MENGENAL TEKNOLOGI AGEN BERPINDAH

Lukito E. Nugroho[1], Lussiana[2], Dewi L. Puspitasari[2]

[1]Monash University, Australia
[2]Universitas Gunadarma, Indonesia

Abstrak

Agen berpindah adalah sebuah teknologi baru yang memungkinkan eksekusi komputasi berpindah dari satu komputer ke komputer lain dalam sebuah jaringan. Dengan perkembangan Internet dan teknologi komunikasi yang begitu pesat, teknologi ini menjadi sangat relevan dalam membuka paradigma baru komputasi yang mengandalkan mobilitas. Tujuan makalah ini adalah mengupas aspek arsitektur sistem migrasi agen berpindah serta pemrogramannya. Makalah ini juga mengusulkan paradigma pemrograman baru berbasis mobilitas untuk pengembangan sistem agen berpindah.

Kata Kunci: agen berpindah, komputasi bergerak, pemrograman, java

1. Pendahuluan

Internet dan kemajuan teknologi komunikasi telah mengubah secara drastis cara orang bekerja dengan komputer. Diawali dengan konsep komputasi terdistribusi yang memungkinkan orang bekerja dengan komputer tanpa tergantung pada lokasi, trend ini berlanjut dengan mode kerja bergerak *(mobile)*. Komputasi bergerak adalah mode komputasi di mana mobilitas menjadi salah satu ciri. Salah satu komponen yang sering dieksploitasi mobilitasnya adalah kode program. Mobilitas kode adalah kemampuan kode program untuk berpindah dari satu lingkungan eksekusi ke lingkungan eksekusi yang lain. Kemampuan ini sering dimanfaatkan untuk mengimplementasikan konsep agen berpindah *(mobile agent)*.

Agen adalah sebuah program yang dibuat untuk melaksanakan suatu tugas tertentu. Agen sering dibuat untuk mewakili pemiliknya, dan memiliki otonomi dalam menjalankan tugasnya. Sebagai contoh, agen untuk berbelanja secara *on-line* dapat diprogram untuk melakukan penawaran dan transaksi secara non-interaktif. Agen berpindah adalah agen yang dalam proses eksekusinya dapat berpindah dari satu lokasi ke lokasi yang lain.

Dibandingkan dengan client-server, paradigma agen berpindah memiliki beberapa kelebihan, antara lain [1,2]:

- *Menghemat biaya komunikasi.* Dalam aplikasi basis data misalnya, sebuah sesi query yang kompleks memerlukan komunikasi intensif antara client dan server. Dengan agen berpindah, kode client dapat dikirimkan ke server dan diaktifkan di sana. Komunikasi yang terjadi bersifat lokal dan jauh lebih cepat.
- *Fleksibilitas operasional melalui proses asinkron.* Agen berpindah dapat dijalankan secara asinkron, lepas dari aplikasi induknya. Pada saat yang ditentukan, sebuah agen dan induknya dapat saling berkomunikasi. Model seperti ini dapat meningkatkan fleksibilitas penggunaan komputer karena aplikasi induk tidak harus menunggu selesainya eksekusi agen berpindah.
- *Dapat disesuaikan dengan kebutuhan secara fleksibel.* Pada model client-server, fungsionalitas diletakkan di sisi server dan biasanya bersifat tetap *(fixed)*. Jika diinginkan ada tambahan layanan dikemudian hari, servernya harus ditingkatkan. Cara ini membuat server menjadi kompleks, dan konsekuensinya peningkatan server sering disertai dengan peningkatan kebutuhan sumberdaya komputasi. Dengan agen berpindah, fungsionalitas diletakkan pada agen sebagai client. Server biasanya berfungsi generik, statis, dan tidak memerlukan banyak sumberdaya. Peningkatan layanan dapat dilakukan pada sisi client tanpa harus mengubah konfigurasi server.

