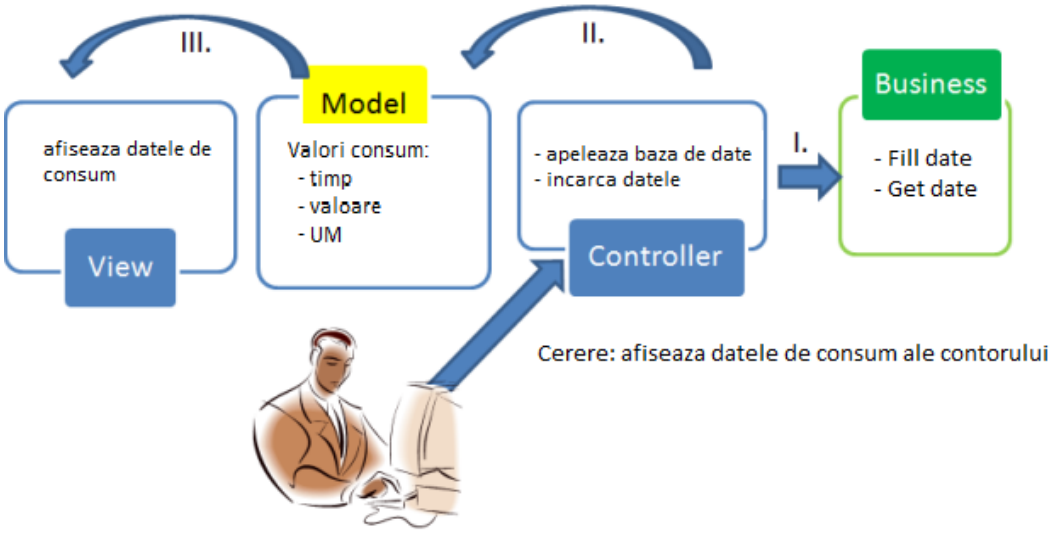
In ultimul timp tehnologiile de realizare a paginilor Web au evoluat spectaculos, punând la dispoziția utilizatorului instrumente de lucru care fac viața programatorului de pagini Web să fie mult mai uşoară. Au apărut frame-work-uri de lucru ce înglobează într-un singur sistem toate componentele necesare în implementarea paginilor Web: baza de date, clasele ce țin de nivelul de business, clasele de vizualizare, etc. In felul acesta, programatorul are la dispoziție un sistem integrat în care poate să-şi dezvolte şi să testeze întreaga arhitectură a paginii Web.
Tehnologia ASP MVC dezvoltată de firma Microsoft răspunde pe deplin acestor cerințe ale dezvoltatorilor şi reprezintă platforma de lucru modernă pe care toți mai mulți programatori o aleg pentru dezvoltarea portalurilor Web. Arhitectura Model-View-Controller (MVC) separă o aplicație software în trei componente principale: modelul, vizualizarea, și controllerul. Cadrul MVC include următoarele componente:



Modelul: obiectele model sunt componente ale aplicației care inglobează datele de lucru şi pun în aplicare logica de funcționare a sistemului. De exemplu, tabela "Contoare" din baza de date reprezintă un model de lucru ce va fi inglobat într-o clasă corespunzătoare. În aplicații mici, modelul este adesea o separare conceptuală în loc de una fizică. De exemplu, dacă cererea citeşte numai un set de date şi îl trimite la vizualizare, aplicația nu are un nivel de model fizic şi clase asociate. În acest caz, setul de date preia rolul unui obiect model.

