ELEKTRYKA 2009
Zeszyt 3 (211) Rok LV

Radosław SOKÓŁ
Instytut Elektrotechniki i Informatyki, Politechnika Śląska w Gliwicach

OBIEKTOWA REPREZENTACJA MODELU SYSTEMU

S**treszczenie.** Implementacja algorytmów optymalizacyjnych operujących na danych stanu systemu elektroenergetycznego wymaga stworzenia informatycznej reprezentacji zbioru tych danych. Największą prostotę i elastyczność zastosowania gwarantują obiektowe techniki programowania. Artykuł prezentuje obiektową implementację opisu systemu elektroenergetycznego, stworzoną na potrzeby oprogramowania opracowywanego w ramach projektu badawczego N511 001 32/0852.

Słowa kluczowe: system elektroenergetyczny, model, C++, programowanie obiektowe

OBJECT-ORIENTED POWER SYSTEM MODEL REPRESENTATION

Summary. An implementation of an optimization algorithm working on data of a power system state requires defining an in-memory representation of the data. A high level of usage friendliness and flexibility can be achieved by means of object-oriented programming techniques. The paper presents an object-oriented power system model representation, created as a part of the research project No N511 001 32/0852.

Keywords: power system, model, C++, object-oriented programming

1. WSTĘP

Oprogramowanie wyznaczające rozpływ mocy w systemie elektroenergetycznym może ograniczać się do `zapisania` stanu systemu bezpośrednio w postaci macierzy admitancyjnej, stanowiącej podstawę dalszych obliczeń[1]. Takie podejście jest jednak niemożliwe w przypadku programu, którego zadaniem jest nie tylko przedstawienie wyników obliczeń, ale też wizualizacja danych i wyników oraz interaktywna edycja stanu systemu. W szczególności operacje, takie jak:

- włączanie i wyłączanie linii,
- włączanie i wyłączanie bloków wytwórczych generatorów,

[1] Macierzy admitancyjnej musi towarzyszyć też zbiór danych opisujących ograniczenia nierównościowe (fizyczne) nakładane na poszczególne elementy systemu elektroenergetycznego. Również ten zbiór danych można jednak zapisać w postaci macierzy.

- zmiana ograniczeń nierównościowych poszczególnych elementów systemu,
- zmiana parametrów węzłów lub linii

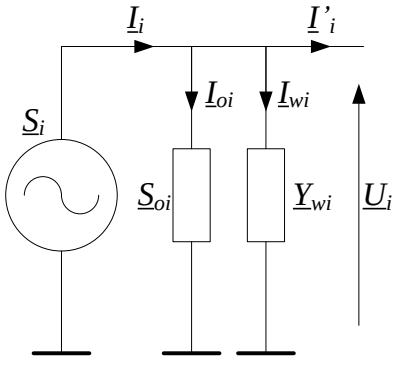
wymagają, po pierwsze, przechowywania w pamięci aktualnych parametrów każdego elementu systemu, a po drugie – możliwości łatwego wprowadzania zmian i generowania nowych macierzy admitancyjnej i ograniczeń nierównościowych, stanowiących punkt wyjścia do kolejnego procesu obliczeniowego.

Największą elastyczność oraz elegancję rozwiązania można uzyskać stosując techniki *programowania obiektowego*. Programowanie obiektowe pozwala stworzyć abstrakcję bezpośrednio odwzorowującą pewne realia fizyczne, przerzucając część obciążenia związanego z budowaniem struktury zbioru danych z programisty na kompilator języka i umożliwiając ścisłe wiązanie danych i operacji realizowanych na tych danych.

W poniższym artykule Autor przedstawia implementację obiektowego modelu systemu elektroenergetycznego, zrealizowaną na potrzeby projektu badawczego, mającego na celu stworzenie oprogramowania wyznaczającego optymalny rozpływ mocy w systemie elektroenergetycznym.

2. MATEMATYCZNY MODEL SYSTEMU

System elektroenergetyczny to skomplikowany układ wzajemnie połączonych węzłów wytwórczych i odbiorczych. W celu dokładnego opisu konieczne jest stworzenie odpowiedniego modelu, który będzie uwzględniał wszystkie elementy systemu. Ponieważ w systemie występują zarówno węzły odbiorcze, jak i wytwórcze konieczne jest ich prawidłowe zamodelowanie. Ze względów praktycznych wprowadzono tzw. uogólniony węzeł systemu elektroenergetycznego (rys. 1) oraz model linii przesyłowej (rys. 2), bazujące na [1] [2].

