У статті розглядаються моделі алгоритмічних конструкцій селекції прийняття рандеву мови програмування Ада, реалізовані за допомогою мереж Петрі

Ключові слова: мова програмування Ада, мережі Петрі, моделі паралельних алгоритмів, механізм рандеву

В статье рассматриваются модели алгоритмических конструкций селекции принятия рандеву языка программирования Ада, реализованые посредством сетей Петри

Ключевые слова: язык параллельного программирования Ада, сети Петри, модели параллельных алгоритмов, механизм рандеву

УДК 519.683.2, 519.179.2, 004.942

МЕРЕЖІ ПЕТРІ ЯК ЗАСІБ МОДЕЛЮВАННЯ МЕХАНІЗМУ РАНДЕВУ МОВИ АДА

А.М. Парнюк
Аспірант
Відділення гібридних моделюючих та управляючих систем в енергетиці
Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України
вул. Генерала Наумова, 15, м. Київ, Україна, 03164
Контактний тел.: 097-359-70-18
E-mail: oxo_@mial.ru

1. Вступ

Активний розвиток багатопроцесорних і багатоядерних обчислювальних систем, вимагає створення відповідних засобів тестування та верифікації паралельних програм, розроблених для них. Для цього необхідно використовувати математично точні методи для перевірки коректності роботи програм, такі як дедуктивний аналіз, імітаційне моделювання, тестування та перевірка на моделі. Останній метод, що використовує техніку модельного підходу Model Checking [1], є найбільш ефективним. Його основна ідея полягає в тому, що специфікації описуються формулами темпоральної логіки, а програма, що перевіряється, представляється у вигляді деякої моделі, яка побудована з використанням графоаналітичних засобів моделювання.

Переваги даного методу в тому, що він може бути реалізований досить ефективним алгоритмом, здатним працювати на обчислювальних машинах невисокого класу.

2. Виділення проблеми та постановка задачі

Основною проблемою в застосуванні цього методу є побудова абстрактної моделі, на якій виконується верифікація, причому вона повинна бути простішою, ніж система, що перевіряється, оскільки по суті це абстракція, в якій повинні бути відображені найбільш суттєві характеристики системи.

В загальному вигляді вирішення сформульованої проблеми є досить складним завданням і вимагає специфічних засобів для побудови й аналізу моделей паралельних алгоритмів. Основними вимогами до подібних засобів моделювання є алгоритмічна надійність та можливість опису асинхронного виконання паралельних гілок алгоритму з однозначним математичним описом їх функціонування. Цим критеріям повністю відповідає апарат мереж Петрі [2, 3] розроблений на основі абстрактних кінцевих автоматів.

3. Основна частина

Під час роботи паралельної програми можливі ситуації, коли кілька задач намагаються одержати доступ до одних і тих же даних і щоб уникнути конфліктних ситуацій і підтримати коректність роботи програми необхідна синхронізація доступу до поділюваних змінних. Забезпечення синхронізації являє собою основну проблему багатозадачності паралельного програмування, і називається взаємним виключенням.

Рішення цієї проблеми ґрунтується на тому, що в який-небудь момент часу право на доступ до поділюваної змінної надається тільки одній задачі. Для забезпечення синхронізації доступу використовуються спеціальні програмні (або апаратні) засоби: семафори, критичні секції, монітори й т.д. Як один з механізмів забезпечення надійного міжзадачного обміну даними й взаємної синхронізації роботи задач, Ада надає механізм рандеву.

Основна ідея механізму рандеву досить проста. Задачі мови Ада зв'язуються між собою за допомогою входів. Якщо одна задача здійснила звернення до входу й воно прийнято, то обидві задачі втрачають свою незалежність: вони встановлюють рандеву, і доти , поки рандеву діє, задачі синхронізовані.

4. Селекція прийняття рандеву

Нижче розглядаються шаблони, що реалізують різні форми оператора відбору (select), який використовується при прийнятті рандеву задачами. Призначення оператора відбору - надати користувачеві розвинені засоби програмування взаємодії задач. В загальному вигляді даний оператор має форму:

select;
альтернатива_відбору
-- в операторі може бути одна або декілька альтернатив відбору
or
альтернатива_відбору
or
альтернатива_відбору
...