Pemrograman dan eksekusi agen berpindah pada umumnya dilakukan pada sistem berbasis

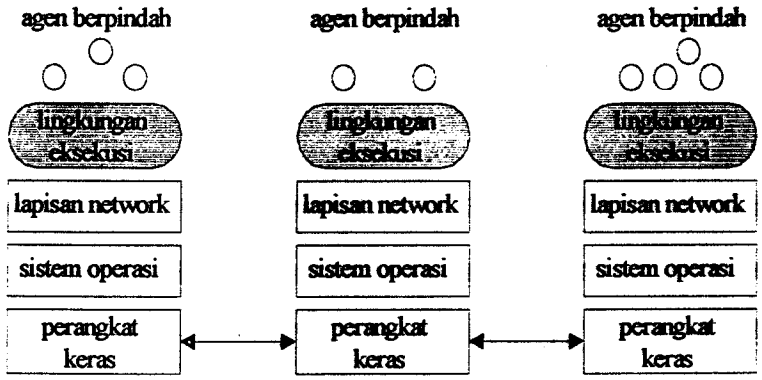
bahasa Java. Hal ini tidak mengherankan karena Java menyediakan dukungan untuk pemrograman sistem terdistribusi yang menjadi cikal bakal pemrograman agen berpindah. Makalah ini mencoba menganalisis aspek teknis implementasi sistem agen berpindah, terutama mengenai arsitektur sistem migrasi serta pemrogramannya.

Dengan semakin derasnya intervensi perangkat komputer berbasis jaringan ke dalam kehidupan sehari-hari, semakin tinggi pula tuntutan untuk memanfaatkan teknologi agen berpindah dalam proses pemecahan masalah *(problem-solving)* berbasis mobilitas. Dalam wacana ini, mobilitas agen tidak lagi menjadi bagian dari fungsionalitasnya, tetapi berdiri sendiri sebagai suatu abstraksi terpisah. Dengan demikian pemrograman agen berpindah juga mengalami pergeseran karakteristik. Kini pemrograman mobilitas agen merupakan disiplin baru yang terpisah dari pemrograman terhadapt aspek fungsionalnya. Berdasarkan fenomena ini, makalah ini juga mengusulkan paradigma pemrograman baru yang lebih cocok untuk menghadapi tuntutan kebutuhan.

Organisasi makalah ini adalah sebagai berikut. Bab 2 membicarkan tentang arsitektur sistem agen berpindah. Bab 3 menjelaskan tentang teknologi sistem terdistribusi yang mendasari sistem agen berpindah. Implementasi sederhana agen berpindah didiskusikan pada bab 4. Bab 5 mendiskusikan tentang otonomi agen berpindah, dilanjutkan dengan proposal paradigma pemrograman baru untuk agen berpindah pada bab 6. Akhirnya beberapa kesimpulan disampaikan pada bab 7.

2. Arsitektur

Secara umum arsitektur sistem agen berpindah ditunjukkan pada gambar 1. Pada gambar 1, tiap kelompok komputer merepresentasikan sebuah "lokasi" yang saling terhubung melalui jaringan.



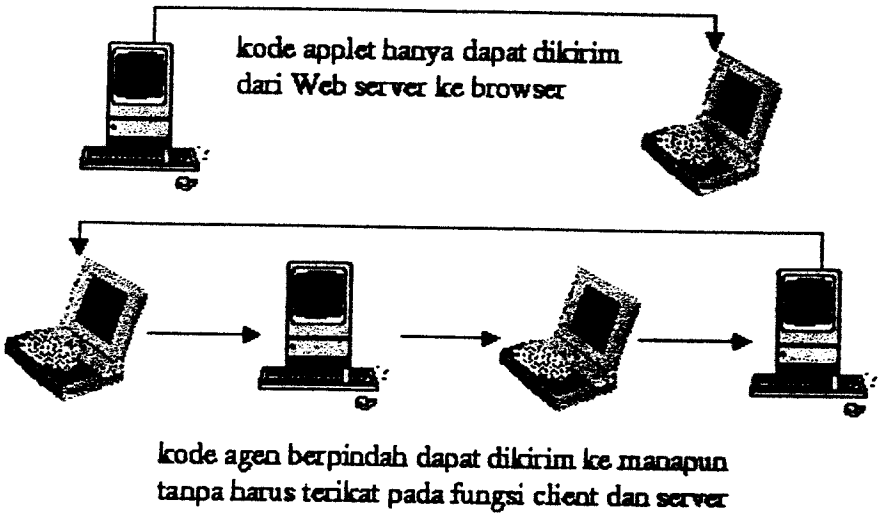
Gambar 1. Arsitektur Sistem Agen Berpindah

Di sebuah komputer, agen berpindah "hidup" di atas lingkungan_eksekusi yang merupakan *run-time system* baginya. Perbedaan lingkungan eksekusi agen berpindah dengan lingkungan eksekusi program tradisional adalah kemampuannya dalam "mengirimkan" dan "menerima" agen berpindah ke dan dari lingkungan eksekusi yang lain.