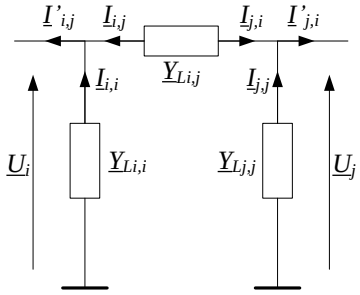


Rys. 1. Uogólniony węzeł systemu elektroenergetycznego
Fig. 1. A generic model of a power system node

gdzie: $\underline{S}_i$ – moc zespolona generatora w węźle *i* (równa zero dla węzłów odbiorczych),
$\underline{S}_{oi}$ – moc zespolona pobierana w węźle *i* (dla węzłów wytwórczych są to potrzeby własne, dla węzłów odbiorczych jest to moc pobierana przez odbiorcę),

$\underline{U}_i$ – napięcie skuteczne zespolone w węźle *i*, $\underline{U}_i = U_i e^{j\varphi i}$

$\underline{Y}_{wi}$ – admitancja zespolona węzła *i*.



Rys. 2. Model linii przesyłowej
Fig. 2. A generic model of a power line

gdzie: $\underline{U}_i$ – napięcie skuteczne zespolone w węźle *i*,

$\underline{U}_j$ – napięcie skuteczne zespolone w węźle *j*,

$\underline{Y}_{Li,i}$ – admitancja zespolona poprzeczna linii w węźle *i*,

$\underline{Y}_{Lj,j}$ – admitancja zespolona poprzeczna linii w węźle *j*,

$\underline{Y}_{Li,j}$ – admitancja zespolona podłużna linii pomiędzy węzłami *i* i *j*.

Model linii przesyłowej uwzględnia możliwość transformacji napięcia. Parametry linii muszą zostać przeliczone z uwzględnieniem istnienia transformatora. Model ten musiałby ulec zmianie, jeżeli konieczna byłaby zmiana nastaw transformatora w czasie trwania obliczeń.

3. IDENTYFIKACJA I PODZIAŁ ELEMENTÓW SYSTEMU ELEKTROENERGETYCZNEGO

Elementy systemu elektroenergetycznego[2] (węzły i linie) są identyfikowane unikatowymi nazwami, nadawanymi przez zarządcę danych opisujących system. Nazwy te nie nadają się do szybkiego odszukiwania elementów systemu, a tym bardziej do opisywania zależności między elementami systemu a elementami macierzy i wektorów tworzących układ równań rozpływu mocy. Z tego powodu konieczne jest nadanie elementom systemu identyfikatorów w postaci liczb całkowitych z ciągłego zakresu od 1 do:

- N_W – w przypadku węzłów (przy czym N_O oznacza liczbę węzłów odbiorczych, a N_G – liczbę generatorów; $N_W = N_O + N_G$),
- N_L – w przypadku linii.

W celu skrócenia czasu wyznaczania optymalnego rozpływu mocy w dużych SEE zasadne jest dokonanie podziału grafu systemu na mniejsze części i niezależne przeprowadzanie obliczeń dla każdej z nich [3-5]. W takim przypadku numeracja węzłów i linii musi

[2] Dalej stosowany będzie skrót *SEE*.

zostać powtórzona w każdym podgrafie z osobna, przy czym należy zachować informację o zależności między elementem SEE, znajdującym się w podgrafie, a jego identyfikatorem globalnym[3]. W tym celu wprowadzono następujące klasy identyfikatorów elementów:

- **GUNN** (ang. *Global Unique Node Number*) – globalny, unikatowy identyfikator węzła w ramach całego SEE niezależnie od dokonanego podziału na podgrafy,
- **GULN** (ang. *Global Unique Line Number*) – globalny, unikatowy identyfikator linii w ramach całego SEE niezależnie od dokonanego podziału na podgrafy,
- **LUNN** (ang. *Local Unique Node Number*) – lokalny identyfikator węzła, unikatowy wyłącznie w ramach jednego podgrafu,
- **LULN** (ang. *Local Unique Line Number*) – lokalny identyfikator linii, unikatowy wyłącznie w ramach jednego podgrafu.

Jednocześnie wprowadzono dwa bufory tłumaczące (**TLB**, ang. *Translation Lookaside Buffer*), dokonujące szybkiej translacji numeracji globalnej na lokalną i lokalnej na globalną w ramach wybranego podgrafu SEE.

Podgrafy identyfikowane są unikatowymi globalnymi identyfikatorami całkowitymi **GUSN** (ang. *Global Unique Subgraph Number*), przy czym – w przeciwieństwie do identyfikatorów węzłów i linii – numeracja podgrafów nie musi być ciągła i w wielu przypadkach może bezpośrednio odpowiadać danym wejściowym[4].

