Pengnyahralat bagi Bahasa-bahasa Perlaksanaan Acara Tunggal dan Serempak

oleh
Faizah bt Ahmad

Jabatan Kejuruteraan Perisian
Fakulti Sains Komputer dan Sistem Maklumat
Universiti Teknologi Malaysia
45100 Kuala Lumpur

Abstrak

Kertaskerja ini memuatkan kajian mengenai perkembangan-perkembangan yang terdapat pada pelbagai generasi perisian pengnyahralat. Ini termasuk pengnyahralat bagi bahasa himpunan hinggalah kepada bahasa logik. Khususnya, kajian ini cuba meneliti pendekatan-pendekatan yang telah digunakan dalam merekebentuk pengnyahralat bagi bahasa-bahasa Pascal, Prolog dan Ada. Bahasa-bahasa Pascal dan Prolog mendokong perlaksanaan acara tunggal sementara Ada membolehkan perlaksanaan acara serempak. Bahasa Pascal merupakan bahasa berprosidur yang telah lama bertahan dan merupakan salah satu bahasa yang kerap diajar di tahap awal kursus-kursus sains komputer. Bahasa Prolog, bahasa pengaturcaraan berasaskan logik, mempunyai kegunaan yang meluas di dalam kepintaran buatan. Pemprosesan acara serempak yang didokong oleh Ada pula merupakan suatu komponen penting bagi sistem-sistem terbenam atau pun sistem masa nyata.

Katakunci: pengnyahralat, kejuruteraan perisian, pengujian simbolik, pengujian berpandukan laluan, pembantu penumpu ralat, model kotak, acara serempak.

Abstract

This article attempts to identify the development of various generation of debugging software. This includes debuggers for assembly languages up to and including logic languages. In particular, this research attempts to investigate the approaches that have already been adopted for the debuggers of Pascal, Prolog and Ada. Pascal and Prolog are languages that support single event execution while Ada enable execution of concurrent events. The Pascal language which is a procedural language has been around for quite some time and is often taught to students at the intoductory level of science computer courses. The Prolog language which is logic-based has a widespread application in artificial intelligence. Processing of concurrent event which is supported by Ada is an essential component of embedded systems, otherwise referred to as real-time systems.

1.0 Pendahuluan

Mengnyahralat merupakan suatu fasa yang penting di dalam pembangunan sesuatu perisian. Secara tak langsung seorang pengaturcara perlu mahir di dalam mengnyahralat. Dari segi fungsi, proses mengnyahralat terletak di tahap pembangunan sistem, di mana perisian dibangunkan bagi algoritma-algoritma yang telah dikenalpasti sesuai bagi permasaalahan yang sedang diselesaikan. Secara khusus,

Kertaskerja ini disunting oleh: En Mohd Daud Kasmuni dan Pn Noriah Idris.

penghyahralatan adalah sebahagian daripada proses pengujian perisian. Pengujian perisian adalah proses pengumpulan dan penaksiran bukti-bukti kesesuaian perisian bagi satu-satu operasi. Apabila pengujian digunakan bagi mengasingkan punca kecacatan yang disaksikan di dalam periakuan sesuatu aturcara maka ia digelar pengujian diagnostik atau mengnyahralat. Sekiranya pengujian digunakan bagi menilai kegunaan masa, ruang dan lain-lain bahan saperti keupayaan saluran input-output, ianya digelar pula sebagai pengujian keberkesanan atau pun *performance evaluation*. Bagi kedua-dua takrifan, terdapat tiga ciri yang sejagat iaitu:

i. aturcara dilaksanakan dengan input yang dipilih yang dipanggil kes-uji;

ii. pelaksanaan dilakukan di dalam persekitaran di mana ciri-ciri aturcara diperhatikan atau dikawal;

iii. hasil perlaksanaan kes-uji digunakan bagi membuat kesimpulan mengenai sifat-sifat aturcara yang sedang diuji.

Di satu sudut yang lain, proses mengnyahralat juga boleh dianggap selari dengan proses pengujian kes-uji di mana kes-uji yang tepat akan menyerlahkan berbagai kelemahan (baik yang dijangka dan yang tidak disangka-sangka) pada perisian tersebut dan sekaligus memerlukan perisian diperbaiki dan dinyahralat semula. Di samping itu, mengnyahralat juga dilaksanakan oleh pengguna ke atas perisian yang telah sempurna dibina (samada perisian dibina sendiri atau pun dibeli). Keperluan ini mungkin timbul kerana perlunya perisian diperkembangkan bagi menampung applikasi semasa.

Seperti yang kita sedia maklum, kesilapan-kesilapan atau *bugs* dalam aturcara terdiri daripada dua jenis: sintaksis dan logik. Kesilapan jenis pertama dapat dikesan dengan mudah oleh pengkompil atau pentafsir bahasa yang digunakan. Kesilapan jenis kedua bersifat tersembunyi dan memerlukan kemahiran untuk mengesannya. Antara kesilapan-kesilapan jenis ini adalah kesilapan pada penggunaan sintaks bahasa saperti memanggil subrutin atau fungsi (terutama sekali apabila kita mengguna semula rutin-rutin yang pernah diuji atau pun rutin-rutin yang terdapat pada buku teks, umpamanya pembolehubah ditakrifkan setempat sedangkan ia sepatutnya sejagat), kesilapan pada penggunaan struktur data sedia ada saperti penunjuk, kesilapan yang berpunca daripada kemudahan-kemudahan yang berbeza pada versi-versi bahasa yang digunakan (misalnya, kemudahan memproses fail) dan juga tidak ketinggalan kesilapan *transcription* (iaitu kesilapan pengaturcara semasa menyalin/menaip aturcara ke dalam komputer). Walau bagaimana pun, dengan adanya bantuan pengnyahralat kita dapat meringankan tugas pengaturcara menyungkil kesilapan-kesilapan ini. Dengan ini, kertaskerja ini bertujuan mengenalpasti beberapa pendekatan yang telah digunakan dalam merekabentuk pengnyahralat sehingga kini.

1.1 Perkembangan perisian pengnyahralat

Sejak dahulu lagi tahap mengnyahralat merupakan langkah yang sering merumitkan kerja-kerja melaksanakan aturcara. Ini timbul daripada sifat mengnyahralat itu sendiri yang secara tradisinya *adhoc* atau pun tidak berancang dan tiada mempunyai garis panduan yang piawai. Kebanyakan kajian di dalam kejuruteraan perisian tertumpu kepada tahap yang terdahulu daripada mengnyahralat terutamanya tahap rekabentuk aturcara dan pemilihan data. Kajian di tahap rekabentuk misalnya telah menghasilkan berbagai pendekatan saperti rekabentuk atas-bawah, kaedah pemecahan Parnas, rekabentuk berorientasikan objek dan sebagainya.