Dalam bahasa berorientasi obyek seperti Java, agen, seperti halnya unit komputasi lainnya, diwujudkan dalam struktur obyek. Dengan demikian kita dapat mengatakan bahwa terminologi "agen" dan "obyek" dalam konteks makalah ini sesungguhnya menunjuk pada jenis entitas yang sama. Istilah "agen" lebih mengarah pada makna "peran", sementara "obyek" bersifat netral dan mengarah pada aspek linguistik pada level implementasi.

Berdasar pada pengertian di atas, kita dapat mengatakan bahwa agen berpindah pada dasarnya adalah obyek berpindah. Agen berpindah direalisasikan dengan mengirimkan kode program dari sebuah obyek dalam bahasa Java dari satu lokasi ke lokasi lain. Hal ini mirip dengan applet, tetapi agen berpindah tidak mengikuti paradigma client-server yang dipakai applet. Tidak ada batasan konseptual bahwa perpindahan agen hanya berlaku searah (dari lokasi server Web ke lokasi client

tempat browser berlokasi). Sebaliknya, kode program dapat dikirimkan ke tiap lokasi yang mendukung. Gambar 2 mengilustrasikan perbedaan antara applet dan agen bergerak.



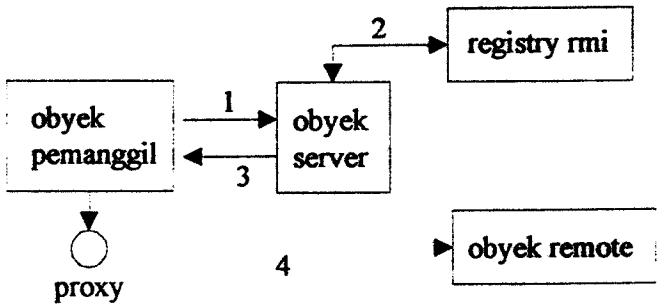
Gambar 2. Perbedaan Antara Applet dan Agen Bergerak

Dalam sistem berbasis Java, perpindahan obyek dapat diwujudkan melalui teknologi *Remote Method Invocation* (RMI) [3]. RMI adalah teknologi berbasis client-server yang memungkinkan sebuah method dari suatu obyek dipanggil dari jauh (obyek pemanggil dan yang dipanggil tidak berada pada lokasi yang sama). Dalam sistem obyek terdistribusi, kemampuan ini menjadikan akses ke obyek transparan terhadap distribusi lokasinya.

3. Pemrograman RMI

Dalam RMI, sebuah obyek disebut "jauh" *(remote)* bila kode programnya dapat dipanggil dari lingkungan eksekusi yang berbeda. Bagaimana obyek dalam sebuah program mengakses obyek lain yang berada di luar lingkup lingkungan eksekusinya ? RMI menggunakan model "referensi" untuk mengatasi problem ini. Referensi adalah sebuah obyek yang berfungsi sebagai proxy (penghubung) antara obyek pemanggil dan obyek jauh yang dipanggil. Bagi obyek pemanggil, referensi tampak seperti obyek yang dipanggil. Obyek pemanggil dapat memanggil method dari obyek jauh sepetri halnya memanggil sesama obyek lokal.