Program zawiera następujące definicje typów identyfikatorów podgrafów, węzłów i linii:

```
typedef unsigned int gunn_t;
typedef unsigned int lunn_t;
typedef unsigned int guln_t;
typedef unsigned int luln_t;
typedef unsigned int gusn_t;
```

4. OBIEKTOWA REPREZENTACJA WĘZŁÓW I LINII

Zgodnie z opisanym w punkcie 2 uogólnionym modelem węzła SEE, obiektowa reprezentacja węzła SEE (klasa Node) obejmuje zarówno węzły odbiorcze, jak i wytwórcze. Umożliwia to między innymi traktowanie węzłów wytwórczych niepodlegających regulacji jak węzłów odbiorczych z narzuconą na sztywno mocą generowaną, bez utraty informacji o poziomie generacji mocy oraz potrzebach własnych węzła.

Klasa Line opisuje dowolną linię składową SEE. Opisywana linia może dokonywać transformacji napięcia, w tym z zespoloną przekładnią transformatora. Parametry transformatora są przeliczane na impedancję podłużną i poprzeczną linii w momencie

[3] Jest to niezbędne w celu przepisania wyników obliczeń z poszczególnych podgrafów do globalnej bazy opisującej stan całego SEE.
[4] W szczególności danym wejściowym wygenerowanym z myślą o programie MatPower, w którego przypadku identyfikatory obszarów są przypisywane w sposób dowolny i nieciągły.

tworzenia macierzy admitancyjnej systemu, co utrudnia dynamiczne modyfikowanie nastaw transformatorów w ramach procesu optymalizacyjnego.

4.1. Typy danych

W celu umożliwienia globalnej zmiany dokładności obliczeń wszystkie wartości zmiennoprzecinkowe zostały zapisane z użyciem własnych typów `numeric_type` oraz `numeric_complex`. Standardowo typy te zdefiniowane są w następujący sposób:

```
typedef double numeric_type;
typedef std::complex<numeric_type> numeric_complex;
```

4.2. Definicja klasy Node

Definicja klasy `Node` została przedstawiona na listingu 1.

Listing 1

Definicja klasy Node

```
enum NodeType { Unknown, Consumer, Producer };

struct Node
{

   NodeType              Type;
   bool                  reference;
   gunn_t                GUNN;
   std::string           Name;
   unsigned int          Area;
   unsigned int          OrigArea;

   bool                  Coord;
   bool                  Duplicate;
   bool                  Variable;

   numeric_type          U;
   numeric_type          PhaseU;
   numeric_type          Ubase;
   numeric_complex       Y;
   numeric_complex       Sd;
   numeric_complex       Strans;
   numeric_type          Umax;
   numeric_type          Umin;

   numeric_complex       Sg;
   numeric_type          Pmax;
   numeric_type          Pmin;
   numeric_type          Qmax;
   numeric_type          Qmin;

   numeric_type          Sb;

   numeric_type          PriceStartup;
   numeric_type          PriceShutdown;
   numeric_type          Price2;
```

```
  numeric_type          Price1;
  numeric_type          Price0;

  Node();
  Node(const Node&);

  inline numeric_complex Uc() const throw();

  inline numeric_type Cost() const throw();
  inline numeric_type Cost(const numeric_type p) const throw();

  inline bool IsDuplicate() const throw();
  inline bool IsConsumer() const throw();
  inline bool IsProducer() const throw();
  inline bool IsPureProducer() const throw();
  inline bool IsCompensator() const throw();
  inline bool IsReference() const throw();

  inline void SetAsCoord() throw();
  inline bool IsCoordNode() const throw();

  inline NodeType GetType() const throw();
  char * GetTypeString() const;

  inline bool FixedSgConstrainst() const throw();

};
```

4.3. Opis elementów składowych klasy Node

W tabeli 1 zawarto opis poszczególnych pól składowych klasy `Node`, przedstawionej na listingu 1. Analogicznie, w tabeli 2 znajduje się lista metod implementowanych przez tę klasę.

Tabela 1

Opis pól składowych klasy Node

Pole	Opis
`Type`	Typ węzła. Możliwe wartości: • `Unknown` – typ nieznany[5], • `Consumer` – węzeł odbiorczy (niepodlegający regulacji), • `Producer` – węzeł wytwórczy (mogący podlegać regulacji).
`Reference`	Jeżeli `true`, węzeł jest węzłem odniesienia.
`GUNN`	Identyfikator GUNN węzła. W przypadku opisu węzła należącego do globalnej bazy stanu SEE pole to jest nieużywane i zawiera indeks elementu w bazie. W przypadku opisu węzła należącego do lokalnej bazy podgrafu SEE pole to umożliwia szybkie wyszukanie opisu tego samego węzła w bazie globalnej.
`Name`	Nazwa węzła ustalona w pliku źródłowym przez zarządcę danych opisujących SEE.
`Area`	Numer GUSN podgrafu, do którego należy dany węzeł.