Keadaannya adalah berbeza bagi tahap perlaksanaan aturcara. Seandainya sesuatu aturcara yang agak besar mempunyai ralat, maka kerapkali pengaturcara A akan merujuk kepada pengaturcara yang lebih kanan. Atau pun, pengaturcara A akan berhempas-pulas seorang diri bagi mengenalpasti di mana terletaknya ralat itu sehingga kadangkala memakan masa berminggu-minggu. Ini bukanlah kerana pengaturcara A tidak pintar tetapi adalah kerana proses mengnyahralat tradisi bersifat *hit-and-miss* iaitu tekaan. Berdasarkan pendekatan tradisi ini, tembereng aturcara yang disyaki akan dikenalpasti dan pengaturcara akan menumpukan pengawasilapan kepada bahagian-bahagian ini. Bayangkanlah, tentu sekali aturcara yang besar mempunyai banyak tembereng-tembereng yang disyaki mempunyai ralat dan kemungkinan juga setiap tembereng ini juga agak panjang. Ini bermakna di dalam satu

tembereng itu saja terdapat lebih daripada satu ralat atau pun punca ralat. Maka sebarang carian ralat secara manual tidak dapat mengecam kesemua ralat yang ada pada tembereng sekaligus. Di samping itu, oleh kerana pada dasarnya salah satu punca ralat adalah nilai-nilai pembolehubah yang tidak tepat, maka perlulah pengaturcara menguji setiap pembolehubah pada satu-satu tembereng. Ini merupakan satu tugas yang rumit dan memakan masa yang lama.

Sejak pertengahan dekad ke 80 [Gould, '75] logik bagi pengnyahralat telah giat dikaji dan penyelidikan telah ditumpukan kepada perlakuan aturcara dan mengnyahralat logik aturcara. Usaha-usaha ini telah menghasilkan berbagai pendekatan baik bagi bahasa peringkat rendah atau bahasa himpunan hinggalah ke tahap bahasa berfungsi dan logik. Di peringkat bahasa himpunan misalnya, format-format suruhan pengnyahralat telah diperbaiki dengan adanya ciri-ciri seperti mengnyahralat berdasarkan frasa syarat di mana pengaturcara akan meminta pengnyahralat mengenalpasti arahan-arahan aturcara yang tidak menepati syarat yang diberi. Sebagai contoh, apabila daftar A atau pun lokasi ingatan tidak mengandungi nilai tertentu atau mengandungi nilai yang melebihi julat tertentu.

Dengan itu kita dapat mengenalpasti bahagian-bahagian aturcara yang mempunyai nilai-nilai data yang menyeleweng dan seterusnya mengecilkan skop carian ralat dengan lebih cepat. Kaedah ini dipanggil pengnyahralatan berdasarkan frasa [Fairley,'79] dan tergulung di dalam aliran pengujian simbolik. Di samping itu terdapat juga pengnyahralat yang dapat mempamerkan persekitaran sistem bagi sesuatu perlaksanaan atau pun disebut juga *display of run-time environment*. Pendekatan ini telah digunakan di dalam pembinaan pengnyahralat bagi sistem pengajaran bahasa himpunan, LILG [Faizah, '87]. Pameran dipaparkan bagi titik henti tertentu atau pun bagi setiap arahan bahasa himpunan. Pameran terdiri daripada komponen-komponen saperti daftar-daftar dan timbunan serta tetingkap bagi arahan setempat.

Bagi bahasa-bahasa berprosedur pula perkembangan dapat dilihat pada pengnyahralat masing-masing di mana perisian-perisian ini telah diterapkan sedikit kecerdikan. Maka pendekatan tekaan telah mula digantikan dengan sistem yang mampu membantu pengguna (pengaturcara) menumpukan kepada bahagian yang mengandungi ralat. Penyelidikan ke arah ini telah bermula dengan pendekatan instrumentasi aturcara secara automatik di mana pengnyahralat berkemampuan menyelitkan mesej di bahagian-bahagian yang menentukan aliran aturcara [Huang, '80]; umpamanya di badan THEN dan ELSE bagi arahan pengujian IF, atau pun di awal sesuatu gelung dan fungsi. Apabila aturcara yang telah dilengkapi mesej ini dilaksanakan, mesej-mesej yang dipamerkan pada output dapat memberikan gambaran perlakuan aturcara. Kaedah instrumentasi ini tergulung di dalam aliran pengujian berpandukan laluan. Kaedah ini juga sering kita gunakan tanpa kita sedari iaitu apabila kita menggemakan semula output (*echoing of output*) di paparan dan ketika menggunakan *stubs* bagi mengenalpasti modul-modul yang telah dimasukki. Seterusnya, kita akan mengkaji pendekatan yang digunakan pada pengnyahralat-pengnyahralat bagi bahasa Pascal, Prolog dan Ada. Pascal dan Prolog merupakan bahasa yang mendokong perlaksanaan aturcara acara tunggal sementara Ada membolehkan perlaksanaan aturcara acara serempak. Di samping itu bahasa Pascal merupakan bahasa berprosedur yang telah lama bertahan dan Prolog pula adalah salah satu bahasa kepintaran buatan yang mempunyai banyak applikasi. Bahasa Ada pula sesuai digunakan bagi sistem-sistem perisian yang besar seperti kawalan radar dan kilang-kilang berautomasi. Kita tinjau dahulu takrifan pengaturcaraan serempak agar dapat memanfaatkan perbincangan seterusnya.

1.2 Pengaturcaraan Acara Serempak dan Ada

Konsep acara serempak telah diterapkan dalam Ada bagi membolehkan pengaturcaraan sistem masa nyata. Sebelum bahasa Ada dicipta, sistem-sistem masa nyata yang menggunakan bahasa-bahasa pengaturcaraan lain saperti Fortran terpaksa menggunakan teknik prapemproses(S-FORTRAN) atau pun teknik perlanjutan atau *extension* (FORTRAN-77) bagi mengendalikan peralatan-peralatan yang membentuk komponen-komponen lain sistem tersebut. Kini setelah kajian yang meluas dilakukan terhadap sistem-sistem perisian yang besar, bahasa Ada pun dicipta dan dilengkapi dengan ciri-ciri pengaturcaraan serempak. Model yang digunakan bagi pengaturcaraan acara serempak dalam Ada ini dipanggil *task*. Turut disertakan bersama model *task* ini adalah mekanisma bagi kawalan acara serempak seperti saling menyingkirkan (mutual exclusion) dan sinkronisasi proses, yang mana kedu-duanya merupakan masalah-masalah utama di dalam pemprosesan acara serempak.

1.2.1 Model Acara Serempak -- *Task*

Task adalah ciri bahasa Ada bagi pemerihalan acara serempak samada serentak atau selari. Perlaksanaan acara serempak ini mugkin sebenar (menggunakan dua atau lebih pemproses) atau pun seakan-akan (*apparent* iaitu menggunakan perlaksanaan-perlaksanaan tunggal yang diselang-selikan pada satu pemproses). Konsep acara serempak ini mungkin dapat dijelaskan oleh masalah berikut.

Suatu bahagian perakaunan syarikat ditugaskan melaksanakan tugas-tugas:

i. membuat pesanan barangan
ii. menjelaskan pembayaran
iii.menyediakan invois

Jika syarikat mempunyai satu akauntan sahaja, maka aturcara Ada bagi sistem keseluruhannya akan berbentuk

```
procedure Akaun is
begin

Pesan_Barangan;
Bayar_Bil;
Sediakan_Invois;

end Akaun;
```

Sebaliknya jika syarikat mempunyai tiga akauntan, maka ketiga-tiga tugas tadi dapat dilaksanakan selari seperti yang digambarkan oleh aturcara Ada berikut.