Тут можна використати одну або декілька альтернатив відбору, що розділяються зарезервованим словом or. Альтернативи можуть бути відсутніми. Частина *else* необов'язкова. Альтернатива відбору може мати наступні різновиди:

when умова => оператор_прийому
послідовність_операторів
when умова => оператор_затримки
послідовність_операторів
when умова => terminate;

Необов'язковими частинами тут є: *"послідовність_операторів"* і *"when умова =>"* (умова відбору альтернативи) [4]. Загальна схема реалізації оператору вибіркового очікування зображена на рис. 1.

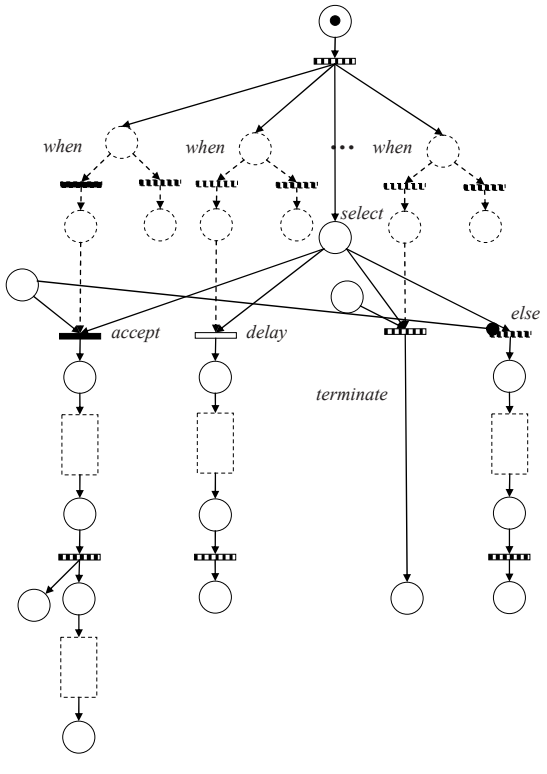


Рис. 1. Загальна схема оператору відбору в термінах мереж Петрі

Оскільки як вже було сказано "послідовність_операторів" і *"when умова =>"* (умова відбору альтернативи) являються необов'язковими частинам відповідні їм зображення конструкцій мережі виділено пунктиром. Якщо для альтернативи відсутнє зарезервоване слово *when* або ж якщо умова, що слідує за словом *when*, істинна, то відповідна альтернатива оператора *select* називається відкритою. В протилежному випадку вона називається закритою. Ці терміни корисні для розуміння роботи операторів селективного очікування. В термінах мереж Петрі це означає, що відкрита альтернатива є аналогом ситуації коли перехід готовий до спрацювання, якщо у вершині місця, що відповідає за виклик рандеву буде знаходитись мітка. При закритій альтернативі спрацювання переходу буде неможливим навіть при наявності мітки у вершині виклику рандеву. В операторі селективного очікування повинна бути принаймні одна альтернатива прийому з оператором *accept*.

Виконання оператора селективного очікування відбувається в такий спосіб. Спочатку в деякому довільному порядку обчислюються умови, що знаходяться за зарезервованими словами *when*. Одна з альтернатив відбору або *else*-частина, якщо тільки вона є, буде претендувати на виконання. Перевага буде віддано кожній з відкритих альтернатив з оператором прийому, що може встановити рандеву. Якщо кілька альтернатив задовольняє цій вимозі, то одна з них буде обрана випадковим чином. Вибір альтернативи з оператором прийому означає, що будуть виконані й послідовність операторів, що входять в оператор прийому (якщо вони існують), і послідовність операторів, що слідує за оператором прийому (якщо ця послідовність існує).