[5] Pojawienie się w czasie obliczeń węzła o nieznanym typie stanowi jasną informację o błędzie w kodzie wczytującym dane z pliku źródłowego do pamięci operacyjnej komputera.

cd. tabeli 1

`OrigArea`	Numer GUSN podgrafu, do którego powinien należeć dany węzeł zgodnie z danymi zapisanymi w pliku danych źródłowych[6].
`Coord`	Określa, czy węzeł jest węzłem koordynacyjnym[7].
`Duplicate`	Określa, czy węzeł jest duplikatem węzła koordynacyjnego, stworzonym automatycznie w sąsiednim podgrafie.
`Variable`	Określa, czy węzeł podlega regulacji w ramach puli węzłów wytwórczych centralnie dysponowanych.
`U, PhaseU`	Zapis napięcia względnego w węźle w postaci moduł-faza.
`Ubase`	Napięcie bazowe węzła.
`Y`	Admitancja węzłowa zespolona w jednostkach względnych.
`Sd`	Moc potrzeb własnych w jednostkach względnych.
`Strans`	Moc przesyłowa międzygrafowa w jednostkach względnych. Niezerowa jedynie w przypadku węzłów koordynacyjnych oraz ich duplikatów.
`Umin, Umax`	Ograniczenie nierównościowe modułu napięcia względnego w danym węźle (ograniczenie fizyczne).
`Sg`	Moc generowana w węźle wytwórczym.
`Pmin, Pmax`	Ograniczenie nierównościowe generacji mocy czynnej w danym węźle (ograniczenie fizyczne).
`Qmin, Qmax`	Ograniczenie nierównościowe generacji mocy biernej w danym węźle (ograniczenie fizyczne).
`Sb`	Moc bazowa.
`PriceStartup`	Koszt rozruchu generatora.
`PriceShutdown`	Koszt wstrzymania pracy generatora.
`Price2, Price1, Price0`	Współczynniki trójmianu kosztów K generacji mocy, gdzie: $K = \text{Price2}\cdot P^2 + \text{Price1}\cdot P + \text{Price0}$

Tabela 2

Opis metod składowych klasy Node

Metoda	Opis
`Node()`	Konstruktor domyślny, gwarantuje możliwość wykrycia niezainicjalizowanych obiektów klasy `Node`.
`Node(`*N*`)`	Konstruktor kopiujący.
`Uc()`	Zwraca napięcie w węźle w postaci zespolonej.
`Cost()` `Cost(`*P*`)`	Zwraca koszt generacji mocy dla danego węzła przy jego aktualnym lub zadanym poziomie generacji.
`IsDuplicate()`	Zwraca `true`, jeżeli dany węzeł jest duplikatem węzła koordynacyjnego istniejącego w innym podgrafie.
`IsConsumer()`	Zwraca `true`, jeżeli węzeł jest węzłem odbiorczym.
`IsProducer()`	Zwraca `true`, jeżeli węzeł jest węzłem wytwórczym.
`IsPureProducer()`	Zwraca `true`, jeżeli węzeł jest węzłem wytwórczym generującym wyłącznie moc czynną.

[6] W przypadku korzystania z podziału grafu systemu na podgrafy, pole `OrigArea` zawiera ten sam numer GUSN podgrafu co pole `Area`. W przypadku zablokowania podziału na podgrafy i traktowania SEE jako jednej całości pole `OrigArea` umożliwia zorientowanie się, do którego podgrafu należałby węzeł, gdyby podział jednak istniał (pole `Area` w takim przypadku zawsze będzie zawierać wartość 1).

[7] Węzły koordynacyjne to węzły znajdujące się na granicy podgrafu i mające swoje lustrzane odbicia (*duplikaty*) w sąsiednich podgrafach (w celu koordynowania stanu systemu na granicy dwóch podgrafów). Techniki podziału grafu SEE na podgrafy i koordynowania obliczeń zostały opisane w [5-6].

cd. tabeli 2

`IsCompensator()`	Zwraca `true`, jeżeli węzeł jest węzłem wytwórczym pełniącym rolę regulowanego kompensatora.
`IsReference()`	Zwraca `true`, jeżeli węzeł jest oznaczony w pliku danych źródłowych jako węzeł odniesienia procesu wyznaczania rozpływu mocy.
`SetAsCoord()`	Zaznacza węzeł jako koordynacyjny.
`IsCoordNode()`	Zwraca `true`, jeżeli węzeł został oznaczony jako koordynacyjny.
`GetType()`	Zwraca kod typu węzła (patrz definicja pola `Type` w tabeli 1).
`GetTypeString()`	Zwraca słowny opis typu węzła.
`FixedSgConstraints()`	Zwraca `true`, jeżeli węzeł ma narzucone ograniczenia nierównościowe generacji mocy uniemożliwiające regulację[8].

4.4. Definicja klasy Line

Definicja klasy `Line` została przedstawiona na listingu 2.