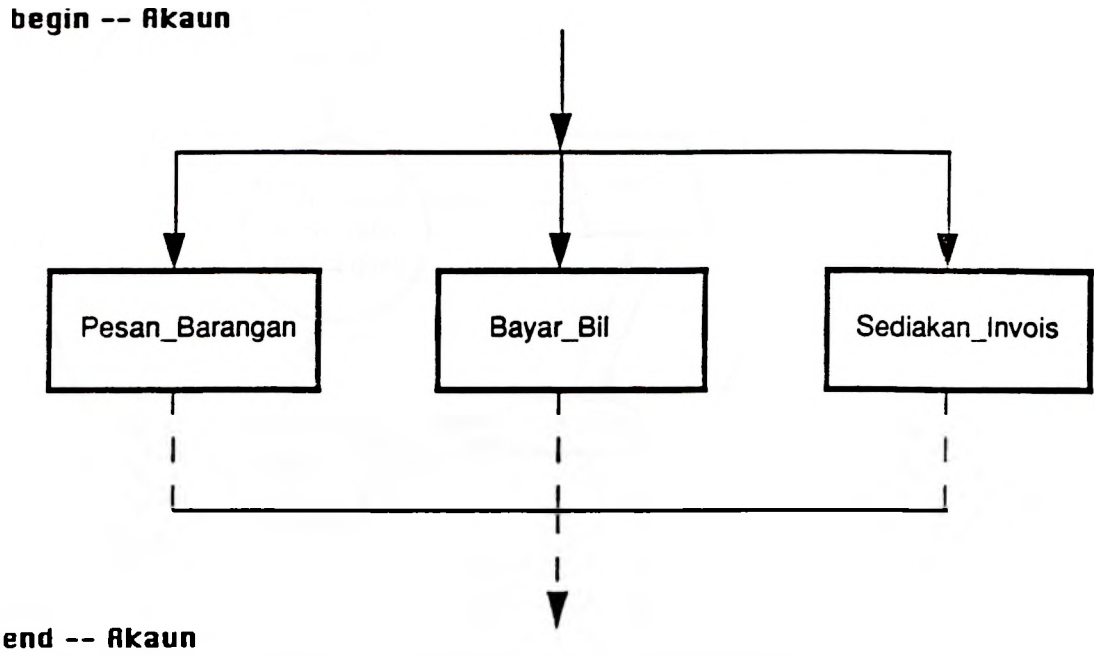
```
procedure Akaun is

   task Pesan;
   task body Pesan is
   begin
       Pesan_Barangan;
    end Pesan;

   task Bayar;
   task body Bayar is
   begin
      Bayar_Bil;
    end Bayar;

begin
    Sediakan_Invois;
end Akaun;
```

Task Pesan dan task Bayar merupakan pengisytiharan nama *task* serta pemerihalan antara muka *task* berkenaan dengan *task-task* lain (dalam kedua-dua kes ini tiada antara muka yang terlibat). **Procedure Akaun** yang merupakan bapa kepada *task-task* lain mengambarkan fungsi invois diuntukkan kepada ketua akauntan dan fungsi-fungsi memesan barangan dan membayar bil diuntukkan kepada pembantu-pembantunya. Perhatikan **task Pesan** dan **task Bayar** diisytiharkan dibahagian pengisytiharan **procedure Akaun** (prosidur bapa). Hanya apabila perlaksanaan sampai kepada **begin** bagi **procedure Akaun** barulah *task-task* diaktifkan secara selari. Ini bermakna, badan prosidur bapa diaktifkan dan dilaksanakan selari dengan *task-task* lain.

Kita juga boleh mengisytiharkan ketiga-tiga proses sebagai *task*. Bagi cara ini, badan prosedur **Akaun** akan hanya mengandungi pernyataan *null*. Walau bagaimana pun kesan kedua-dua cara ini ialah terdapat tiga aliran kawalan : ketiga-tiga proses Pesan_Barangan, Bayar_Bil dan Sediakan_Invois berjalan serentak seperti yang digambarkan oleh Gambarajah A.

Setiap *task* akan tamat secara normal apabila ia tamat perlaksanaan. Namun demikian prosidur bapa tidak boleh ditinggalkan sehinggalah ketiga-tiga *task-task* anak tamat. Syarat ini penting kerana ia bermaksud *task-task* bukanlah agen-agen yang bebas sepenuhnya Sebaliknya *task-task* terkandung di

dalam unit-unit yang lain dan bergantung kepada unit tersebut. Syarat ini juga menjamin sumber-sumber yang diperlukan oleh sesuatu *task*, yang kemungkinan besar diisytiharkan di dalam unit bapa, berada di dalam skopnya sepanjang hayat *task* berkenaan.



Gambarajah A: Tiga proses berjalan serentak

1.2.2 Koordinasi proses serempak

Dua masalah utama bagi pemprosesan acara serempak, saling menyingkirkan dan sinkronisasi, telah menghasilkan beberapa penyelesaian. Penyelesaian tradisi menggunakan suatu eksekutif masa nyata yang khusus dan berdasarkan beberapa *task* dilaksanakan pada satu pemproses. Eksekutif adalah suatu perisian yang mengambilkira keadaan sistem semasa dan memutuskan *task* yang mana akan dilaksanakan seterusnya. *Task-task* akan melalui salah satu keadan berikut:

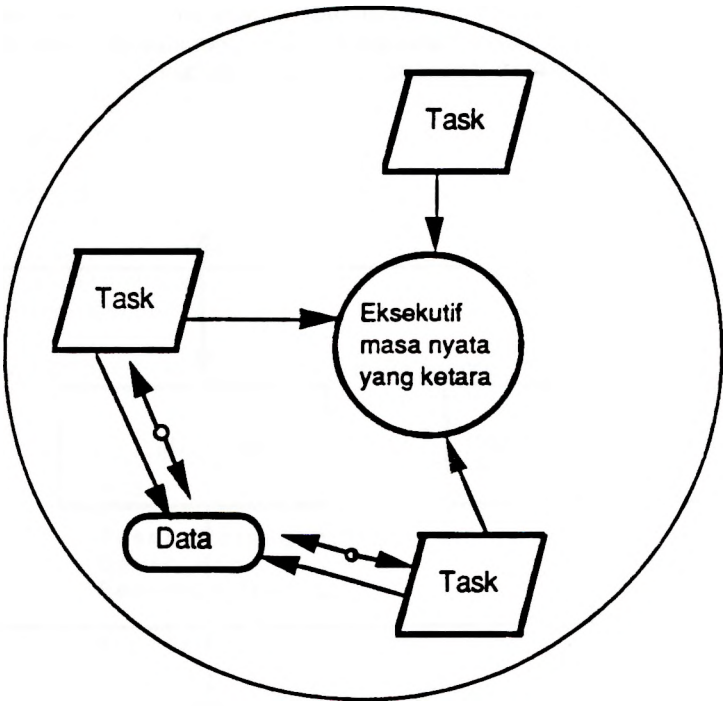
i. dilaksanakan atau *running*,
ii. disekat perlaksanaan atau *blocked* iaitu ia tidak mempunyai sumber-sumber yang membolehkan perlaksanaan,
iii. bersedia atau *ready* iaitu ia mempunyai sumber yang mencukupi tetapi sedang menunggu dilaksanakan.

Hubungkait di antara *task-task* ini diberikan oleh gambarajah di dalam Lampiran I. *Task-tasks* mungkin menggunakan Executive Service Requests (ESRs) bagi menyelaraskan capaian ke atas data atau pun perisian Eksekutif itu mungkin mempunyai penimbal dalaman bagi data seperti yang digambarkan oleh graf struktur pada Gambarajah B1.