Sebuah sistem RMI tersusun dari beberapa komponen program. Komponen pertama adalah interface. Interface adalah deklarasi tentang method yang dimiliki oleh sebuah obyek jauh. Interface ini menjadi "kontrak" yang mengikat obyek pemanggil dan obyek jauh, karenanya interface dimiliki oleh kedua pihak. Bagi obyek pemanggil, interface memberitahu method apa saja yang bisa dipanggil, sementara obyek jauh berkewajiban mengimplementasikan semua method yang tercantum dalam deklarasi interface. Komponen kedua adalah obyek server. Server biasanya berfungsi sebagai "manager". Ia yang membuat obyek jauh dan mengelola *name binding,* semacam teknik untuk mempublikasikan obyek jauh. Bila server mendaftarkan obyek jauh pada komponen *registry RMI* (berfungsi sebagai buku direktori), maka obyek jauh bisa diquery oleh obyek pemanggil yang berada di lokasi yang berbeda. Gambar 3 menunjukkan proses pemanggilan method jarak jauh dengan RMI. Pada saat obyek pemanggil ingin memanggil method obyek jauh, maka ia akan menghubungi obyek server untuk menanyakan obyek jauh yang dimaksud (langkah 1). Obyek server kemudian akan mencari obyek jauh di registry RMI (langkah 2). Jika ada, maka RMI akan mengembalikan proxy dari obyek jauh tersebut ke obyek pemanggil (langkah 3). Akhirnya obyek pemanggil secara langsung memanggil method obyek jauh melalui proxy (langkah 4).

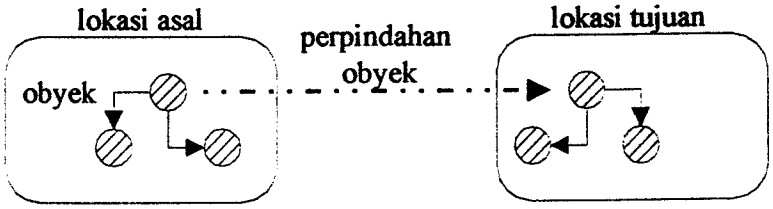


Gambar 3. Cara Kerja RMI

4. Teknik Migrasi Obyek

Seperti disebutkan sebelumnya, RMI menyediakan transparansi akses dengan cara menyembunyikan detil perbedaan lokasi dari obyek-obyek yang terlibat. Dalam sistem agen berpindah, hal yang sebaliknya terjadi. Perpindahan obyek dispesifikasikan dengan cara menyebutkan lokasi tujuan, sehingga, abstraksi tentang lokasi tidak dapat disembunyikan (dibuat transparan).

Sebuah sistem agen berpindah sederhana dapat dibuat dengan membangun abstraksi lokasi di atas fungsi client dan server pada RMI. Program yang berisi obyek pemanggil tidak lagi menyiratkan fungsi client, tetapi menggambarkan sebuah lokasi *(host)* asal dari sebuah agen berpindah. Demikian pula program yang berisi obyek yang dipanggil sekarang menggambarkan lokasi tujuan agen berpindah. Arsitektur sistem agen berpindah secara sederhana ditunjukkan pada gambar 4 berikut ini.



Gambar 4. Arsitektur Sistem Agen Berpindah

Sistem pada gambar 4 dapat diimplementasikan oleh beberapa kelas, seperti yang diusulkan Farley [4]. Kelas `GenericAgentHost` merepresentasikan lokasi agen berpindah. Jika berfungsi sebagai lokasi asal, kelas ini bertugas membuat obyek agen, menentukan lokasi tujuan, menginisialisasi perpindahan obyek, dan mendefinisikan aktivitas agen di lokasi tujuan. Di sisi lain, setelah proses inisialisasi perpindahan obyek, lokasi tujuan bertugas "menarik" obyek agen dari lokasi asal. Karena proses perpindahan obyek dilakukan dengan RMI, maka kelas `GenericAgentHost` harus mengimplementasikan interface `java.rmi.Remote` dan `java.rmi.RemoteException`.

Dalam sistem berikut ini terjadi komunikasi jarak jauh antara dua obyek yang merepresentasikan lokasi asal dan lokasi tujuan agen berpindah, dan obyek agen merupakan bagian dari obyek lokasi. Untuk mengoperasionalkan sistem ini, sebelumnya harus dibuat elemen-elemen proxy dari kedua obyek lokasi dengan kompiler proxy, yaitu rmic.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

// Interface untuk tiap lokasi agen berpindah
public interface AgentHost extends Remote {
  /* Method ini dipanggil oleh lokasi asal yang akan
     mengirimkan obyek agen ke lokasi ini. Lokasi ini kemudian
     memangil method extractAgent() dari lokasi asal yang akan
```

```
    melakukan proses perpindahan obyek */
  public void acceptAgent(AgentID id, AgentHost sender)
              throws RemoteException;