Listing 2

Definicja klasy Line

```
struct Line
{

   guln_t                guln;
   unsigned int          StartNode, EndNode;

   numeric_complex       Z;
   numeric_complex       Y;
   numeric_type          maxSd;
   numeric_type          maxSk;
   numeric_type          maxSa;
   numeric_type          theta;
   numeric_type          psi;
   numeric_type          Imax;

   guln_t GULN() const throw();

   bool AnchoredAt(const unsigned int gunn) const throw();

   bool IsTransformer() const throw();
   numeric_complex T() const throw();

   numeric_type mY() const;

   numeric_type I(const bool AtStart = true) const throw();
   numeric_complex PowerTransfer(const bool AtStart = true) const;

   Line();
   Line(const Line&);

   void SwapEnds();
   void Enable(const bool b) throw();
   bool IsEnabled() const throw();

};
```

[8] W takim przypadku węzeł powinien zostać przekształcony w węzeł odbiorczy generujący moc na stałym poziomie.

4.5. Opis elementów składowych klasy Line

Tabela 3 zawiera opis poszczególnych pól składowych klasy `Line`, przedstawionej na listingu 2. Analogicznie, tabela 4 prezentuje metody implementowane przez tę klasę.

Tabela 3

Opis pól składowych klasy Line

Pole	Opis
`guln`	Identyfikator GULN linii. W przypadku opisu linii należącego do globalnej bazy stanu SEE pole to jest nieużywane i zawiera indeks elementu w bazie. W przypadku opisu linii należącego do lokalnej bazy podgrafu SEE pole to umożliwia szybkie wyszukanie opisu tej samej linii w bazie globalnej.
`StartNode, EndNode`	Identyfikatory LUNN (w przypadku podgrafu) lub GULN (w przypadku opisu całego SEE) węzłów początkowego i końcowego danej linii.
`Z`	Impedancja względna wzdłużna.
`Y`	Admitancja względna poprzeczna.
`maxSd, maxSk, maxSa`	Obciążalność linii długotrwała (S_d), krótkotrwała (S_k) oraz awaryjna (S_a), w MVA.
`theta, psi`	Przekładnia (theta) i kąt regulacji fazowej (psi) transformatora (w przypadku braku transformatora obywa parametry ustawiane są na zero).
`Imax`	Ograniczenie prądu skutecznego w linii.

Tabela 4

Opis metod składowych klasy Line

Metoda	Opis
`Line()`	Konstruktor domyślny gwarantuje możliwość wykrycia niezainicjalizowanych obiektów klasy `Line`.
`Line(`*`L`*`)`	Konstruktor kopiujący.
`GULN()`	Zwraca zbuforowany identyfikator GULN linii.
`AnchoredAt(`*`gunn`*`)`	Zwraca `true`, jeżeli linia zaczyna się lub kończy w węźle o identyfikatorze GUNN równym *gunn*.
`IsTransformer()`	Zwraca `true`, jeżeli linia zawiera transformator.
`T()`	Zwraca zespoloną przekładnię transformatora lub zero, jeżeli linia nie zawiera transformatora.
`mY()`	Zwraca moduł admitancji wzdłużnej linii.
`I(`*`s`*`)`	Zwraca prąd płynący linią. Jeżeli *s* jest równe `true`, zwracany jest prąd widziany przez węzeł początkowy; w przeciwnym przypadku – przez węzeł końcowy.
`PowerTransfer(`*`s`*`)`	Zwraca wartość mocy przesyłanej linią. Jeżeli *s* jest równe `true`, zwracana jest moc widziana przez węzeł początkowy; w przeciwnym przypadku – przez węzeł końcowy.
`SwapEnds()`	Zamienia początek linii z jej końcem.
`Enable(`*`b`*`)`	W zależności od wartości parametru *b* włącza lub wyłącza linię.
`IsEnabled()`	Zwraca `true`, jeżeli linia jest włączona.

5. KLASA GRAPHSTATE

Klasa `GraphState` umożliwia pobieranie (w jednym kroku) bilansu mocy w wybranym podgrafie SEE (lub w całym SEE) i wykonywanie prostych operacji na tak zgromadzonych danych.

5.1. Cel stosowania

Klasa `GraphState` ogranicza liczbę operacji wymaganych przy pobieraniu informacji o rozpływie mocy w podgrafie bez uciekania się do buforowania tych informacji wewnątrz obiektów przechowujących stan podgrafu SEE (patrz punkt 6).

5.2. Definicja klasy

Definicja klasy `GraphState` została przedstawiona na listingu 3.