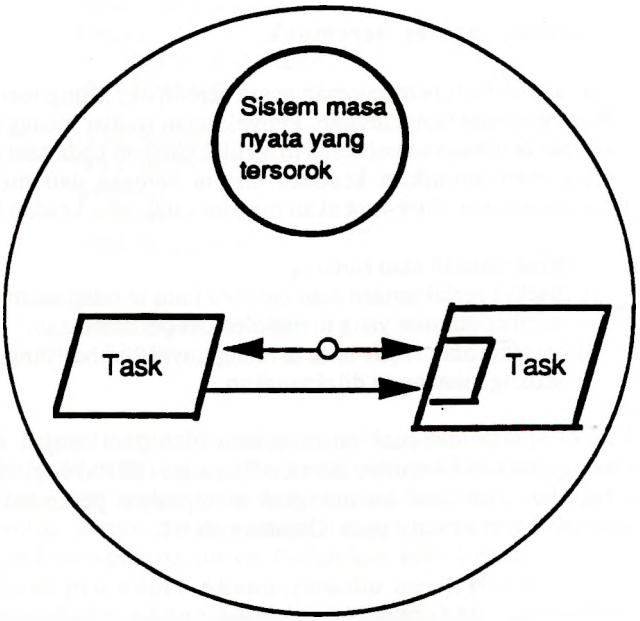
1.2.3 Rendezvous dan nantian berpilih

A) Takrifan rendezvous

Bagi Ada penyelesaian bagi masaalah pemprosesan acara serempak adalah berbeza. *Task-task* Ada dianggap sebagai proses-proses serempak yang tidak sinkronas dan setiap *task* dilaksanakan pada pemproses yang berasingan. Ciri Ada yang menyeluruh bagi perhubungan antara *task-task* dipanggil *rendezvous* dan pertama kalinya digunakan di dalam suatu sistem masa nyata. Ciri ini disemadikan di dalam bahasa Ada dan dengan itu Ada tidak memerlukan eksekutif masa nyata. Sebaliknya terdapat suatu sistem masa perlaksanaan yang tersorok dan merupakan asas kepada model pengendalian *task* Ada seperti yang digambarkan pada Gambarajah B2.



Gambarajah B1: Bahasa pengaturcaraan tradisi dan eksekutif masa nyata[1]



Gambarajah B2: Rendezvous bahasa Ada[2]

i. 1
ms 7, [Shumate, '88].

ii. 2
ms 8, [Shumate, '88].

Selain daripada sinkronisasi *task*, *renaezvous* juga memasukan saling menyingkirkan terjamin. Keperluan saling menyingkirkan umbul kerana *task-task* yang berkongsi data dan perlu mencapai data tersebut secara serentak akan menimbulkan pengnitungan atau pemprosesan yang tidak tepat. Dengan itu task-task perlu dihalang daripada mencapai data serentak. Ini bermakna jika arahan A1 terdapat di dalam task P1 dan arahan A2 terdapat di dalam P2, dan P1 dan P2 ingin melaksanakan A1 dan A2 serentak bagi mencapai data yang sama, kita perlu memastikan hanya salah satu akan berjaya mancapai data tersebut. Dengan itu jika task P1 yang terlebih dahulu mendapat kebenaran mencapai data tersebut, ia akan dibenarkan meneruskan perlaksanaan; sementara itu task P2 perlu menghalang perlaksanaannya (*block execution*). P2 hanya akan meneruskan perlaksanaan setelah P1 tamat melaksanakan A1 dan data bebas dicapai oleh P2. Keadaan ini dipanggil saling menyingkirkan. Terdapat beberapa pendekatan tradisi bagi memastikan saling menyingkirkan terjamin iaitu: penggunaan nilai semafor, kawasan kritikal dan penggunaan aturcara monitor.

Pendekatan Ada, *rendezvous*, berasaskan model *Communicating Sequential Proceses* (*CSP*) yang diperkembangkan oleh Hoare [Shumate, 1988]. Ia membolehkan perhubungan antara proses dengan menggunakan mekanisam yang membolehkan data ditukar-ganti antara dua proses pada satu detik sinkronisasi. Ini memerlukan mesej-mesej dihantar antara proses-proses. (Mekanisma yang sama digunakan oleh eksekutif masa-nyata tetapi adalah kurang rumit.) Arahan-arahan di dalam *task-task* yang terlibat bagi sesuatu rendezvous dilaksanakan sebagai kawasan kritikal. Ada juga menyediakan antara muka dengan hadwer melalui penggunaan pengawal-pengawai sampukan.

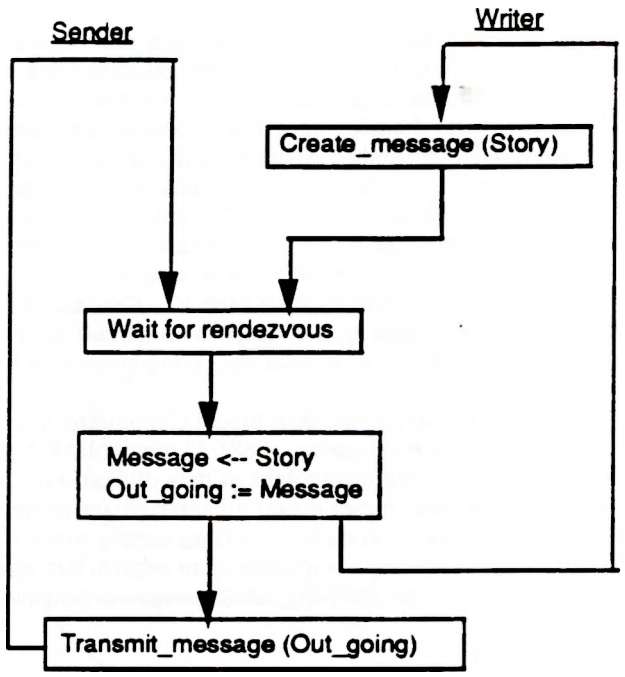
B) Nantian berpilih

Nantian berpilih atau pun *selective wait* adalah mekanisma bagi *task* yang dipanggil memilih untuk menerima salah satu panggilan dari *task-task* yang memanggil. Ini dicapai menggunakan arahan **select**. Berdasarkan interaksi di antara dua *task* berikut, *task* Sender dan Writer,

```
task sender is
  entry Take_message
        (Message : in Message_format);
end Sender;

task body Sender is
  Out-going : Message_format;

  procedure Transmit_Message
        (Message : Message_format) is
  .....
  end Transmit_Message;

begin
  loop
    accept Take_message
          (Message : Message_format) do
       Out_going := Message;
    end Take_message;

     Transmit_message (Out_going);
  end loop;
end Sender;
```

```
task Writer;
task body Writer is
   Story  : Message_format;

   procedure Create_message
      (message: out Message_format) is
   .....
   end Create_message;
begin
   loop
      Create_message (Story);
    Sender.Take_message (Story);
   end loop;

end Writer;
```

yang dicirikan alirannya oleh Gambarajah C, di awal perlaksanaan Sender perlu menunggu Writer menyiapkan Create_Message. Setelah itu kedua-duanya akan *rendezvous*, diikuti oleh perlaksanaan selari oleh Transmit_message dan Create_message. Bagi *rendezvous* seterusnya, *task* yang selesai dahulu perlu menunggu *task* yang satu lagi selesai sebelum kedua-duanya dapat *rendezvous*. Masa penungguan ini perlu dikurangkan sekiranya pemprosesan yang optima ingin dicapai. Ini boleh dicapai dengan *task* pengendali buffer seperti yang diperihalkan berikut. Di masa yang sama *task*



Gambarajah C : Interaksi Sender/Writer [3]