  /* Method ini dipanggil oleh lokasi tujuan yang akan mengambil
     obyek agen. */
  public void extractAgent(AgentID id) throws RemoteException;
}
```

Kelas `GenericAgentHost` didefinisikan sebagai kelas abstrak yang harus diturunkan untuk dapat menggunakannya.

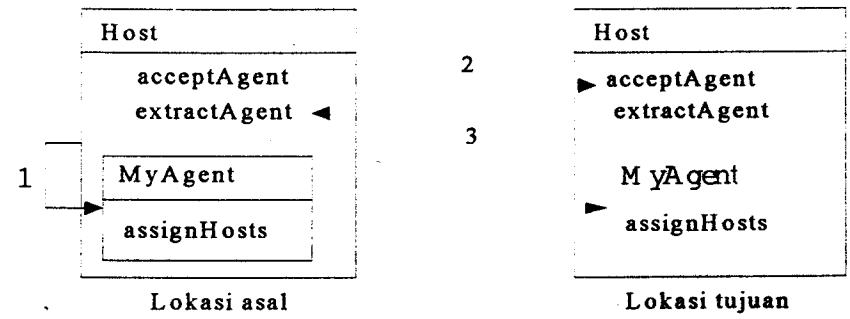
```
import java.util.Hashtable;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public abstract class GenericAgentHost extends UnicastRemoteObject
                implements AgentHost {
  /* Obyek agen, thread eksekusinya, serta thread untuk memonitor
     status agen disimpan dalam hashtable menggunakan identitas agen
     sebagai kuncinya. */
  private Hashtable agents_;
  private Hashtable threads_;
  private Hashtable monitors_;

  public GenericAgentHost() throws RemoteException {
    // Membuat hashtable untuk private data di atas
  }

  // Implementasi method acceptAgent
  public void acceptAgent(AgentID id, AgentHost sender)
              throws RemoteException {
    // 1) ambil agen dari lokasi asal dan simpan di hashtable
    Agent agent = sender.extractAgent(id);
    agents_.put(id,agent);

    // 2) buat thread eksekusi untuk agen dan simpan di hashtable
    // 3) buat thread untuk memonitor status eksekusi agen dan
    //    simpan di hashtable
    // 4) aktifkan kedua thread
  }

  public Agent extractAgent(AgentID id) throws RemoteException {
    // 1) stop thread eksekusi agen dan keluarkan dari hashtable
    // 2) lakukan hal yang sama untuk thread monitor

    // 3) keluarkan agen dari hashtable dan kirim ke lokasi tujuan
    Agent agent = (Agent) agents_.remove(id);
    return agent;
  }
}
```

Obyek agen direalisasikan dari kelas `Agent`. Karena obyek dari kelas ini akan dieksekusi oleh thread yang berbeda dengan thread program utama, maka kelas ini harus mengimplementasikan interface `Runnable` atau mewarisi kelas `Thread`.

```
import java.util.Vector;
import java.io.Serializable;

public abstract class Agent implements Runnable, Serializable {
  private AgentID identity_;
  private Vector hosts_;

  // Method ini menunjukkan ke mana obyek agen akan pergi
  public void assignHosts(Vector hosts) {
    // set lokasi-lokasi tujuan ke sebuah vektor
```

```
  }

  /* Method ini dipanggil oleh lokasi agen saat itu untuk mengetahui lokasi
     tujuan berikutnya. Method ini dipanggil setelah agen menyelesaikan
     tugasnya di lokasi tersebut. */
  public AgentHost nextHost() {
    // ambil elemen berikutnya dari vektor lokasi
  }