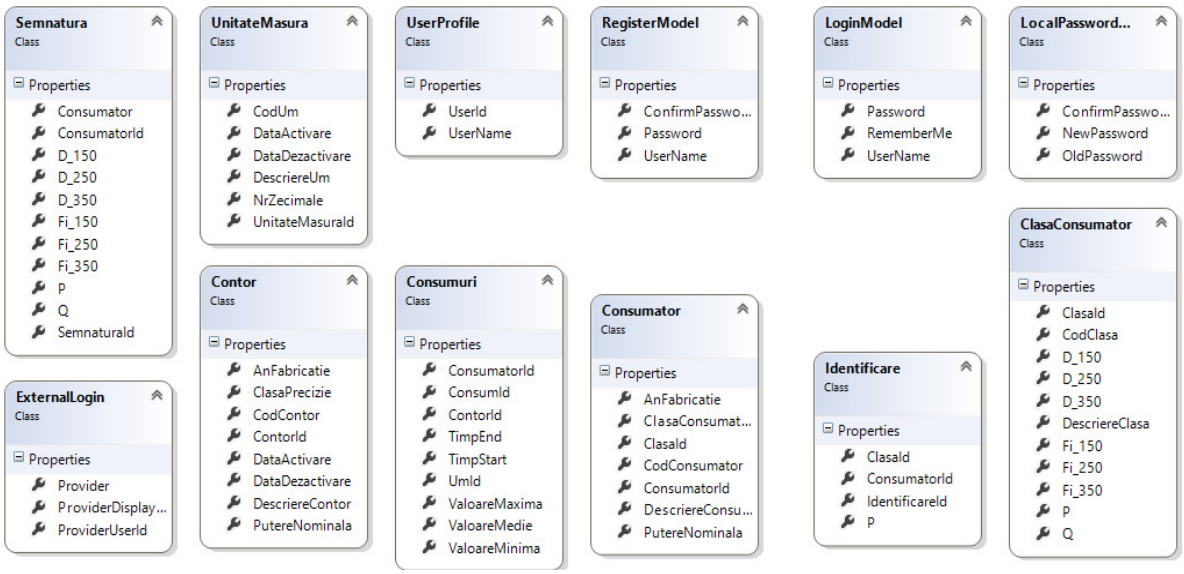
Vizualizarea: reprezintă interfața cu utilizatorul a aplicației (UI). De obicei, această interfață este creată din datele modelului. De exemplu, vizualizarea primeşte ca model o listă de consumuri de la un contor şi afişează acea listă sub forma unui tabel.

Controlerul: este componenta care se ocupă de interacțiunea cu utilizatorul. Pe baza cererii făcute de utilizator, controlerul cere modelului datele respective şi apoi le trimite către vizualizare.

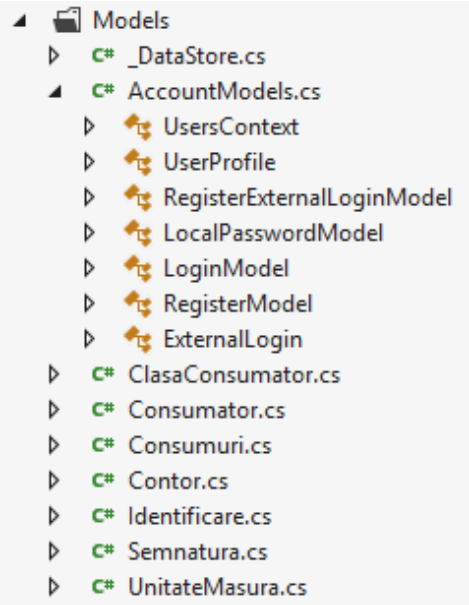


4.2. Proiectarea modelului

Clasele de lucru dezvoltate în procesul de programare pentru a implementa componenta de "Model" din arhitectura MVC reprezintă o traducere în cod a tabelelor dezvoltate în baza de date. Prin aceste modele se va face schimbul de informaţie între cele 3 nivele ale aplicaţiei: baza de date, controlerul şi vizualizările. Proiectul sa va dezvolta pe framework-ul Visual Studio ce pune la dispoziţie toate instrumetele necesare în proiectarea şi dezvoltarea portalurilor Web.
In continuare sunt date clasele ce implementează componenta Model a aplicaţiei:



Aceste clase de obiecte sunt salvate in componenta "Model" a soluţiei software dezvoltate în Visual Studio, conform figurii de mai jos:



Tot in cadrul modelului intră şi modulul de conectare la baza de date şi implementarea tuturor operaţiilor de tip CRUD (Create, read, Update, Delete). In acest scop se utilizează instrumentul Entity Framework ce automatizează întregul proces de creare, citire şi modificare a datelor din bază. Munca programatorului se reduce doar la crearea unei clase moştenite din clasa de bază DbContext în care să definească câte o proprietate get,set pentru fiecare obiect din componenta Model:

```
public class DataStoreContext : DbContext
{
    public DataStoreContext()
        : base("SigMet") // ATENTIE: aici trebuie numele bazei de date
    {
    }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }

    public DbSet<UserProfile> UserProfiles { get; set; }

    public DbSet<ClasaConsumator> ClasaConsumatori { get; set; }
    public DbSet<Consumator> Consumatori { get; set; }
    public DbSet<Consumuri> Consumuri { get; set; }
    public DbSet<Contor> Contoare { get; set; }
    public DbSet<Identificare> Identificari { get; set; }
    public DbSet<Semnatura> Semnaturi { get; set; }
    public DbSet<UnitateMasura> UnitatiMasura { get; set; }
}
```

