SKEMA PEMETAAN PEMODELAN UML DAN PEMROGRAMAN JAVA

Sholiq[1)]

[1)]Program Studi Sistem Informasi Sekolah Tinggi Manajemen Informatika & Teknik Komputer Surabaya (STIKOM), email: sholiq@stikom.edu

Abstract: In fact Java is standard object orieneted programming language. Unified Modelling Language (UML) is in fact visual model standard in designing software object oriented. Java as object oriented programming language used applied model result using UML. Both have elements interpretation, so that can be gotten syncronisesion between some diagram in UML and java language fiture. This paper flatten map scheme like class diagram, sequential, component, and starchart to resource code srtucture in java. Map scheme evaluation test used simple model part, than made UML diagram required. This paper discuss strudtured approach in arousing java code from UML elements.

Keywords: Java, UML, Object Oriented, Visual Model, Map Scheme

Model adalah abstraksi yang menjelaskan hal-hal signifikan pada persoalan yang komplek dengan mengabaikan hal-hal yang tidak diperlukan, sehingga membuat persoalan lebih mudah dipahami. Sedangkan abstraksi adalah kemampuan dasar manusia yang memungkinkan berurusan dengan sesuatu yang komplek. Misalkan, ahli teknik, seniman, dan pengarang membuat model untuk masalah-masalah yang dihadapi sebelum benar-benar secara nyata membangunnya.

Sebagaimana ahli teknik sipil yang akan membangun jembatan, maka untuk membangun sistem yang mempunyai tingkat kompleksitas signifikan, pengembang harus membuat model sistem yang menampilkan beberapa pandangan dari sudut berbeda terhadap suatu sistem yang dihadapi, membangun model menggunakan notasi-notasi yang tepat, melakukan verifikasi bahwa model yang dibuatnya memenuhi persyaratan-persyaratan sistem, dan menambahkan detail secara berangsur-angsur untuk mentranformasikan model menjadi implementasi. (Boggs, 2002).

Ibarat membangun rumah, maka seorang arsitek memerlukan beberapa pandangan rumah yang akan dibangunnya. Tapak depan, tapak samping, pandangan pondasi, atau pandangan lainnya. Banyaknya pandangan yang dibuat, akan memberikan pemahaman yang lebih baik bagi para pembuat rumah untuk mewujudkan apa yang diinginkan seorang arsitek. Demikianlah terhadap sistem informasi yang lebih abstrak dibandingkan dengan bangunan rumah, maka diperlukan banyak pandangan terhadap sistem yang akan dibangun, agar didapatkan pandangan yang lebih komprehensip terhadap sistem informasi.

Untuk mendapatkan banyak pandangan terhadap sistem informasi yang akan dibangun, UML menyediakan beberapa diagram visual yang menunjukkan berbagai aspek dalam sistem. Ada beberapa diagram yang disediakan dalam UML antara lain: (1)Diagram *use case*, (2) Diagram aktivitas, (3) Diagram sekuensial, (4) Diagram kolaborasi, (5)Diagram kelas, (6)Diagram *statechart* , (7) Diagram komponen, dan (9) Diagram *deployment*. Diagram-diagram di atas digunakan untuk memperbaiki komunikasi antara stakeholder yang terlibat dalam rekayasa perangkat lunak.

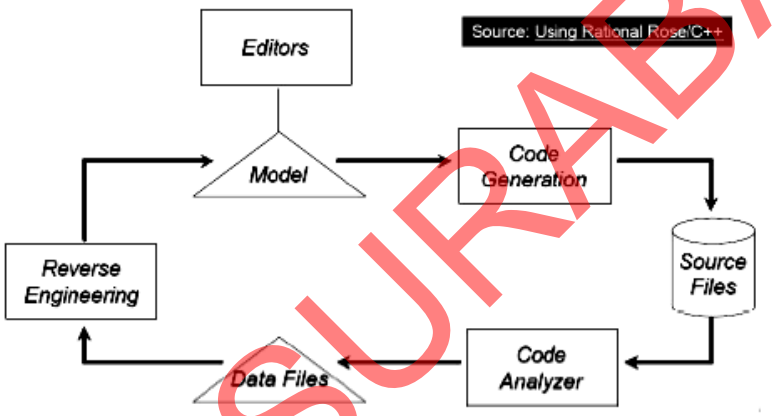
Beberapa kegunaan diagram menurut Boggs(2002) diberikan sebagai berikut:

- Diagram use case : Diagram ini menggambarkan fungsionalitas sistem menurut kaca mata client atau pemakai. Diagram ini lebih banyak digunakan sebagai alat komunikasi antara pengembangan dan client dalam menurunkan kebutuhan.
- Diagram aktivitas : Diagram ini digunakan untuk memodelkan sebuah skenario di dalam use case, dimana skenario ini biasa disebut sebagai flow of events. Flow of events menggambarkan interaksi antara aktor dengan sistem dalam rangka menjalankan sebuah use case tertentu. Dengan diagram aktivitas, maka flow of events di dalam use case akan mudah dikomunikasikan karena ditampilkan dalam notasi-notasi grafik.
- Diagram sekuensial dan Diagram Kolaborasi : Kedua diagram ini digunakan untuk menunjukkan langkah-langkah kerja sama obyek-obyek di dalam *use case*. Obyek apa saja yang dibutuhkan untuk aliran, pesan apa saja yang obyek kiriman ke obyek lainnya, dan urutan pesan-pesan yang dikirimkan.
- Diagram kelas : Diagram ini digunakan menunjukkan kelas-kelas di dalam sistem dan relasinya antara kelas-kelas.
- Diagram statechart : Digunakan menunjukkan tingkah laku dinamik obyek di dalam sistem. Diagram *statechart* menunjukan siklus hidup sebuah obyek tunggal, dari saat dibuat sampai obyek tersebut dihapus. Diagram ini adalah cara tepat untuk memodelkan perilaku dinamis sebuah kelas.
- Diagram komponen : Diagram ini menggambarkan hubungan modul fisik dari kode. Komponen bisa mencantumkan pustaka kode program dan berkas-berkas runtime sekaligus.

- Diagram deployment: *Deployment* adalah segala hal yang berkaitan dengan penyebaran fisik aplikasi. Hal ini termasuk persoalan layout jaringan dan lokasi komponen-komponen dalam jaringan.

Ketika membangun sebuah program, sering sebuah sistem dipresentasikan dalam sebuah model dan kode sumber. Tujuan model adalah untuk memvisualisasikan struktur relasi, interaksi, atau beberapa fitur tentang sistem. Kode sumber mengandung implementasi bahasa pemrograman spesifik dan mengandung semua detail yang dibutuhkan untuk menjalankan program.

Model atau kode sumber masing-masing adalah master yang berkosentrasi pada dua hal yang berbeda. Meskipun demikian keduannya mempunyai fitur-fitur atau informasi yang saling dapat di-sharing untuk menjaga sikronisasi antara model dan kode sumber. Jika antara model dan kode sumber tidak sinkron, maka diperlukan sebuah proses untuk melakukan sinkronisasi. Proses yang digunakan untuk melakukan sinkronisasi antara model dan kode sumber disebut sebagai Rountrip engineering. Di dalam proses rountrip engineering mengandung dua hal utama yaitu forward engineering dan reverse engineering. Forward engineering adalah proses pembangkitan kode sumber dari elemen-elemen model, dan sebaliknya proses reverse engineering adalah proses rekontruksi ulang model dari kode sumber. Skema rountrip engineering diberikan pada Gambar 1.



Gambar 1 Proses rountrip engineering

Melihat skema rountrip engineering antara model dan kode sumber di gambar 1, maka diperlukan sebuah skema pemetaan antara model dan kode sumber dari bahasa pemrograman tertentu. Diperlukan skema pemetaan ini agar didapatkan sebuah konsistensi pemetaan antara model UML dan kode sumber bahasa pemrograman.