  /* Method ini merupakan entry-point bagi eksekusi thread agen.
     Semua aktivitas agen di sebuah lokasi berawal dari method ini.
     Method ini harus di-override oleh turunan dari kelas Agent. */
  public abstract void run();
}
```

Cara kerja sistem agen berpindah ini ditunjukkan pada gambar 5. Proses perpindahan obyek agen dimulai saat lokasi asal memanggil method `assignHosts` dari obyek agen untuk meberitahu lokasi tujuan (langkah 1). Selanjutnya lokasi asal memberitahu lokasi tujuan bahwa obyek agen siap dipindahkan, dengan cara memanggil method `acceptAgent` dari lokasi tujuan (langkah 2). Menimpali permintaan ini, dalam method `acceptAgent` lokasi tujuan menjawab dengan memanggil method `extractAgent` dari lokasi asal (langkah 3), yang kemudian mengirimkan obyek agen sebagai kembalian (return value) dari method `extractAgent`. Setelah obyek agen sampai di lokasi tujuan, thread yang akan mengeksekusinya diaktifkan (masih oleh method `acceptAgent`).



Gambar 5. Cara Kerja Sistem Agen Berpindah Sederhana

Perlu diingat bahwa yang sebenarnya terjadi dalam pengiriman obyek agen ke lokasi tujuan adalah duplikasi obyek. Dengan demikian setelah salinan obyek sampai, ada dua obyek kembar di kedua lokasi. Efek "pengiriman" obyek dapat dicapai dengan menghapus obyek aslinya di lokasi asal. Dalam Java, hal ini dapat secara otomatis dilakukan dengan cara memutus semua referensi ke obyek asli tersebut, sehingga ia akan dihapus dari memori oleh prosedur "pengumpulan sampah" *(garbage collection)*.

Dari contoh sistem agen berpindah di atas, peran RMI sangat besar dalam mewujudkan transparansi. Pengiriman obyek agen antar lokasi sebagai kembalian method misalnya, dilakukan seperti layaknya pengembalian obyek secara lokal. Di samping itu, RMI juga memberikan jenis transparansi lain. Yang pertama adalah transparansi akses terhadap bytecode (tersimpan dalam file .class) dari sebuah obyek jauh. Jika sebuah obyek memerlukan file .class dari sebuah obyek jauh, RMI secara otomatis akan men-downloadnya dari lokasi obyek jauh tersebut. Proses ini dapat dilakukan dengan menggunakan protokol transfer data yang umum dipakai seperti HTTP. Transparansi juga diterapkan dalam proses transfer obyek antar lokasi, khususnya dalam proses *marshalling* dan *unmarshalling* obyek. Proses transformasi struktur obyek ini amat rumit bila dikerjakan secara manual [5], tapi dengan mekanisme serialisasi obyek dari Java, detil teknis dari proses ini dapat sepenuhnya disembunyikan.

5. Konsep Otonomi Dalam Agen Berpindah

Dalam sistem agen berpindah di atas, perpindahan agen dikendalikan sepenuhnya oleh lokasi asal dan lokasi tujuan. Agen tidak berperan apapun dalam proses ini dan hanya bertindak sebagai obyek yang pasif. Hal ini tidak sejalan dengan filosofi dasar agen yang mengisyaratkan adanya otonomi dalam melakukan tugasnya. Bagi agen berpindah, otonomi ini berbentuk inisiatif untuk berpindah. Dalam pemrograman berorientasi obyek, hal ini berarti sebuah obyek agen memiliki kemampuan untuk memindahkan dirinya sendiri tanpa pengaruh dari lokasi asal dan tujuannya.

Beberapa piranti pengembangan untuk sistem agen berpindah seperti Aglet API [6] dan Voyager [7] telah mengimplementasikan karakteristik ini. Dalam Voyager misalnya, mobilitas sebuah obyek diperoleh dengan cara mengaplikasikan "pembungkus" *(facet)* yang memberikan karakteristik mobilitas kepada obyek tersebut. Teknik "pembungkusan" ini diimplementasikan dengan mekanisme interface. Dengan pembungkus mobilitas, sebuah obyek dapat memanggil method `moveTo` yang akan membawanya ke lokasi lain. Kode program berikut ini mengilustrasikan cara pengiriman obyek ke lingkungan eksekusi yang berjalan di mesin xyz.domainku.com port 9000.

Interface IAgent.java

```
public interface IAgent {
  void doTask();
}
```

Kelas Agent.java

```
import java.io.Serializable;

public class Agent implements IAgent, Serializable {
  public void doTask() {
    ...
  }
}
```

Kelas Aplikasi.java