4.3. Proiectarea Controlerului

Controlerul este componenta de legătură între Model, utilizator şi Vizualizare. Pentru fiecare clasă din Model ce trebuie să fie reprezentată în Vizualizare şi să accepte operaţii de tip CRUD, se implementează câte un controler. Acesta include câte o funcţie ce implementează fiecare din operaţiile specifice setului CRUD. In continuare se exemplifică controlerul realizat pentru modelul "Consumuri", cu detalierea funcţiei de editare a unui obiect din această clasă:

```
public class ConsumuriController : Controller
{
    private DataStoreContext db = new DataStoreContext();
    public ActionResult Index()...
    public ActionResult Details(int id = 0)...
    public ActionResult Create()...
    [HttpPost]
    public ActionResult Create(Consumuri consumuri)...
    public ActionResult Edit(int id = 0)
    {
        Consumuri consumuri = db.Consumuri.Find(id);
        if (consumuri == null)
        {
            return HttpNotFound();
        }
        return View(consumuri);
    }
    [HttpPost]
    public ActionResult Edit(Consumuri consumuri)...
    public ActionResult Delete(int id = 0)...
    [HttpPost, ActionName("Delete")]
    public ActionResult DeleteConfirmed(int id)...
    protected override void Dispose(bool disposing)
    {
        db.Dispose();
        base.Dispose(disposing);
    }
}
```

4.4. Proiectarea Vizualizării

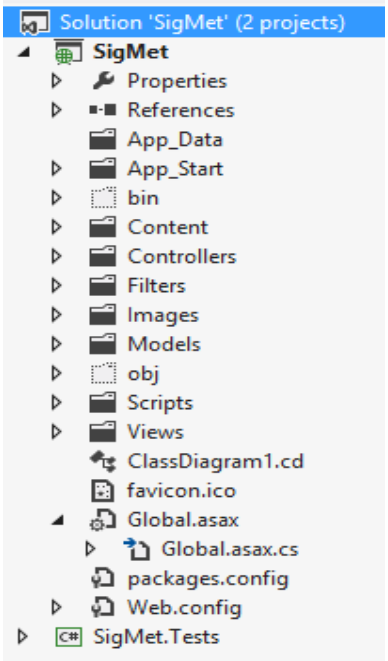
Pentru vizualizare framework-ul Visual Studio pune la dispoziţie librării complexe de obiecte de tip HTML helper. Aceste obiecte generează automat codul HTML necesar pentru vizualizarea unui control HTML cu toate regulile de validare aferente.
Vizualizările vor fi create din pagini de tip cshtml, câte o pagină pentru fiecare din operaţiile CRUD şi pentru fiecare model. Paginile cshtml reprezintă conţinut HTML în care se inserează cod C#, toate acestea la un loc fiind compilate în timpul rulării paginii, rezultând un conţinut HTML pur ce se transmite la browser-ul de internet.
Ca exemplu, mai jos este listată o parte a paginii de afişare a listei de consumatori sub forma unui tabel HTML. Fiecare linie din tabel include şi cele trei linkuri către paginile de editare, detalii şi stergere.