Listing 3

Definicja klasy GraphState

```
class GraphState
{

  public:

  GraphState() throw();

  GraphState(
      const numeric_complex ProductionCapacity,
      const numeric_complex Production,
      const numeric_complex Consumption,
      const numeric_complex Transfer,
      const numeric_complex Losses) throw();

  numeric_complex ProductionCapacity() const throw();
  numeric_complex Production() const throw();
  numeric_complex Consumption() const throw();
  numeric_complex Losses() const throw();
  numeric_complex Transfer() const throw();
  numeric_complex Balance() const throw();

  void operator += (const GraphState &g) throw();

};
```

5.3. Opis elementów składowych klasy

Klasa `GraphState` jest całkowicie zahermetyzowana. W tabeli 5 przedstawiono wszystkie metody przez nią udostępniane.

Tabela 5

Opis metod składowych klasy GraphState

Metoda	Opis
`GraphState()`	Tworzy nowy obiekt klasy.
`GraphState(...)`	Tworzy nowy obiekt klasy, od razu odpowiadający pewnemu wyznaczonemu rozpływowi mocy w podgrafie.
`ProductionCapacity()`	Zwraca maksymalne zdolności wytwórcze generatorów znajdujących się w wybranym podgrafie.
`Production()`	Zwraca aktualny poziom generacji mocy w wybranym podgrafie.
`Consumption()`	Zwraca aktualny pobór mocy w wybranym podgrafie.
`Losses()`	Zwraca wartość strat przesyłowych mocy w wybranym podgrafie.
`Transfer()`	Zwraca sumaryczną wartość przesyłu mocy między wybranym podgrafem a podgrafami sąsiednimi.
`Balance()`	Zwraca bilans mocy w podgrafie.
`operator+=(`*g*`)`	Umożliwia sumowanie obiektów klasy `GraphState` i wyznaczanie sumarycznego stanu zbioru podgrafów.

6. KLASA SYSTEMSTATE

Klasa `SystemState` unifikuje zarządzanie bazą stanu dowolnego fragmentu SEE. Dzięki wbudowanym buforom TLB może być wykorzystywana do przechowywania danych opisujących stan węzłów i linii należących do podgrafu systemu (i numerowanych za pomocą lokalnych indeksów LUNN i LULN, unikatowych tylko w ramach podgrafu), jak i do całego SEE (i numerowanych za pomocą globalnych identyfikatorów GUNN i GULN).

6.1. Definicja klasy

Definicja klasy `SystemState` została przedstawiona na listingu 4.

Listing 4

Definicja klasy SystemState

```
class SystemState
{

  public:

  bool SortNoNg;

  SystemState(const bool Global);
  SystemState(const SystemState&);
  ~SystemState();

  SystemState & operator = (const SystemState&);

  unsigned int Nw() const throw();
  unsigned int No() const throw();
  unsigned int Ng() const throw();
  unsigned int Nl() const throw();
```

```
  gusn_t HighestGUSN() const;

  bool IsGlobal() const throw();

  bool IsLunnValid(const lunn_t lunn) const throw();
  bool IsGunnValid(gunn_t gunn) const throw();

  numeric_complex SumSd() const throw();
  numeric_complex SumSgmax() const throw();

  GraphState GetState() const throw();

  numeric_type PowerCost() const throw();
  numeric_type ConstantCost() const throw();

  void Clear();

  void CancelTransfers() throw();

  lunn_t AddNode(const Node&);
  luln_t AddLine(const Line&);

  void RemoveNodeByLunn(lunn_t);
  void RemoveLineByLuln(luln_t);

  void SwapNodesByLunn(lunn_t lunn1, lunn_t lunn2);
  void SwapNodesByGunn(gunn_t gunn1, gunn_t gunn2);

  void ConvertProducerToConsumerByLunn(lunn_t lunn);
  void ConvertProducerToConsumerByGunn(gunn_t gunn);

  bool GunnExists(const gunn_t) const throw();
  bool GulnExists(const guln_t) const throw();

  Node & GetNodeByName(const char *name);

  Node & GetNodeByGunn(gunn_t gunn);
  const Node & GetNodeByGunn(gunn_t gunn) const;
  Node & GetNodeByLunn(lunn_t lunn);
  const Node & GetNodeByLunn(lunn_t lunn) const;

  Line & GetLineByGuln(guln_t guln);
  const Line & GetLineByGuln(guln_t guln) const;
  Line & GetLineByLuln(luln_t luln);
  const Line & GetLineByLuln(luln_t luln) const;

  Line & GetLineByLunns(lunn_t lunn1, lunn_t lunn2);

  void LevelBy(const numeric_type deltaU);
  void RotateBy(const numeric_type deltaF);

};
```

6.2. Opis elementów składowych klasy

Tabela 6 zawiera opis poszczególnych pól składowych klasy `SystemState`, przedstawionej na listingu 4. Analogicznie, w tabeli 7 zaprezentowano metody implementowane przez tę klasę.