```
import com.objectspace.voyager.*
import com.objectspace.voyager.mobility.*;

public class Aplikasi {
  public static void main(String[] args) {
    try {
      Voyager.startup(7000);          // start lingkungan eksekusi di port 7000
      IAgent agent = (IAgent) Factory.create("Agent");   // membuat obyek lokal
      IMobility magent = Mobility.of(agent);     // "bungkus" untuk obyek lokal
      magent.moveTo("//xyz.domainku.com:9000"); // pindah ke lokasi lain
      agent.doTask();                           // dieksekusi di lokasi lain
    } catch (Exception e) {
      System.err.println(e);
    }
    Voyager.shutdown();
  }
}
```

Dibandingkan dengan model agen berpindah pada bab 4, pendekatan yang dilakukan oleh Voyager di atas lebih sesuai dengan konsep awal agen berpindah. Sebuah obyek dapat menjadi *mobile* bila ia "dibungkus" dengan bungkus yang berkemampuan mobilitas.

Model agen berpindah seperti yang ditawarkan Voyager di atas sebenarnya merupakan pengembangan dari model sederhana yang dijelaskan sebelumnya. Inti dari perpindahan obyek tetaplah RMI dan mekanisme serialisasi. Yang dilakukan oleh Voyager adalah membangun lapisan abstraksi baru di atas lapisan RMI. Voyager juga mengimplementasikan arsitektur seperti yang ditunjukkan pada gambar 1: agen berpindah hidup dan beroperasi pada lingkungan eksekusi yang merepresentasikan lokasi. Berbeda dengan model sebelumnya, dalam konsep ini lokasi merupakan entitas pasif, sebaliknya agen berpindah adalah entitas aktif yang memiliki otonomi untuk berpindah dari satu lokasi ke lokasi lain.

6. Paradigma Pemrograman Berbasis Mobilitas

Mengimplementasikan mobilitas sebagai sifat intrinsik dari sebuah obyek dan menjadikannya sebagai bagian dari fungsionalitas merupakan solusi *ad-hoc* untuk problem-problem komputasi bergerak pada umumnya. Seperti yang diungkapkan dalam [8], pendekatan *ad-hoc* ini menimbulkan konsekuensi negatif ditinjau dari aspek rekayasa perangkat lunak, misalnya diingkarinya prinsip-prinsip pemrograman yang ada. Pendekatan *ad-hoc* juga tidak bisa digunakan untuk memberikan dukungan secara generik untuk sistem-sistem bergerak, karena berbagai tipe entitas bergerak (misalnya pemakai, kode program, dan data) membutuhkan dukungan dalam bentuk yang berbeda [9].

Strategi pemisahan aspek-aspek pemrograman *(separation of programming concerns)* dapat digunakan untuk mengatasi persoalan-persoalan di atas. Secara umum, aspek mobilitas dipisahkan dari aspek fungsionalitas dari sebuah program. Aspek mobilitas diangkat ke level abstraksi yang lebih tinggi. Dalam pemrograman agen berpindah, hal ini berarti pengendalian migrasi agen dilakukan *di luar* konstruksi fungsional. Hal ini dapat dilakukan bila bahasa pemrograman yang digunakan mampu mengabstraksi konsep tentang lokasi dan merepresentasikannya dalam konstruksi linguistik yang sesuai. Sebagai contoh, bahasa **Mocha** [8] memiliki tipe data untuk merepresentasikan sebuah lokasi, mengabstraksi migrasi agen dalam metafora "kendaraan", serta menggunakan sebuah konstruksi loop yang bersifat meruang *(spatial)* untuk mengontrol migrasi. Kode program berikut ini melakukan hal yang sama dengan kode program pada bab 5.

```
journey Vehicle {        // kendaraan untuk berpindah ke lokasi lain
  Agent agent;           // agen yang akan berpindah
  Path tujuan = new Path("xyz.domainku.com");   // lokasi tujuan