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.ClasaConsumator.CodClasa)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.CodConsumator)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.DescriereConsumator)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.AnFabricatie)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.PutereNominala)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ConsumatorId }) |
            @Html.ActionLink("Details", "Details", new { id=item.ConsumatorId }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ConsumatorId })
        </td>
    </tr>
```

S-au folosit obiectele din biblioteca HtmlHelper pentru afişare text şi link:

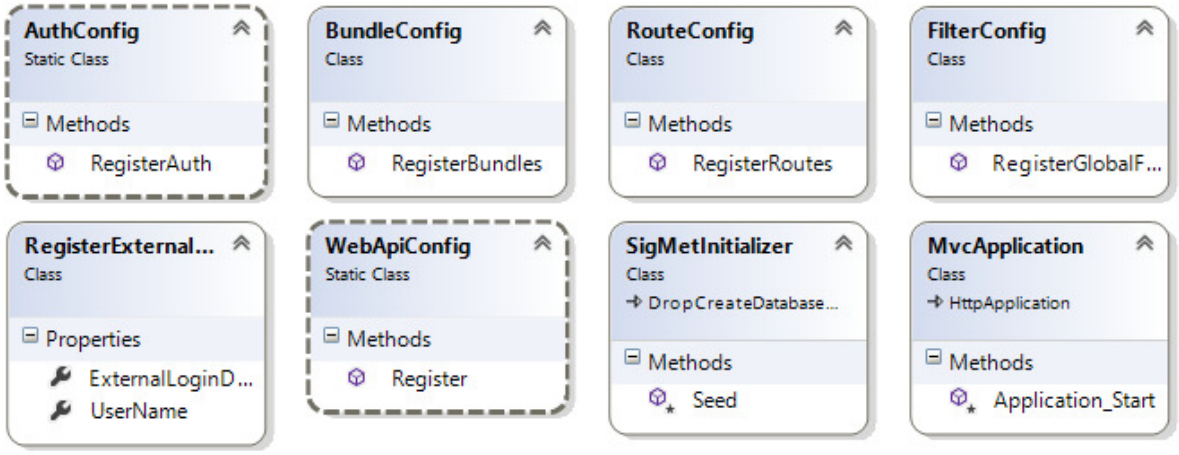
- DisplayFor: afişează textul primit ca parametru printr-o expresie de tip Lambda (=>)
- ActionLink: afişează un link în pagină, are 3 parametri: textul linkului, funcția din controller de tip ActionResult care construieşte pagina HTML de răspuns şi eventual o listă de parametri ce se tranmit controlerului.

4.5. Proiectarea structurii aplicației SigMET