Tabela 6

Opis pól składowych klasy SystemState

Pole	Opis
`SortNoNg`	Wymusza porządkowanie węzłów tak, aby wszystkie odbiorcze znalazły się przed wytwórczymi.

Tabela 7

Opis metod składowych klasy SystemState

Metoda	Opis
`SystemState(`*`Global`*`)`	Tworzy obiekt bazy stanu systemu. Jeżeli *Global* jest równe `true`, baza zakłada indeksowanie elementów identyfikatorami GUNN i GULN i pomija translację. W przeciwnym przypadku wykorzystywane są bufory TLB.
`SystemState(`*`SystemState`*`)`	Konstruktor kopiujący.
`operator=(`*`SystemState`*`)`	Operator przypisania umożliwia stworzenie zapasowej kopii stanu systemu.
`Nw(), No(), Ng(), Nl()`	Zwracają liczbę wszystkich węzłów (N_w), węzłów odbiorczych (N_o), węzłów wytwórczych (N_g) oraz linii (N_l) w danym podgrafie.
`HighestGUSN()`	Zwraca najwyższy identyfikator GUSN podgrafu zapisany w opisach węzłów znajdujących się w danym obiekcie bazy stanu systemu.
`IsGlobal()`	Zwraca `true`, jeżeli dany obiekt bazy stanu systemu został stworzony jako globalny[9].
`IsLunnValid()`	Sprawdza, czy w bazie istnieje węzeł o podanym numerze LUNN.
`IsGunnValid()`	Sprawdza, czy w bazie istnieje węzeł o podanym numerze GUNN.
`SumSd()`	Zwraca sumę mocy pobieranych przez poszczególne węzły, co odpowiada zapotrzebowaniu podgrafu (lub systemu) na moc.
`SumSgmax()`	Zwraca sumę górnych ograniczeń nierównościowych nałożonych na generację mocy, co odpowiada szczytowym możliwościom wytwórczym generatorów znajdujących się w podgrafie (lub systemie).
`GetState()`	Zwraca obiekt[10] opisujący bilans mocy w podgrafie.
`PowerCost()`	Zwraca koszt generacji mocy dla aktualnego stanu systemu.
`ConstantCost()`	Zwraca wartość stałego (niezależnego od generowanej mocy) kosztu utrzymywania SEE w ruchu.
`Clear()`	Usuwa wszystkie informacje z bazy stanu systemu.

[9] A więc elementy zapisane w danej bazie stanu są indeksowane globalnymi identyfikatorami GUNN oraz GULN.

[10] Zwracana struktura `GraphState` została opisana w punkcie 5.

cd. tabeli 7

`CancelTransfers()`	Zeruje wartości przesyłów mocy w węzłach koordynacyjnych oraz w duplikatach węzłów koordynacyjnych.
`AddNode(N)`	Dodaje do bazy nowy węzeł.
`AddLine(L)`	Dodaje do bazy nową linię.
`RemoveNodeByLunn(lunn)`	Usuwa z bazy węzeł o podanym identyfikatorze LUNN.
`RemoveLineByLuln(luln)`	Usuwa z bazy linię o podanym identyfikatorze LULN.
`SwapNodesByLunn(l1,l2)`	Zamienia w bazie miejscami opisy węzłów o identyfikatorach LUNN *l1* i *l2*.
`SwapNodesByGunn(g1,g2)`	Zamienia w bazie miejscami opisy węzłów o identyfikatorach GUNN *g1* i *g2*.
`ConvertProducer-ToConsumerByLunn(lunn)`	Dokonuje zamiany węzła wytwórczego o podanym identyfikatorze LUNN w węzeł odbiorczy generujący stałą moc, niepodlegający regulacji.
`ConvertProducer-ToConsumerByGunn(gunn)`	Dokonuje zamiany węzła wytwórczego o podanym identyfikatorze GUNN w węzeł odbiorczy generujący stałą moc, niepodlegający regulacji.
`GunnExists(gunn)`	Zwraca `true`, jeżeli w bazie istnieje węzeł o podanym identyfikatorze GUNN.
`GulnExists(guln)`	Zwraca `true`, jeżeli w bazie istnieje linia o podanym identyfikatorze GULN.
`GetNodeByName(nazwa)`	Zwraca referencję do węzła o podanej nazwie.
`GetNodeByGunn(gunn)`	Zwraca referencję do węzła o podanym identyfikatorze GUNN.
`GetNodeByLunn(lunn)`	Zwraca referencję do węzła o podanym identyfikatorze LUNN.
`GetLineByGuln(guln)`	Zwraca referencję do linii o podanym identyfikatorze GULN.
`GetLineByLuln(luln)`	Zwraca referencję do linii o podanym identyfikatorze LULN.
`GetLineByLunns(l1,l2)`	Zwraca referencję do linii mającej początek i koniec w węzłach o podanych identyfikatorach LUNN *l1* i *l2*.
`LevelBy(dU)`	Zmienia o *dU* wartości modułów napięć we wszystkich węzłach znajdujących się w danym podgrafie.
`RotateBy(dF)`	Zmienia o *dF* wartości kątów fazowych we wszystkich węzłach znajdujących się w danym podgrafie.

