УДК 004.4:004.032.24

Гулаєва Н. М.

ПРО ОДИН ПІДХІД ДО ОРГАНІЗАЦІЇ РОБОТИ WAP-ШЛЮЗУ В БАГАТОПРОЦЕСОРНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Пропонується паралельний варіант процесу обробки WAP-шлюзом WML-сторінки. Як моделюючий засіб використовується розширена мережа Петрі.

Події останніх років показали, що впровадження нових технологій доступу до даних відбувається набагато швидше, ніж цього можна було очікувати. Інтеграція Internet та стільникового зв'язку стає дедалі щільнішою. Основною сполучною ланкою між Internet та стільниковим зв'язком є WAP-технологія.

Нагадаємо, що схема доступу абонента стільникового зв'язку до Internet-ресурсів включає три основні компоненти: WAP-браузер, WAP-шлюз та WAP-сервер (HTTP-сервер з WML-вмістом) (рис. 1).

Слабкою ланкою в ланцюгу WAP-браузер - WAP-шлюз - WAP-сервер на сьогодні є пересилка даних по бездротовій мережі, де швидкість передачі даних є вкрай низькою (в стандарті GSM не більше 9.6 Кб/с). Проте з переходом до радіостандартів третьої генерації (3G), де швидкість передачі даних сягатиме 2 Мб/с [1], проблема пересилання даних стає менш актуальною і на перший план виходить проблема забезпечення коректної роботи WAP-шлюзу, навантаження на який зростає із збільшенням кількості користувачів.

Огляд існуючих реалізацій WAP-шлюзів можна знайти у [2, 3]. В даній роботі пропонується паралельний варіант організації роботи WAP-шлюзу, зокрема паралельний варіант обробки WAP-шлюзом отриманої від WAP-сервера WML-сторінки. Як моделюючий засіб використовується запропонована в [4] розширена мережа Петрі.

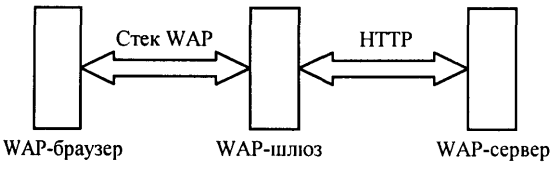

Рис. 1. Схема доступу абонента стільникового зв'язку до Internet-ресурсів

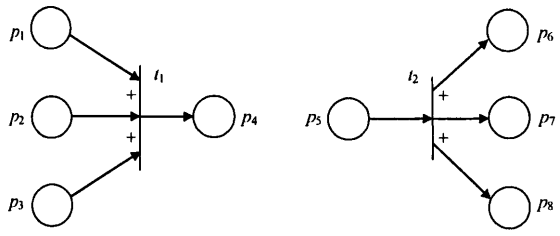

Рис. 2. Розширення класичної мережі Петрі: диз'юнктивна логіка

Нагадаємо, що в [4] класичну мережу Петрі було розширено операціями диз'юнктивної логіки та перемикачем; було також показано, що у випадку k-обмеженої розширеної мережі Петрі введені розширення лише підсилюють її описові можливості.

На рис. 2 зображено диз'юнктивну логіку (позначається знаком «+» на входах чи виходах переходу). Перехід t_1 допустимий, коли тільки одне з місць p_1, p_2 чи p_3 містить фішку. Після спрацювання переходу t_2 тільки одне з місць p_6, p_7 чи p_8 отримає фішку.

Перемикач (позначається ▷) представляє спеціальний тип вхідного місця, що визначає, яке з двох вихідних місць отримає фішку. На рис. 3 показано роботу переходу з перемикачем. Зауважимо, що наявність або відсутність фішок у перемикачі впливає тільки на розміщення фішки після виконання переходу, але не на те, щоб зробити перехід допустимим.

З переходами в розширеній мережі Петрі пов'яжемо оператори над даними. Для керуючих переходів диз'юнктивної логіки такі оператори діють як перемикачі-вирішувачі. Множина комірок пам'яті, з якої читають та/або в яку пишуть пов'язані з переходами оператори, представлена в моделі прямокутними вузлами з штриховими напрямленими дугами, що вказують на зв'язок між входами/виходами операторів.