  void doMigration() {
    Location loc;        // variable lokasi
    on loc of tujuan {   // loop spatial
      agent.doTask();    // dieksekusi di lokasi tujuan
    }
  }
}
```

Dari kode di atas terlihat bahwa aspek perpindahan agen sepenuhnya dikendalikan dari luar agen itu sendiri. Konstruksi `journey` berfungsi meng-enkapsulasi aspek ini dalam sebuah konstruksi linguistik. Tiap obyek yang didefinisikan di dalam `journey` bersifat *mobile* dan dapat berpindah ke lokasi lain. Konsep lokasi sendiri direalisasikan melalui sebuah kelas, `Location`, yang dapat digunakan untuk membentuk senarai *(list)* lokasi-lokasi tujuan dalam kelas `Path`. Proses perpindahan agen diwujudkan melalui sebuah loop yang bersifat meruang. Dalam loop ini, eksekusi bersifat meruang pula. Pada contoh di atas, eksekusi method `agent.doTask` akan dilakukan di setiap lokasi yang sedang ditunjuk oleh variabel `loc` yang beriterasi di atas senarai lokasi tujuan.

Pemisahan aspek mobilitas dari aspek fungsionalitas memberikan beberapa keuntungan bagi pemrograman agen berpindah. Kontrol mobilitas dapat dilakukan dalam level tinggi (misalnya menggunakan loop meruang seperti contoh di atas), sehingga pemrogramannya dapat dilakukan secara lebih ringkas. Pemisahan juga memungkinkan pemrograman mobilitas dilakukan tanpa harus mengorbankan prinsip-prinsip rekayasa perangkat lunak. Sebagai contoh, modifikasi pada salah satu aspek tidak akan mengganggu modularitas aspek yang lain.

7. Kesimpulan

Makalah ini menjelaskan tentang teknologi agen berpindah sebagai salah satu sarana untuk menuju moda komputasi bergerak di mana mobilitas menjadi ciri dalam pemanfaatan komputer. Dalam lingkungan pemrograman Java, sebuah sistem agen berpindah dapat diimplementasikan secara langsung dengan memanfaatkan teknologi RMI. Dalam perkembangannya, aspek otonomi agen dapat pula direalisasikan dengan cara membangun lapisan abstraksi baru di atas lapisan RMI. Makalah ini membahas pula tentang paradigma baru untuk pemrograman sistem bergerak, termasuk sistem agen berpindah. Dengan paradigma baru ini mobilitas diangkat ke tingkat abstraksi yang lebih tinggi,

terpisah dari aspek fungsionalitas program.

Sebagai penutup, perlu diingat bahwa teknologi agen berpindah tidak hanya mencakup aspek perpindahan agen saja. Aspek-aspek lain yang penting adalah masalah keamanan, konkurensi, dan adaptivitas. Masalah keamanan muncul karena, tidak seperti applet, agen yang masuk ke sebuah lingkungan eksekusi dapat berasal dari lokasi yang tidak dikenal *(untrusted)*. Konkurensi terutama berkaitan agen yang menyediakan layanan tertentu (berfungsi sebagai server). Persoalannya adalah bagaimana layanan tersebut dapat diakses secara bersamaan tanpa harus mengorbankan integritas data. Faktor adaptivitas muncul karena sifat non-deterministik dari sistem terdistribusi di mana agen berpindah berada. Untuk dapat berfungsi sebagaimana mestinya, agen berpindah perlu memiliki derajad adaptivitas tertentu.

8. Daftar Pustaka

[1] D. Chess, C. Harrison dan A. Kershenbaum, "Mobile Agents: Are They a Good Idea?. Mobile Object Systems: Towards the Programmable Internet," *J. Vitek dan C. Tschudin (Ed)*, LNCS 1222, Springer, hal. 25-47, 1997.

[2] A. Fugetta, G.P. Picco dan G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, hal. 342-361, 1998.

[3] Sun Microsystems, Java Remote Method Invocation (RMI) Specification. <http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmi.TOC.doc.html>

[4] S.R. Farley, "Mobile Agent System Architecture: A Flexible Alternative to Moving Data and Code to Complete a Given Task," *Java Report*, SIGS Publications, April 1997.

[5] L. Nugroho, *A Programmer's Tool for Managing Persistent Object Structures*, Masters Thesis, James Cook University, 1994.

[6] D.B. Lange, *Java Aglet Application Programming Interface White Paper*, IBM Tokyo Research Laboratory, 1997.

[7] ObjectSpace Inc., "Voyager ORB Developer Guide". <http://www.objectspace.com/>

[8] L. Nugroho, B. Srinivasan and A.S.M. Sajeev, "Separation of Concerns in Mobile Object Programs: The Case of Supporting Mobility and Concurrency," in Proceedings of *the International Symposium on Mobile Agents and Applications*, Baden-baden, Germany, July-August 2000.

[9] L. Nugroho, S.W. Loke, B. Srinivasan, A.S.M. Sajeev and I.K. Ibrahim, "A Context-Based Model for Programming Mobility," *Accepted for presentation in the Second International Workshop on Information Integration and Web-based Applications and Systems*, Yogyakarta, September 2000.