Якщо для жодної альтернативи прийому неможливо здійснити рандеву і якщо *else*-частина відсутня, то задача ввійде в стан очікування. Однак якщо є *else*-частина, то вона буде відібрана й виконана. Як видно зі схеми на рис. 1 *else*-перехід спрацює лише у випадку відсутності викликів рандеву, тобто якщо всі наявні вершини виклику рандеву будуть порожніми (це позначається дугою заперечення, що виходить з даної вершини до відповідного *else*-переходу).

У той час як задача буде перебувати в стані очікування, буде виконуватися відбір відкритої альтернативи затримки із *delay*. Це відбудеться, якщо задана затримка минула, а ніяка альтернатива прийому не могла бути відібрана. Якщо в декількох альтернативах затримки зазначена однакова тривалість, то буде відібрана випадковим чином одна із цих альтернатив.

Зрештою буде відібрана *terminate*-альтернатива, якщо всі залежні задачі від даної задачі-власника завершилися або ж перебувають у стані очікування, а в чергах до входів цих завдань немає жодного виклику. Якщо всі альтернативи закриті, а *else*-частина відсутня, то виникає виняткова ситуація [5].

Для кращого розуміння роботи конкретних елементів оператору вибіркового очікування *select* розглянемо частинні випадки, що ілюструють основні можливості і правила взаємодії задач.

4.1. Оператор прийому accept

Припустимо, що нам необхідна задача-сервер, що буде обслуговувати безліч задач-клієнтів. Припустимо також, що задача-сервер повинна мати не один, а безліч входів для надання різних сервісів задачам-клієнтам, а один із входів буде вказувати на необхідність завершення роботи задачі-сервера, тобто задача-сервер повинна виконуватися доти, поки не одержить явної команди на завершення своєї роботи [4].

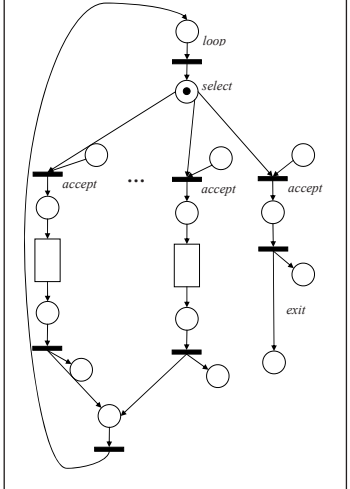
Розглянемо наступний приклад використання інструкції відбору в тілі задачі-сервера (табл. 1). У цьо-

му прикладі при виконанні інструкції відбору задача-сервер циклічно "опитує" свої входи на наявність виклику рандеву від задач-клієнтів (без блокування в стані очікування виклику рандеву на якому-небудь вході).

Опитування триває доти, поки не буде виявлений виклик рандеву на якому-небудь вході, що відповідає одній з перерахованих інструкцій прийняття рандеву. Після виявлення виклику виконується відповідна альтернатива (завершення обробки альтернативи приводить до завершення інструкції відбору).

Таблиця 1

```
task Server_Task is
   entry Serv1 [парам.];
   . . .
   entry ServN [парам.];
   entry Stop;
end task;
task body Server_Task is
begin
  loop
    select
     accept Serv1[парам.] do
       . . .
     end Serv1;
    or
      . . .
     accept ServN [парам.] do
       . . .
     end ServN;
    or
     accept Stop;
     exit ; --вихід із циклу
    -- і завершення завдання
    end select
  end loop;
end Server_Task;
```



Варто звернути увагу, що якщо в процесі опитування, виконуваного інструкцією відбору, одночасно з'являться два й більше виклики рандеву, то інструкція відбору вибере для обробки тільки один з викликів, що надійшли, причому правила вибору альтернативи для такого випадку не визначаються стандартом. Інакше кажучи, коли інструкція відбору поміщена в тіло циклу, з одночасною появою двох і більше викликів рандеву, інструкція відбору буде здійснювати обробку тільки одного виклику рандеву протягом одного "витка" циклу, а виклики рандеву, що надійшли одночасно, будуть поставлені в чергу порядок якої не визначається стандартом [5].