У задачі обробки WML-сторінки виділимо такі підзадачі:

а) контроль вкладеності тегів;

б) перевірка відповідності структури документа набору правил DTD, який завантажується з окремого текстового файлу ([5]);

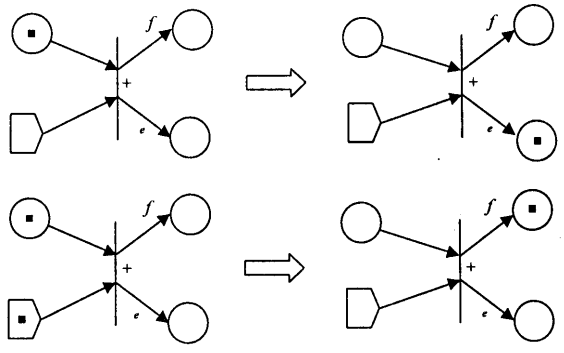

Рис. 3. Розширення класичної мережі Петрі: перемикач

в) генерація байт-коду.

Організуємо роботу WAP-шлюзу в чотири етапи.

1) За набором правил DTD будується дерево визначення типу документа. Вузол цього дерева містить ім'я тега, спеціальний символ, що визначає кількість можливих входжень тега до батьківського тега («1» - одне входження, «?» - нуль або одне входження, «*» - нуль або більше входжень, «+» - одне або більше входжень), імена атрибутів тега. Коренем дерева є вузол, що представляє тег <wml>; відношення «батько - син» визначається можливістю вкладеності тегів. Для зручності пересування по побудованій структурі в пам'яті можна також тримати вказівник на батька. Початкову частину дерева визначення типу документа, побудованого за правилами, визначеними специфікацією стандарту WAP 1.1, зображено на рис. 4.

2) Будується таблиця лексем з полями «Тип лексеми» / «Значення лексеми». Типами лексем є:

«StartTagName» - ім'я відкривального тега;

«EndTagName» - ім'я закривального тега;

«AttrName» - ім'я атрибута;

«AttrValue» - значення атрибута;

«Text» — текст;

«DONE» - документ прочитаний до кінця.

Зауважимо, що, за умови відповідності структури документа набору правил DTD, перший рядок таблиці лексем завжди містить «Тип лексеми»: «StartTagName», «Значення лексеми»: «wml»; останній рядок таблиці містить «Тип лексеми»: «DONE», «Значення лексеми».

Процедуру побудови таблиці лексем наведено в додатку 1.

3) Використовуючи дерево визначення типу документа та таблицю лексем, побудовані на етапах 1) та 2) відповідно, перевіряють вкладеність тегів та відповідність структури документа набору правил DTD. Процедуру перевірки наведено в додатку 2.

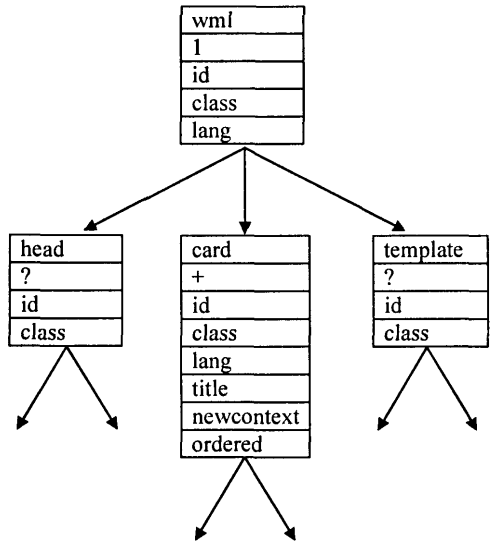

Рис. 4. Дерево визначення типу документа

4) За інформацією з таблиці лексем, побудованої на етапі 2), генерується байт-код, який і пересилається по бездротовій мережі.

Очевидно, що етапи 1) та 2) можна виконувати паралельно, оскільки вони не використовують спільних даних. Більше того, одночасно з 2) в магістральний спосіб можна виконувати ще й етап 4), а після закінчення побудови дерева визначення типу документа - також етап 3). Інший варіант - паралельне виконання етапів 1) та 2), а після закінчення роботи етапу 1) - етапів 2), 3), 4); при цьому етапи 2), 3) та 2), 4) виконуються в магістральний спосіб, оскільки вони використовують спільні дані - таблицю лексем, що генерується на етапі 2). Модель описаного способу організації обчислювального процесу представлена на рис. 5. Як бачимо, кожен крок побудови таблиці лексем (етап 2)) може ініціювати крок перевірки відповідності структури документа набору правил DTD (етап 3)) та крок генерації байт-коду (етап 4)); фішки, кожна з яких означає додавання чергового рядка до таблиці лексем, накопичуються у відповідних чергах. На рис. 6 показана детальніша модель етапу генерації байт-коду; деталізація інших етапів виконується аналогічно.