Unique_message ini menerima panggilan samada bagi Store_message atau Retrieve_Message. Keadaan ini dipanggil nantian berpilih atau selective wait.

```
task Unique_message is
    entry Store_Message          (Massage : in Message_format);
    entry Retrieve_Message       (Message : out Message_format);
end Unique_message;

Task body Unique_message is
   Hold               : Message_Format;
    New_message : Boolean := False;

begin
       accept Store_message (Message: in Message_format) do
       Hold := Message;
    end Store_message;
loop
   select
        accept Store_message (Message : in Message_format) do
          Hold := Message;
       end Store_message;
       New_message := True;
  or
    when New_message  =>
          accept Retrieve_message (Message: out Message_Format) do
      Message := Hold;
       end Retrieve_message;
      New_message := False;
```

[3] ms 43, [Shumate, '88].

```
    end select;
end loop;
end Unique_message;
```

Ciri nantian berpilih juga boleh digunakan bagi menghadkan tempoh bagi sesuatu *task* menunggu *rendezvous*. Kaedah ini dipanggil *timed selective wait* dan boleh dicapai dengan cara berikut:

```
select
   accept Store_message.....
    .....
   end Store_message;
or
   delay 10.0;
   ..... -- optional sequence of statements.
    .....
end select;
```

Task berkenaan akan menunggu 10.0 saat hingga berlaku *rendezvous*; jika tiada panggilan kepada Store_message dalam tempoh ini, *task* yang mengandungi arahan **delay** akan menjadi *unblocked*. Perlaksanaan diteruskan bagi pernyataan-pernyataan *optional* selepas pernyataan **delay**.

2.0 Rekabentuk Pengnyahralat

Di dalam bahagian ini kita akan mengenalpasti model-model yang digunakan bagi merekabentuk pengnyahralat bagi versi-versi tertentu bahasa-bahasa Pascal, Prolog dan Ada serta faktor-faktor yang mempengaruhi rekabentuk. Terdapat dua faktor utama yang telah dikenalpasti iaitu proses mengnyahralat itu sendiri dan semantik bahasa pengaturcaraan yang terlibat. Pengnyahralat Pascal memanafaatkan faktor pertama di mana kajian terhadap proses mengnyahralat manual (seperti dibincangkan di dalam seksyen 1.0) digunakan bagi mengautomasikan proses mengnyahralat. Sementara faktor kedua dapat dilihat di dalam pembinaan penghnyahralat Prolog dan Ada. Perlaksanaan Prolog berdasarkan Model Kotak dan seterusnya pembinaan pengnyahralat Prolog berorientasikan konsep ini bagi menghasilkan antaramuka pengguna yang berkesan. Bahasa Ada pula membolehkan perlaksanaan acara serempak (*task*) di samping perlaksanaan acara tunggal. Perlaksanaan *task* melibatkan pengongsian sumber dan penjadualan *task-task* dan dengan itu antaramuka pengnyahralat Ada mempamerkan maklumat sumber dan jadual *task*.

2.1 Pengnyahralat Pascal -- Program Error-Locating Assistant System (PELAS)

Kemudahan ini telah dilaksanakan pada sistem NCI UCSD Pascal di mesin IBM PC [Korel, '88]. Versi Pascal ini melaksanakan acara tunggal. Ia membantu pengguna menumpu kepada bahagian aturcara yang mempunyai ralat. Ia berdasarkan pengujian berpandukan laluan. Sistem ini berbentuk interaktif. Sewaktu sesi mengnyahralat ia akan bertanya kepada pengguna tentang ketepatan perlakuan aturcara yang ingin dibaiki. Jawapan ini kemudian akan digunakan bagi membantu usaha-usaha penumpuan. Penumpuan bermula pada arahan pertama yang mengandungi ralat. Apabila ini telah dikenalpasti, penaakulan *backtracking* akan menentukan susunan arahan yang telah memungkinkan terjadinya ralat tersebut. Penaakulan akan merujuk kepada dua struktur data penting: pertama, surihan perlaksanaan dan kedua , rangkaian pergantungan . Surihan perlaksanaan diperlukan kerana sebarang arahan aturcara sehingga arahan yang disyaki berpotensi menyumbang kepada terjadinya ralat. Rangkaian pergantungan pula membentuk hubungan yang mungkin wujud di antara sebarang dua arahan. Hubungan ini menentukan bentuk pengaruh arahan pertama ke atas arahan kedua. Pengaruh ini terdiri daripada tiga jenis [4]:

A) Pengaruh data (data influence) ditakrifkan sebagai :

4
ms 1254, [Korel,'88]

$X^{(q)}$ adalah takrifan bagi v iaitu di mana v diumpukkan nilai (seperti pernyataan umpukan , atau pernyataan input),

$Y^{(p)}$ adalah penggunaan v dan tiada takrifan baru bagi v antara q dan p.

B) Pengaruh kawalan (control influence) ditakrifkan sebagai:

i. *If X then B1 Else B2* bermakna Y adalah di dalam skop pengaruh terhadap X jika Y terdapat di dalam B1 atau B2.

ii. *While X do B1* bermakna Y adalah di dalam skop pengaruh X jika Y di dalam B1 atau X = Y .

C) Pengaruh berkemungkinan (potential influence) ditakrifkan sebagai:

$X^{(q)}$ mempunyai pengaruh berkemungkinan ke atas $Y^{(p)}$ jika terdapat pembolehubah v sekiranya perkara-perkara berikut adalah benar:

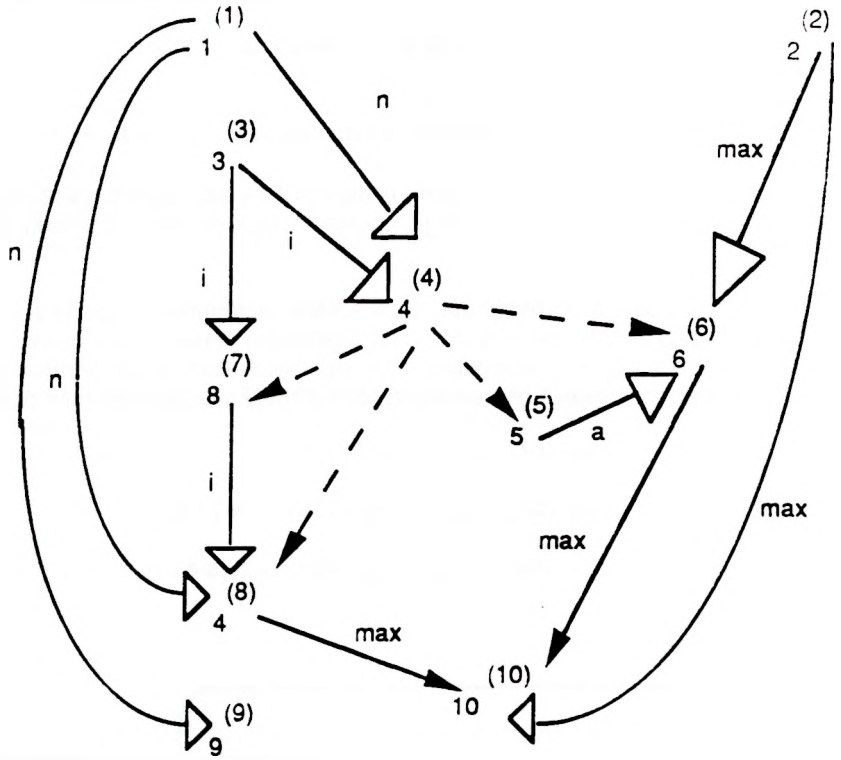
i. X adalah arahan pengujian
ii. Y merupakan arahan yang menggunakan v
iii. Tiada terdapat takrifan bagi v antara p dan q
dan
iv. Terdapat laluan kawalan dari X hingga Y di mana v diubahsuai.