Розглянемо детальніше схему роботи мережі Петрі, що демонструє роботу цієї програми. Конструкція *select* моделюється за допомогою однієї вершини місця з міткою і n-ної кількості переходів, для яких вона являється вхідною. Подібна будова забезпечує спрацювання лише одного переходу, причому того для якого вершина місця *accept* міститиме мітку, що вказує на наявність виклику рандеву паралельною задачею. Якщо переходів готових до спрацювання буде декілька, то не важливо який із них буде активовано, так як в стандарті Ада явно не вказується який з викликів рандеву буде прийнятий.

У даному прикладі цікавість представляє альтернатива прийняття рандеву на вході Stop. У цьому випадку відбувається вихід з нескінченного циклу виконання інструкції відбору, що, у свою чергу, приводить до завершення роботи задачі-сервера. Якщо в процесі завершення роботи задачі-сервера надійде виклик рандеву на який-небудь із входів Service_1 -Service_N, то задача-клієнт, що викликає рандеву, швидше за все одержить виключення Tasking_Error. Недолік такого підходу полягає в тому, що завершення роботи задача-сервера вимагає явної вказівки виклику рандеву на вході Stop.

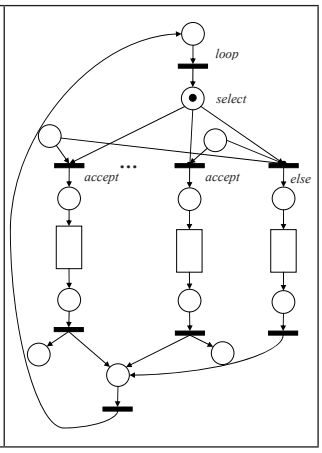
4.2. Оператор else

Може виникнути ситуація, коли необхідно, щоб у процесі очікування виклику рандеву від задач-клієнтів задача-сервер виконувала які-небудь додаткові дії.

Для цього можна використовувати інструкцію відбору, у якій замість альтернативи завершення роботи використовується розділ *else*. Використання такого варіанта інструкції відбору може мати такий вигляд:

Таблиця 2

```
loop
 select
  accept Serv1[пар.] do
       . . .
  end Serv1;
 or
     . . .
 or
  accept ServN [пар.] do
       . . .
  end Service_N;
 else
-- послідовність інструкцій,
-- що виконується якщо немає
-- жодного  виклику рандеву
 end select
end loop;
```



Як було сказано вище *else*-частина буде виконуватись, якщо в даний момент не має жодного виклику рандеву.

Для реалізації цього в мережах Петрі можна використати дуги заперечення, що дозволяють спрацювання переходу, якщо у відповідних вершинах місця немає мітки, тобто якщо всі *accept*-вершини порожні.

4.3. Альтернатива завершення - terminate

Щоб позбутися від необхідності явної вказівки виклику рандеву на вході Stop, можна використовувати інструкцію відбору, у якій вказується альтернатива завершення задачі (табл. 3).

У такому випадку альтернатива завершення задачі буде завершувати роботу задачі-сервера, коли відсутні виклики рандеву й відсутні задачі-клієнти, які потенційно здатні викликати рандеву із задачею-сервером. Таким чином, для завершення роботи задачі-сервера не потрібно явного виконання ніяких спеціальних дій. Варто також враховувати, що альтернатива завершення роботи задачі повинна вказуватися останньою в списку альтернатив інструкції відбору.

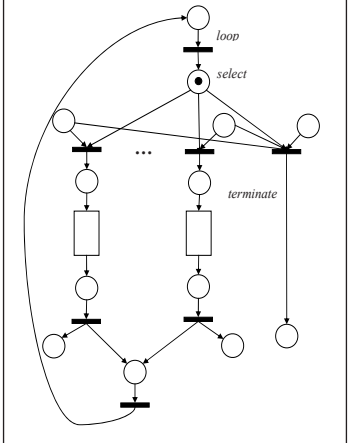
Реалізувати відсутність викликів рандеву досить просто шляхом використання відповідних дуг за-

перечення, що дозволяють спрацювання переходу, якщо всі accept-вершини порожні.