Зауважимо, що побудована мережа Петрі не є k-обмеженою в цілому (за рахунок можливого накопичення фішок у чергах між етапами). Проте, оскільки k-обмеженими є всі блоки мережі, можна ввести буфер для накопичення фішок між етапами; розмір цього буфера має визначатись експериментальне.

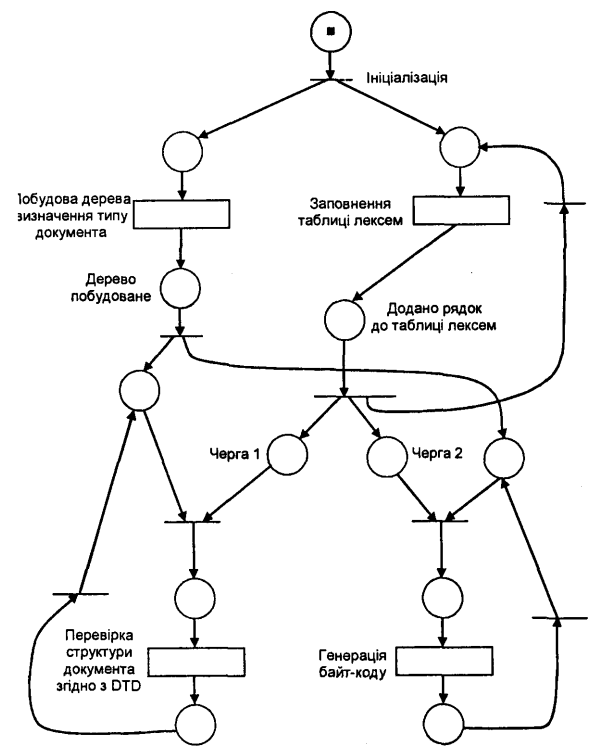

Рис. 5. Модель процесу обробки WML-сторінки

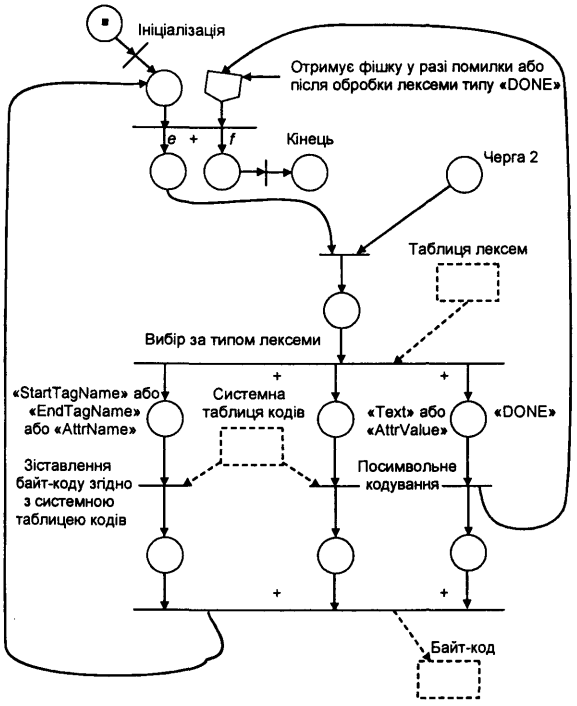

Рис. 6. Деталізована модель етапу генерації байт-коду

Додаток 1

Виділимо множину типів допоміжних лексем: «TagStart» («<»), «TagEnd» («>»), «EndTagStart» («</»), «Equal» («=»), «Word» (одне слово), «Word_Sequence» (послідовність слів). Нехай функція GetAuxLexemQ змінює значення глобальних змінних TypeAuxLex та ValueAuxLex - відповідно тип та значення зчитаної допоміжної лексеми.

Введемо глобальну змінну State, значеннями якої можуть бути «Error» (помилка), «Start» (початок обробки), «TagBegin» (було зчитано символ «<»), «InTagBegin» (було зчитано символ «<» та ім'я відкривального тега), «TagEnd» (було зчитано символи «</»), «InTagEnd» (було зчитано символи «</» та ім'я закривального тега), «OutOfTags» (після читання символу «>» не було зчитано жодного символу «<»), «AttrName» (зчитано ім'я атрибута). На початку роботи State = «Start».