Sebagai contoh , jika aturcara yang digunakan adalah:

```
1       read (n);
2       max := 0;
3       I := 1;
4       while  i < = n   do
begin
5               read (a);
6,7             if  a > max  then  max := a;
8               I := i + 1
          end;
9       write (n);
10      write ( max);
```

Maka surihan perlaksanaan bagi aturcara ini adalah:

arahan	teks arahan
1 (1)	read (n)
2 (2)	max := 0
3 (3)	I := 1
4 (4)	I <= n
5 (5)	read (a)
6 (6)	a > max
8 (7)	I := I +1



Jenis-jenis arca:

hubungan pengaruh data

hubungan pengaruh kawalan

hubungan pengaruh berkemungkinan

Gambarajah D: Rangkaian pergantungan bagi aturcara contoh [5]

4 (8)	**i <= n**
9 (9)	**write (n)**
10 (10)	**write (max)**

di mana bagi arahan a (i), a adalah nombor arahan dan i adalah turutan perlaksanaan arahan. Sementara itu rangkaian pergantungannya pula diberikan oleh Gambarajah D.

Proses penumpuan bermula dengan menyemak nilai-nilai pembolehubah agar seperti yang dijangka. Apabila arahan yang mempunyai nilai pembolehubah yang tidak tepat dijumpai , penaakulan akan mengenalpasti daripada rangkaian pergantungan jenis pengaruh antara arahan tersebut dengan arahan-arahan sebelumnya. Misalnya, jika pergantungannya adalah pengaruh data, maka penaakulan akan menyemak bahawa pernyataan saperti input dan pengumpukan yang terlibat berfungsi saperti

5 ms 1255, [Korel,'88]

jangkaan. atau pun tidak terdapat pengumpukan yang tidak dirancang (misalnya berpunca daripada kesilapan *transcription*). Contoh-contoh yang menyumbang kepada ralat seperti ini adalah

y := z sedangkan kita maksudkan z:=y

atau pun,

y := z sedangkan kita maksudkan y:= t

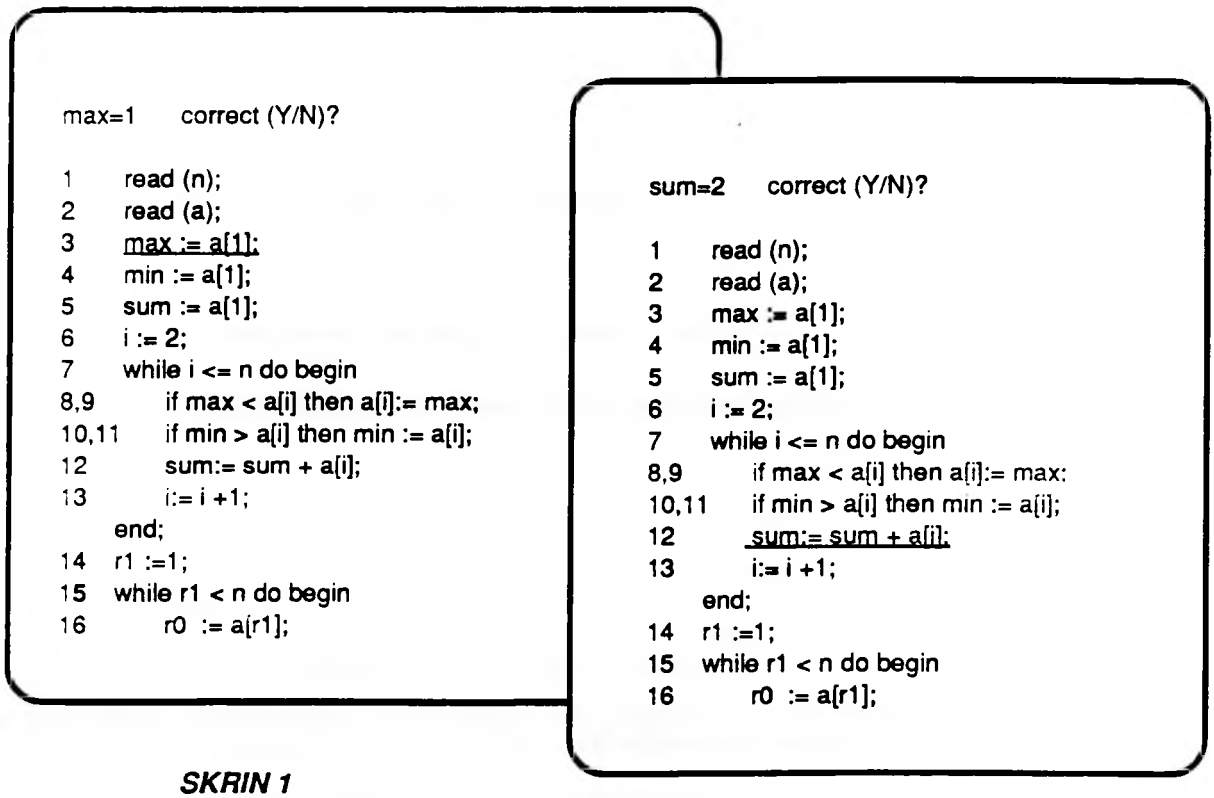
Jika pergantungan merupakan jenis pengaruh kawalan seperti yang melibatkan IF-THEN-ELSE misalnya, maka frasa-syarat pada pengujian IF tersebut akan disemak. Begitu juga dengan gelung WHILE.

Di sepanjang sesi mengnyahralat, arahan yang meyumbang kepada ralat (yang telah dikenalpasti daripada rangkaian pergantungan) berserta beberapa arahan di sekitarnya dipamerkan dan pengguna akan digesa mengenai ketepatan nilai pembolehubah yang beralat setakat ini. Umpamanya, berdasarkan subrangkaian pergantungan berikut (bagi aturcara Min-max di Lampiran II),

30 (57) -----> 3 (3)

29 (56) -----> 12 (170) , 7 (19)

31 (58) -----> 26 (52), 15 (54)



SKRIN 1 ***SKRIN 2***

Gambarajah E: Skrin-skrin penyemakan yang dipamerkan oleh sistem PELAS[6]

6 ms 1259, [Korel, '88]

Sistem PELAS akan mempamerkan skrin 1 dan 2 pada Gambarajah E bagi menyemak nilai-nilai pembolehubah max, sum dan a. (Nama-nama pembolehubah ini telah diberi oleh pengguna kepada Sistem Pelas apabila paparan nilai a pada arahan 30 (57) didapati salah semasa aturcara dilaksanakan sebelum ini.) Arahan 30 (57) mempunyai pergantungan terhadap arahan 3 (3), maka skrin 1 menggesa pengguna mengenai ketepatan nilai max. Oleh kerana pengguna menjawab YES , maka Skrin 2 pula dipamerkan bagi menyemak nilai sum. Pengguna mendapati nilai sum yang dipamerkan hasil daripada arahan 12 (170) adalah salah. Dengan itu Sistem Pelas mengesahkan pula nilai tatasusunan a , yang merupakan salah satu input kepada arahan 12 (170). Ternyata nilai-nilai dalam a tidak tepat. Daripada rangkaian pergantungan didapati arahan 9 (9) menyumbang kepada ralat ini.

```
9 (9)        If max < a[i]    then a[i]  := max;
```

Oleh kerana max telah disahkan ketepatannya di awal sesi , bermakna input kepada arahan ini adalah betul. Seterusnya ini memberi kesimpulan bahawa arahan itu sendiri tidak tepat. Sebenarnya pengguna ingin melaksanakan

```
If max < a[i]   then  " max := a[i] " ;
```

Dengan itu kita telah dapat mengenalpasti suatu ralat logik.