Pe lângă componentele MVC, aplicația software mai include câteva module suplimentare care să rezolve problemele specifice legate de configurarea aplicației, modelarea structurii URL, definirea filtrelor pe diverse operații, modul de startare, valori default, logarea utilizatorilor folosind conturi deja create în rețelele de socializare cunoscute (Facebook, Twiter, Google) etc.

In imaginea alăturată este dată structura arborelui ce include toate aceste componente vizualizat în fereastra "Solution Explorer" din Visual Studio, iar mai jos, lista claselor ce implementează aceste funcționalități.



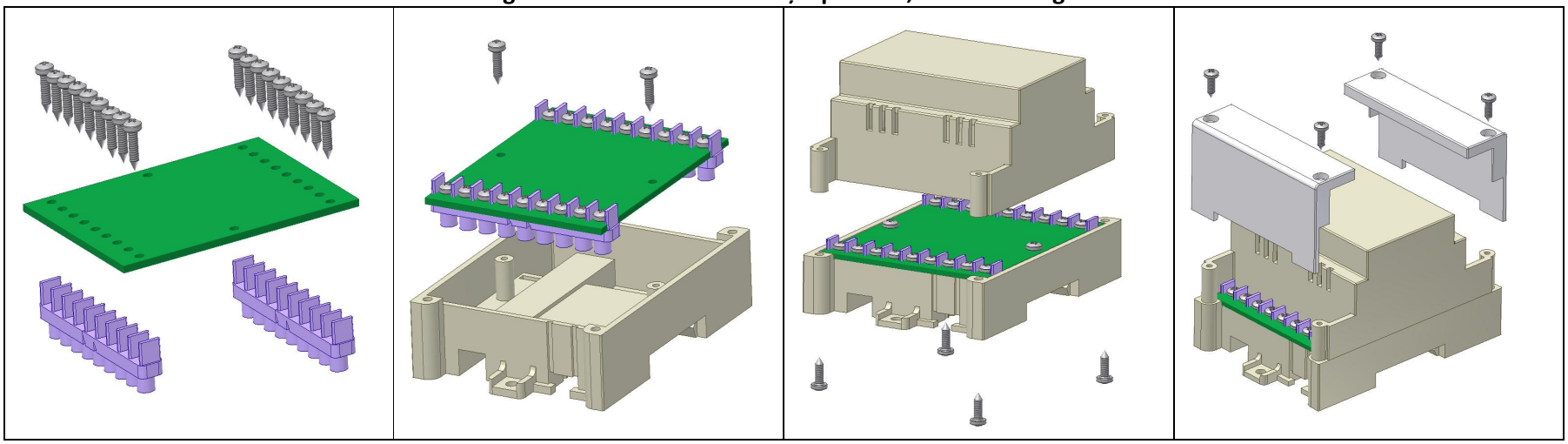
5. Proiectarea componentelor mecanice şi a celor hardware auxiliare

Pentru componentele mecanice şi hardware auxiliare sistemului SigMET au fost proiectate doar acele repere a căror execuţie poate fi realizată la preţuri mai mici decât cele ale componentelor cu funcţii similare disponibile pe piaţă. S-a urmărit astfel menţinerea unui cost estimat al produsului la un nivel cât mai scăzut în vederea creşterii gradului de acceptabilitate.
Au fost proiectate:

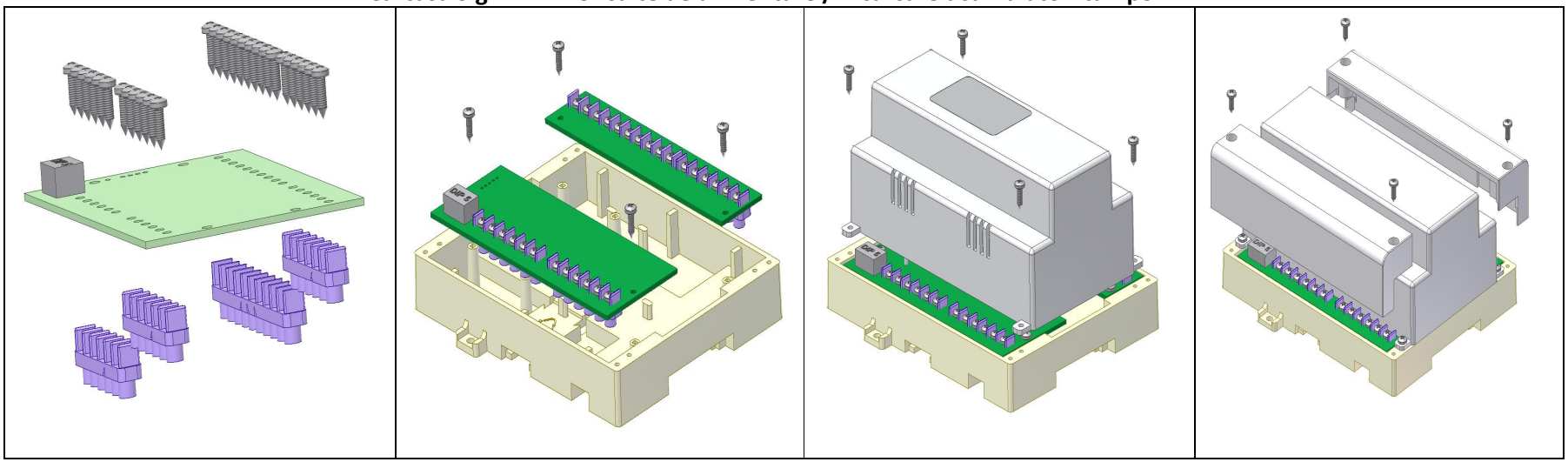
- carcasa pentru circuitele de achiziţie primară cu senzori Hall a semnalelor de tensiune şi curent, carcasă care va fi montată pe şină DIN în imediata vecinătate a firidei;
- carcasa pentru circuitul de alimentare neîntreruptibilă a sistemului SigMET
- schema electronică a sursei de alimentare în comutaţie, sursă care asigură şi încărcarea / comutarea acumulatorilor locali pentru evitarea pierderii datelor inregistrate în cazul apariţiei unei înreruperi în alimentarea cu energie electrică.

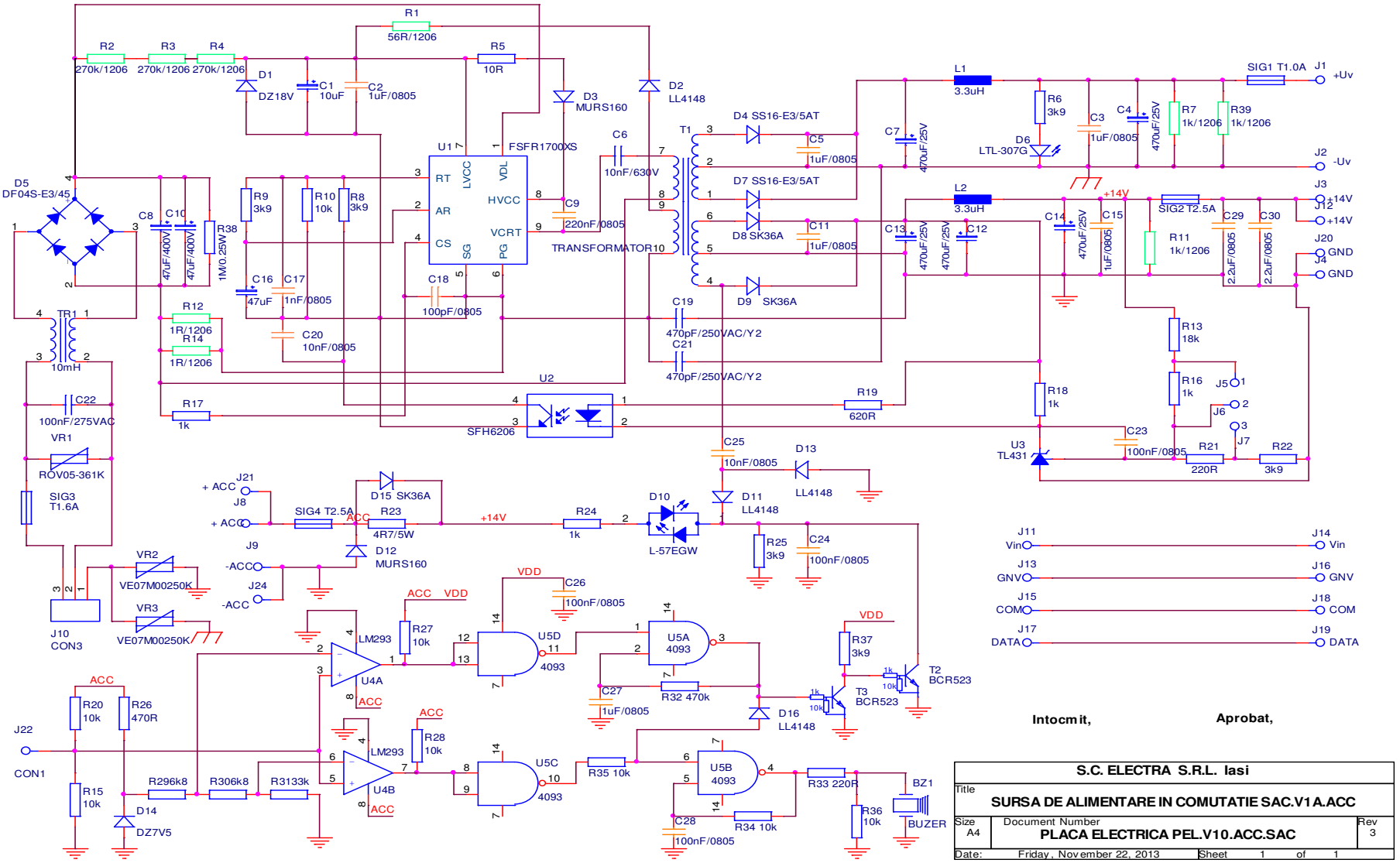
Sunt prezentate în figurile următoare elementele descriptive 3D şi de montare pentru cele două carcase şi respectiv schema electronică a sursei de alimentare.

Carcasă SigMET 1 - Circuite de achiziţie primară, montate lângă firidă



Carcasă SigMET 2 - Circuite de alimentare / incărcare acumulatori tampon





Intocmit, **Aprobat,**

S.C. ELECTRA S.R.L. Iasi		
Title: SURSA DE ALIMENTARE IN COMUTATIE SAC.V1A.ACC		
Size: A4	Document Number: PLACA ELECTRICA PEL.V10.ACC.SAC	Rev: 3
Date: Friday, November 22, 2013	Sheet 1 of 1	