Нехай функція InsertTable(TypeLex, ValueLex) заповнює черговий рядок таблиці лексем. Тоді процедура побудови таблиці лексем запишеться в такий спосіб:

```
procedure Lexical_Analysis ()
begin
  while (вхідний текст не прочитано) do
  begin
    GetAuxLexem ();
    Case State:
      «Start»:
        if TypeAuxLex = «TagStart» then
          State: = «TagBegin»
          else
            State: = «Error»;
      «TagBegin»:
        if TypeAuxLex = «Word» then begin
          InsertTable («StartTagName», ValueAuxLex);
          State: = «InTagBegin»
        end
        else
        State : = «Error»;
      «InTagBegin»:
        if TypeAuxLex = «Word» then
        begin
          InsertTable («AttrName», ValueAuxLex);
          GetAuxLexem ();
          if TypeAuxLex <> «Equal» then
          State: = «Error»
          else
          State: = «AttrName»
        end
      else begin
        if TypeAuxLex = «TagEnd» then
          State: = «OutOfTags»
        else
          State : = «Error»
      end;
      «AttrName»:
        if TypeAuxLex = «Word» then begin
          InsertTable («AttrValue», ValueAuxLex);
          State : = «InTagBegin»
        end
        else
        State: = «Error»;
      «OutOfTags»:
        if TypeAuxLex = «Word_Sequence» then
          InsertTable («Text», ValueAuxLex)
        else begin
          if TypeAuxLex = «EndTagStart» then
          State: = «TagEnd»
          else begin
            if TypeAuxLex = «TagStart» then
              State: = «TagBegin»
            else
            State: = «Error»
          end
        end
      «TagEnd»:
        if TypeAuxLex = «Word» then begin
          InsertTable («EndTagName», ValueAuxLex);
          State: = «InTagEnd»
        end
        else
          State: = «Error»;
      «InTagEnd»:
        if TypeAuxLex = «TagEnd» then
            State: = «OutOfTags»
          else
            State: = «Error»;
      «Error»:
        InsertTable («DONE», «»);
        STOP: ERROR;
    end case
  end while
  InsertTable («DONE», «»);
  if State = «Error» then
  STOP: ERROR;
end.
```

Додаток 2

Нехай NP - вказівник на поточний вузол дерева визначення типу документа. Через $NP^{\wedge}$ будемо позначати вузол, на який вказує NP. Нехай функція GetNext () зчитує в змінні LType та LValue значення полів з чергового рядка таблиці лексем. Тоді процедура перевірки вкладеності тегів та відповідності структури документа набору правил DTD запишеться в такий спосіб.

```
procedure check_Rules ()
begin
  getNext ();
  NP вказує на корінь дерева;
  if LType <> «StartTagName» or LValue <> «wml» then
    STOP: ERROR;
  while (LType <> «DONE») do
    begin
      case LType:
        «StartTagName»:
          if (серед синів NP^ є син з іменем LValue) then
            if (цього сина помічено «O») then
              STOP: ERROR
            else NP вказує на сина з іменем LValue
          else STOP: ERROR;
        «AttrName»:
          if (у вузлі NP^ немає атрибута
                                  з іменем LValue) then
            STOP: ERROR;
        «EndTagName»:
          if (LValue не є іменем вузла NP) then
            STOP: ERROR
          else begin
              у вузлі NP^ замінити «+» на «*»,
                                  «?» на «O», «1» на «O»;
              вказівник NP вказує на батька
            end
      end case
      GetNext ();
    end while
    if (вказівник NP вказує не на допоміжний вузол,
                    що виступає як батько кореня) then
      STOP: ERROR;
    if (в дереві є вузли з символами «+» чи «1») then
      STOP: ERROR;
end.
```


1. *Кравченко С.* На пути к третьему поколению // CHIP.-2001.-№2.-С. 86-92.
2. *Русеев С. Р.* WAP: технология и приложения- СПб.: БХВ-Петербург, 2001.- 432 с.
3. *СысойкинаМ.* Введение в WAP// Byte.-2001.-№ 6- С. 40-45.
4. *Baer J.-L., Ellis C. A.* Model, Design and Evaluation of a Compiler for a Parallel Processing Environment // IEEE Transactions on Software Engineering.- 1977.- V. SE-3.- № 6.- P. 394^105.
5. http://www.wapforum.org/DTD.

N. M. Gulayeva

ON ONE APPROACH TO WAP-GATEWAY FUNCTIONING ORGANIZATION IN THE MULTIPROCESSING ENVIRONMENT

A parallel version of WML-page processing by WAP-gateway is proposed. An extended Petri Net is used as a modeling tool.