2.2 Pengnyahralat Prolog

Kemudahan Pengnyahralat Prolog ini terdapat pada mesin DEC-10, [Burnham, '85]. Ia direkabentuk dengan mengambilkira cara Prolog melaksanakan aturcara. (Perlu diingatkan di sini versi Prolog ini melaksanakan acara tunggal. Terdapat versi Prolog yang membolehkan perlaksanaan acara serempak dan dipanggil Parlog). Model Kotak telah digunakan di mana setiap prosidur (set arahan) Prolog terdapat di dalam suatu kotak. Andaian dibuat bahawa kotak ini sentiasa tertutup dan kita hanya boleh masuk atau keluar dari kotak apabila menepati syarat-syarat tertentu sahaja. Berikut adalah syarat-syarat yang membolehkan aliran ke dalam kotak:

i.
untuk masuk (ditanda dengan **CALL**) bagi memenuhi gol yang melibatkan cuba memenuhi sebahagian daripada syarat atau pun apabila kita cuba memenuhi permintaan pengguna di terminal.

Contoh:

CALL

```
conc( A,[], A).
conc(A,[H/T],D) : conc([H/T],T,D,
```

ii.
untuk keluar terdapat tiga kemungkinan:

a. ditanda oleh **EXIT** jika berjaya.
b. ditanda oleh **FAIL** jika gagal.
c. ditanda oleh **REDO** jika ia sebahagian daripada gol seterusnya; umpamanya, semasa penilaian alternatif kedua, ketiga dan seterusnya bagi syarat mahu pun fakta.

Teknik instrumentasi aturcara digunakan bagi menggambarkan aliran masuk dan keluar kepada sebarang set gol-gol (sebarang kotak) dengan memaparkan mesej-mesej CALL, EXIT, FAIL dan REDO pada output pengnyahralat. Di samping itu nombor panggilan (invocation number) menyatakan tingkat(depth) perlaksanaan. Sebagai contoh, bagi pangkalan pengetahuan berikut [7],

```
fruit (victoria_plum).
fruit (watermelon).
shape (victoria_plum,round).
shape (watermelon,round).
colour (victoria_plum,purple).
colour (watermelon,green).
tembikai(X) :- fruit(X), shape (X,round), colour(X,green).
```

penilaian tembikai(X) akan menghasilkan output:

```
(1)  0 Call: tembikai(_24)          ----------baris 1
(2)  1 Call: fruit(-24)             ----------baris 2
(2)  1 Exit: fruit(victoria-plum)   ----------baris 3
(3)  1 Call: shape (victoria_plum, round)  ----------baris 4
(3)  1 Exit: shape (victoria_plum,round)   ----------baris 5
(4)  1 Call: colour(victoria_plum,green)   ----------baris 6
(4)  1 Fail: colour(victoria_plum,green)   ---------- baris 7
(3)  1 Redo:shape(victoria_plum,round)     ----------baris 8
(3)  1 Fail: shape(victoria_plum,round)    ----------baris 9
(2)  1 Redo: fruit(victoria_plum)          ----------baris 10
(2)  1 Exit: fruit(watermelon)             ----------baris 11
(5)  1 Call: shape(watermelon,round)       ----------baris 12
(5)  1 Exit: shape(watermelon,round)       ----------baris 13
(6)  1 Exit: shape(watermelon, round)      ----------baris 14
(6)  1 Call: colour(watermelon,green)      ----------baris 15
(6)  1 Exit: colour(watermelon,green)      ----------baris 16
(1)  0 Exit: tembikai(watermelon)          ----------baris 17

X = watermelon
```

Kotak-kotak berikut dibentuk oleh pengnyahralat bagi penilaian tembikai(X) ke atas pangkalan pengetahuan yang diberi.

a. kotak fruit

```
fruit(victoria_plum).
fruit(watermelon).
```

b. kotak shape

```
shape(victoria_plum, round).
shape(watermelon, round).
```

7
ms 83, [Burnham, '85].

c. kotak color

```
colour(victoria_plum, purple).
colour(watermelon, green).
```

d. kotak tembikai

```
tembikai(X):- fruit(X),shape(X,round),colour(X,green).
```

Untuk memahami kegunaan kotak-kotak ini kita kaji apa yang berlaku semasa perlaksanaan. Bagi baris 1 output, [1] adalah nombor panggilan dan 0 adalah tingkat perlaksanaan. Di baris 7 subgol **colour** gagal (Fail), dan kotak-kotak **shape** dan **fruit** dimasuki di baris 8 dan 10 sebelum REDO untuk *backtracking*. Di baris 11 **fruit** dimasukki dan berjaya dengan pilihan kedua X = watermelon. Setelah itu kotak-kotak **shape** dan **colour** dimasuki di baris 12 dan 15 , dan berjaya (Exit). Seterusnya gol **tembikai** pun berjaya (baris 17, Exit). Mesej instrumentasi yang merangkumi nombor baris, nombor panggilan dan kotak yang dimasuki atau yang baru ditinggalkan berserta sifat panggilan (Fail, Exit, Redo) menggambarkan kepada pengguna aliran perlaksanaan.

2.3 Pengnyahralat ADA -- Debugging Interpreter System

Pengnyahralat yang akan dihuraikan di sini merupakan pengnyahralat yang dibina khusus bagi pengaturcaraan serempak dalam Ada. Ia dibina dan disepadukan dengan sistem Acturus, suatu pentafsir interaktif yang dibangunkan di University of California, Irvine bagi Programming Environment Project [Brindle, '89]. Setakat ini takrifan umum yang dapat disimpulkan bagi pengnyahralat adalah alat bantu bagi penumpuan ralat dan pembetulan ralat. Maka bagi persekitaran Ada, skop tradisi pengnyahralatan dikecilkan sedikit agar kita dapat memanfaatkan kerja-kerja suatu pengnyahralat. Khususnya, pengnyahralat Debugging Interpreter System bagi Ada ini akan membolehkan pengguna menyemak perlaksanaan aturcara dengan lebih meluas dan berdasarkan skala masa yang boleh diherotkan (*distorted time scale*). Untuk ini, Debugging Interpreter System akan membantu pameran persekitaran masa-perlaksanaan dan membolehkan kawalan kemajuan penghitungan(*progress of computation*).

Bagi mencapai objektif-objektif di atas, beberapa ciri penting telah disertakan pada pengnyahralat Ada ini. Pertama, ia memanafaatkan suatu ciri penting sesuatu pengnyahralat iaitu persekitaran yang membolehkan pengulangan perlaksanaan. Khususnya bagi Ada, pengnyahralat perlu mampu mengawal masa agar acara-acara dalam satu *task* boleh diasingkan daripada perlaksanaan *task-task* lain dan kesan-kesan *task* yang diasingkan ini boleh diselidiki. Bagi melaksanakan *task* ini, Debugging Interpreter System menyediakan suruhan-suruhan bagi pengaturcaraan di titik hentian dan penjejakan keseluruhan sistem atau proses-proses tertentu. Proses-proses merangkumi *task-task* dan waktu.