Таблиця 3

```
task Server_Task is
   entry Serv1 [пар.];
   . . .
   entry ServN [пар.];
end task;

task body Server_Task is
   . . .
begin
 loop
  select
   accept Serv1[пар.] do
      . . .
   end Serv1;
  or
    . . .
  or
   accept ServN[пар.] do
      . . .
   end ServN;
  or
   terminate;
--завершення роботи задачі
  end select
 end loop;
end Server_Task;
```



Що стосується відсутності задач-клієнтів, що можуть викликати рандеву, то подібна ситуація виходить за межі можливостей даних моделей Петрі і буде визначатись програмно.

В мережі це відображено за допомогою окремої вершини місця, яка являється вхідною до переходу terminate.

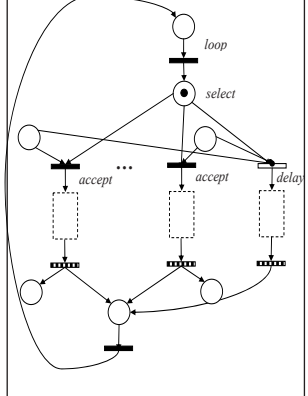
4.4. Затримка (таймаут) - delay

Ще одним різновидом інструкції відбору служить інструкція відбору, що використовує альтернативу таймаута (або затримки).

Така інструкція відбору може мати вигляд наведений в табл. 4.

Таблиця 4

```
loop
 select
  accept Serv1[пар.] do
     . . .
  end Serv1;
 or
      . . .
 or
  accept ServN [пар.] do
        . . .
  end Service_N;
 or
  delay 1.0;
   . . . - інструкції, що
виконуються коли немає
жодного виклику рандеву
протягом однієї секунди
 end select
end loop;
```



Альтернатива таймаута, зазначена в інструкцій відбору, дозволяє задачі-серверу виконувати певну послідовність дій, якщо протягом зазначеного інтервалу часу не надійшло жодного виклику рандеву.

Інтервал часу вказується в альтернативах таймаута аналогічно інтервалу часу в інструкціях затримки виконання.

При побудові моделі як і в операторі *else* відсутність викликів рандеву моделюється шляхом використання дуг заперечення від відповідних вершин місця. Затримка ж буде реалізовуватись програмно, шляхом асоціювання з переходом *delay* певного часу на протязі якого він не зможе спрацювати.

В одній інструкції відбору, допускається наявність більше однієї альтернативи таймаута. При цьому, буде оброблена та альтернатива таймаута, що має найменший інтервал часу.

4.5. Захисна умова - when

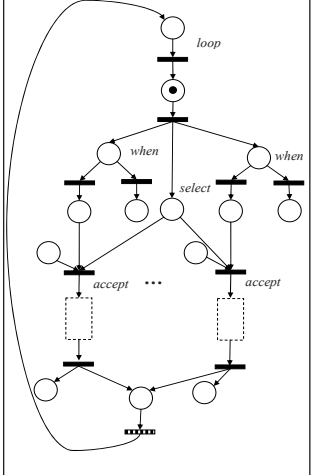
Ще однією особливістю інструкцій відбору є можливість вказівки додаткової перевірки якої-небудь умови для альтернатив прийняття рандеву, завершення роботи завдання й таймаута.

Для вказівки такої перевірки використовується конструкція виду: *"when умова =>"*, - умова, що перевіряється.

Як правило, таку перевірку називають захисною або охоронною (guard), а її використання демонструється в табл. 5.

Таблиця 5

```
declare
   Serv1_Count: Integer;
   . . .
   ServN_Count: Integer;
begin
 loop
  select
   when (Serv1_Count>0)=>
    accept Serv1[параметри] do
    . . .
    end Service_1;
  or
        . . .
  or
   when (ServN_Count>100) =>
    accept ServN[параметри] do
    . . .
    end Service_N;
end select
   end loop;
   . . .
end;
```



Обчислення значення логічного виразу здійснюється на початку виконання інструкції відбору. Якщо результат обчислення виразу *true*, то відповідна альтернатива має право бути обраною інструкцією відбору, якщо результат false, альтернатива обрана не буде, причому навіть у тому випадку, коли яка-небудь задача-клієнт здійснила виклик відповідного входу й чекає на обслуговування.