7. PODSUMOWANIE

Struktura bazy danych SEE została zaprojektowana w sposób umożliwiający osiągnięcie wysokiej wydajności. W szczególności, zapewniono ścisłą relację między lokalnymi identyfikatorami elementów składowych podgrafu (LUNN i LULN) a indeksami elementów macierzy i wektorów opisujących układ równań rozpływu mocy.

Zastosowanie buforów TLB umożliwia – w razie potrzeby – szybkie wyszukiwanie elementów podgrafu o określonych identyfikatorach globalnych oraz elementów globalnej bazy stanu SEE na podstawie lokalnego identyfikatora elementu składowego jednego z podgrafów.

W czasie tworzenia oprogramowania nie stwierdzono negatywnego wpływu zastosowanej struktury danych na czas dostępu do informacji o elementach SEE. Testowano zarówno standardowe zestawy danych testowych IEEE o rozmiarze 9, 18, 30, 57, 118 i 300 węzłów, jak i większe zbiory danych o rozmiarach sięgających 3 000 węzłów.

Stworzenie abstrakcji zbioru danych opisujących podgraf SEE umożliwiło wykorzystanie tego samego kodu do opisu zarówno całego SEE, jak i poszczególnych podgrafów. Ponadto, dało to możliwość tworzenia *migawek* (ang. *snapshot*) stanu systemu, zapamiętywanych w buforze w pamięci lub w pliku dyskowym i eksportowanych do bazy danych.

Zaprezentowana w punkcie 6.1 definicja klasy pozwala stwierdzić, że nie zaimplementowano wszystkich możliwych odmian poszczególnych metod. Wynika to ze ścisłego związku między rozwojem klasy a faktycznymi potrzebami wyłaniającymi się w trakcie realizowania projektu badawczego. Silna hermetyzacja klasy pozwala jednak na dokonanie w przyszłości refaktoryzacji kodu w celu dostosowania jej do nowych zagadnień.

BIBLIOGRAFIA

1. Kremens Z., Sobierajski M.: Analiza systemów elektroenergetycznych. WNT, Warszawa 1996.
2. Kujszczyk Sz., Brociek S., Flisowski Z., Gryko J., Nazarko J., Zdun Z.: Elektroenergetyczne układy przesyłowe. WNT, Warszawa 1997.
3. Baron B., Pasierbek A., Kraszewski T., Połomski M., Sokół R.: Calculation coordination of subgraphs obtained in the decomposition process of power system graph. Międzynarodowa konferencja z podstaw elektrotechniki i teorii obwodów. Proc. of International Conference IC-SPETO, Ustroń 2008, p. 29-30.
4. Baron B., Kraszewski T., Pasierbek A., Połomski M., Sokół R.: Coordination of optimization results of power system subgraphs. Control of Power Systems, Słowacja 2008, p. 50.
5. Baron B., Pasierbek A., Kraszewski T., Połomski M., Sokół R.: Optymalizacja rozpływu mocy w systemie elektroenergetycznym z zastosowaniem koordynacji przetwarzania rozproszonego. Konferencja „Aktualne problemy w elektroenergetyce”, Jurata, 3-5 czerwca 2009, p. 11-20.
6. Baron B., Pasierbek A., Kraszewski T., Połomski M., Sokół R.: Graph partitioning algorithms for power system decomposition. Międzynarodowa konferencja z podstaw elektrotechniki i teorii obwodów. Proc. of International Conference IC-SPETO, Polska, Ustroń 2009, p. 145-146.

Recenzent: Prof. dr hab. inż. Wojciech Machczyński

Wpłynęło do Redakcji dnia 16 grudnia 2009 r.

Abstract

A power system can be modeled as a set of nodes of certain generation and consumption properties and lines connecting these nodes, optionally transforming voltage (containing a transformer). An efficient implementation of optimization algorithms requires a lean and efficient in-memory representation of such model. The representation should also be easy and straight-forward to use, as to minimize a number of errors and shorten development time.

Additionally, the representation should take into account a possibility of storing a temporary power system state in a separate set of variables and dividing the power system into separate sub-graphs, this being a necessary step for parallel and distributed computing.

The paper presents such an in-memory object-oriented power system model representation, written in the C++ programming language and created with the above properties in mind. The library contains C++ classes representing a power system node, power line, power system graph and sub-graph and a saved state of the power system. Application of the library has increased a level of abstraction of the OPF package (being developed by a team the Author is a member of) and allowed for a high level of dependability and functionality.