Ciri kedua adalah pengnyahralat menyediakan pameran sejarah perlaksanaan dan data aturcara [redacted]; umpamanya, pengnyahralat mengumpulkan maklumat mengenai panggilan prosidur-prosidur atau sebahagian daripadanya. Pengguna boleh mendapatkan pameran ini dengan memasukkan suruhan tertentu di titik hentian. Bagi perlaksanaan tunggal, pameran terdiri daripada timbunan perlaksanaan berserta penunjuk kepada kod sumber. Bagi perlaksanaan serempak pula pameran akan merangkumi :

i. timbunan perlaksanaan berserta penunjuk kepada setiap *task* .

ii. maklumat setiap *task* saperti status perlaksanaan. sebab bagi status menanti dan keutamaan *task* .

iii. maklumat hubungan antara proses-proses iaitu: hubungan tuan-hamba antara frem dan *task-task* , hubungan bapa-anak antara frem dan *task* - *task* , giliran bagi setiap kemasukan dan masa-masa lengah.

Ciri ketiga yang telah dipertimbangkan berkaitan dengan perlaksanaan *task*. Algoritma penjadualan, pemilihan alternatif bagi nantian berpilih dan tempoh sebenar sesuatu lengah bergantung kepada perlaksanaan sesuatu sistem Ada. Berdasarkan faktor pertama iaitu penjadualan . Debuggung Interpreter System menyediakan kemudahan pengaturcaraan di titik hentian. Penjadualan berpandukan acara boleh digunakan oleh pengguna bagi titik hentian yang tidak dapat diletakkan pada kod sumber. Umpamanya, bagi menghentikan sebentar perlaksanaan apabila suatu *task* baru saja digera setelah menanti di pernyataan nantian berpilih. Walau bagaimana pun hanya acara-acara yang berkaitan dengan perlaksanaan serempak dibenarkan sebagai acara syarat. Bagi faktor kedua pula iaitu pemilihan alternatif , pengguna boleh memasukkan alternatif-alternatif pada kod sumber dengan didahului oleh "S".

Dari segi rekabentuk, pereka-pereka Debugging Interpreter System ini telah menggabungkan konsep pentafsir selaku korutin dan pentafsir selaku subprogram. Konsep korutin ini dapat dilihat pada perhubungan di antara dua korutin utama bagi Debugging Interpreter System iaitu: Ada Interpreter dan Break and Insert Manager (BIM). Pemindahan dari BIM ke Ada Interpreter dilaksanakan melalui suruhan "start", "continue" atau "step", sementara pemindahan dari Ada Interpreter ke BIM terjadi apabila pengguna menekan kekunci <break> secara interaktif atau pun kodseudo "break" ditemui di dalam kod sumber Ada. Sementara itu pengaturcaraan di titik hentian dikendalikan oleh rutin Debugger Executor yang merupakan subprogram kepada Debugging Interpreter System.

3.0 Kesimpulan

Pengnyahralat bagi bahasa-bahasa pengaturcaraan telah melalui berbagai perkembangan. Perkembangan-perkembangan ini dipengaruhi oleh, pertama, logik proses mengnyahralat itu sendiri serta, kedua, oleh ciri-ciri semantik bahasa pengaturcaraan yang terlibat.

Faktor pertama telah menghasilkan pendekatan saperti penumpu ralat yang pada asasnya merupakan suatu usaha mengautomasikan proses mengesan ralat secara manual. Pendekatan ini terdapat pada Sistem PELAS saperti yang telah dibincangkan di Seksyen 2.1.

Faktor kedua iaitu pengaruh semantik bahasa pengaturcaraan dapat dilihat pada rekabentuk suatu pengnyahralat Prolog. Output pengnyahralat tersebut merupakan label-label bagi setiap kotak prosidur yang dimasuki dan ditinggalkan samada penilaian berjaya atau tidak.

Faktor semantik juga menyedarkan kepada kita betapa pentingnya ciri-ciri seperti pengasingan proses, kejelasan data dan sejarah perlaksanaan aturcara disertakan bersama perisian pengnyahralat bagi perlaksanaan acara serempak. Ini kerana sesuatu pengnyahralat itu dibina khusus bagi sesuatu bahasa dan dengan itu fungsi pengnyahralat bergantung kepada cara bahasa tersebut dilaksanakan pada mesin tertentu. Bagi bahasa Ada, dengan adanya ciri-ciri seperti pengasingan proses dan sejarah perlaksanaan disediakan oleh pengnyahralatnya, pengaturcara dapat memastikan proses-proses dilaksanakan mengikut giliran saperti yang dirancang. Sementara itu ciri kejelasan data pada pameran pengnyahralat dapat membantu pengaturcara mengenalpasti ketepatan skop butir-butir data yang terlibat.

Akhir sekali walau pun tugas pengaturcara dapat diringankan dengan mengautomasikan proses pengesanan ralat atau pun dibantu oleh mesej-mesej pengnyahralat yang canggih, namun kesemua ini hanya akan mendatangkan menafaat yang berlipat ganda sekiranya di tahap rekabentuk pengaturcara amat teliti dan menggunakan teknik-teknik rekabentuk yang terjamin prestasinya seperti rekabentuk

atas-bawah dan rekabentuk berorientasikan objek. Di samping itu pengaturcara perlulah sentiasa berpegang teguh kepada panduan-panduan *deskchecking* seperti memastikan skop dan penggunaan pembolehubah tepat; pemerihalan input-output selaras dengan kehendak masaalah : pembilang-pembilang dan indeks-indeks gelung diberikan nilai awal yang tepat; syarat-syarat gelung sesuai dan akhir sekali aturcara yang dibina merupakan terjemahan algoritma yang paling tepat. ■

Rujukan

1. [**Korel, '88], Korel, Bogden**; PELAS -- "Program Error-Locating Assistant Systems"; IEEE Transactions on Software Engineering , Vol.14, No. 9, September 1988.

2. [**Brindle, '89], Brindle, Anne F., Taylor, Richard N. , and Martin, David F.** : "A Debugger for Ada Tasking"; IEEE Transactions on Software Engineering, Vol 15, No 3, March 1989.

3. [**Booch, '83], Booch, Grady**; Software Engineering with Ada; The Benjamin /Cummings Publishing Company, Inc., Menlo Park, California, 1983.

4. [**Burnham, '85], Burnham, W.D. and Hall, A.R.**; Prolog Programming and Applications: Macmillan Education Ltd., 1985.

5. [**Fairley, '79], Fairley, Richard E.**; "ALADDIN : Assembly Language Assertion-Driven Debugging Interpreter"; IEEE Transactions on Software Engineering, Vol SE-5, No. 4, July 1979.

6. [**Faizah, '87], Faizah Ahmad**; Level-independant Language as a Tool For Teaching Assembly Language Programming; M Sc. Thesis, Universiti Sains Malaysia, P.Pinang, April 1987.

7. [**Goodenough], Goodenough, John B., SoftTech Inc.**; A Survey of Program Testing Issues.

8. [**Gould, '75], Gould, J.D.**; " Some psychological evidence on how people debug computer program,"; International Journal on Man-Machine Studies, vol. 7, pp. 151-182, March 1975.

9. [**Huang, '80], Huang, J.C.**; "Instrumenting Programs for Symbolic Trace Generation", COMPUTER, December 1980.

10. [**Shumate, '88], Shumate, Ken**; Understanding Concurrency in Ada; MacGraw-Hill Book Company, New York, 1988.