Варто також врахувати, що у випадку, коли перевірки умов використовуються для всіх альтернатив інструкції відбору, і результати перевірок всіх умов мають значення *false*, це приведе до порушення виключення Program_Error.

Для того щоб реалізувати можливість перевірки всіх умов перед початком відбору, до конструкцій при-

кладів, що розглядалися раніше додається додатковий перехід, що множить мітки у відповідні конструкції перевірки умов *when* (являють собою класичне розгалудження) і у вже відому вершину вибору *select*. Таким чином спрацює лише той із переходів, серед тих що слідують з вершини-select, для якого результат перевірки умови буде позитивним (що характеризується наявністю мітки у відповідній вершині конструкції розгалуження).

При черговому запуску структури мітка присутня лише у початковій вершині, інші мітки, що існували у вершинах місць сруктури (окрім вершин виклику рандеву) перед її запуском обнуляється.

5. Висновки

В межах даної роботи розглядався один з механізмів забезпечення надійного міжзадачного обміну даними й взаємної синхронізації роботи задач мови Ада – механізм рандеву. Рандеву має більш високий рівень абстракції в порівнянні з класичними методами синхронізації, наприклад такими, як семафорами оскільки він має засоби захисного блокування й таймаутів, а також засоби для виконання вибіркової перебудови черг клієнтів і аварійного завершення.

У рамках проведеної роботи були розроблені шаблони, що реалізують додаткові можливості механізму рандеву, шляхом використання різних елементів оператору відбору (select). Для кращого розуміння роботи конкретних елементів оператору селективного очікування, що визначають умови, при яких стає можливим рандеву з задачею, що викликається, були створені моделі, що демонструють частинні випадки роботи, таких елементів як: оператор прийому (accept), оператор (else), альтернатива завершення (terminate), затримка (delay) та захисна умова (when), що розширюють можливості і правила взаємодії задач.

Література

1. Карпов, Ю.Г. Model Checking. Верификация параллельных и распределённых программных систем. [Текст] / Ю.Г. Карпов. – СПб.: БХВ-Петербург, 2010. – 560 с. – ISBN 978-5-9775-0404-1.
2. Питерсон, Дж. Теория сетей Петри и моделирование систем. [Текст] / Дж. Питерсон. – М.: Мир, 1984.–264 с.
3. Кузьмук, В.В. Модифицированные сети Петри и устройства моделирования параллельных процессов: монография / В. В. Кузьмук, О. А. Супруненко. – К. : Маклаут, 2010. – 252 с. – ISBN 978-966-2200-07-2.
4. Гавва А. «Адское» программирование. Ada-95. Компилятор GNAT. [Електронний документ]. Режим доступу: http://ada-ru.org/V-0.4w/index.html. Перевірено: 18.07.2012.
5. Ada Reference Manual ISO/IEC 8652:2007(E) Ed. 3. [Електронний документ]. Режим доступу: http://www.adapower.com/rm-95/index.html. Перевірено: 18.07.2012.

Abstract

Investigating and analysis of processes interaction in parallel Ada programs by creation of models in terms of Petri Nets is considered in this article. The topicality of the research is stipulated by the development of multiprocessing and multinuclear systems and necessity of parallel software analysis created for them. Currently, there are several technologies for development of parallel programs. Ada programming language was chosen because it has its own built-in means for the development of parallel software. The subject matter of the research is the formation of the algorithmic constructions which describes the mechanism rendezvous (by detailed modelling of operator "selective accept") on the basis of Petri Nets. The practical application of the research is based on the possible usage of the obtained data and the main conclusions for verification program code and construction of more effective programs

Keywords: Ada language, Petri nets, models of parallel algorithms, rendezvous