METODE

Beberapa konsep dalam ML dan bahasa pemrograman berorientasi obyek mempunyai pemetaan secara langsung. Sebagaimana yang diberikan oleh Genova et.ol (2003) pada tabel 1 tentang pemetaan dari bahasa pemrograman obyek oriented ke konsep-konsep UML.

Tabel 1 Pemetaan bahasa pemrograman berorientasi obyek ke konsep-konsep UML

Fitur bahasa berorientasi obyek	Konsep UML
Assignments	Actions in State Diagram Action states in Activity Diagram
Control Flow	Branch in Activity Diagram Guard Condition in Interaction Diagram Guard Condition in State Diagram
Primitive Data Types	User Defined Stereotypes in Class Diagram
Comments	Notes in all Diagram
Operators	Not mentioned in UML but can be used in the action state in activity diagram
Classes Declarations	Declare as class in class diagram
Data Encapsulations	Declare as class in class diagram
Polymorphisms	Supported since inheritance can be represented in UML
Assertions	Constraint in OCL

Messages	Messages in Interaction Diagrams
Interfaces	Interface in class diagram
Class Templates	Class templates in class diagram
Control name Space Conflict	Packages in component/class diagram

Ada tiga pendekatan untuk membangkitkan kode antara lain: pendekatan terstruktur yang didasarkan pada penggunaan model-model stuktur obyek. Sebagian besar tool-tool UML membangkitkan struktur dari diagram kelas, model pembangkitan ini mempunyai kekurangan dimana tidak membangkitkan ekspresi perilaku yang ada di dalam kode yang dibuat.

Pembangkitan kedua adalah pendekatan mesin state, dimana pendekatan ini menggunakan statemachine dari obyek-obyek ditambahkan dengan struktur obyek. Struktur obyek dan state-machine memungkinkan pembuatan kode yang komplit termasuk perilaku obyek.Kekurangan dari oendekatan ini adalah dimana pengembang harus mengekspresikan semua perilaku dengan mesin state yang mana dalam praktek kadang-kadang tidak berguna.

Sedangkan pendekatan ketiga adalah pendekatan translative yang didasarkan pada aplikasi dan arsitektur yang dilandasi dengan metode Shlaer dan Mellor. Dengan pendekatan ini model aplikasi dan arsitektur yang lengkap dikembangkan. Setelah pemgembangan model, berdasarkan aturan yang ditemukan di model maka kode dibuat.

Pada makalah ini akan dibahas pembangkitan kode dari model dengan menggunakan penndekatan pertama, sedangkan pendekatan kedua dan ketiga diluar bahasan makalah ini. Pemetaan antara elemen-elemen notasi UML ke dalam bahasa java diberikan pada tabel 2.

Tabel 2 Pemetaan UML ke Java

UML	Java
Package of classes/component	Package
Dependency between components and packages in Component Diagram	Import
Class Component	Class or defined component
Class Interface	Class Interface
Realization between classes and Interfaces	Implements
Generalization between classes or Interfaces	Extends
Association between classes	Reference Attributes on both classes
Attribute	Attribute
Operation	Method
Properties on classes (Abstract, Final)	Class Modifiers (Abstract, Final)
Properties on attributes	Attribute modifiers
Visibility	Visibility
Implementation access	Package level visibility

Langkah-langkah utama pembangkitan kode untuk diagram kelas diberikan di bawah ini:

- Package akan ditransformasikan ke package di Java. Kelas-kelas dan interface yang ada di dalam paket juga akan dibangkitkan.
- Kelas dan interface di dalam diagram ditranformasikan ke dalam kode Java. Operasi dan attribut di dalam kelas juga akan dibangkitkan ke Java.
- Untuk masing-masing kelas akan dibangkitkan ke kode Java.
- Relasi antar kelas akan dibangkitkan ke kode Java.
- Jika kondisi pre/post sebuah operasi eksis, mereka akan ditranslasikan ke kode sumber kode.

Sedangkan pembangkitan relasi antar kelas diberikan oleh Kirk (2001) diberikan sebagai berikut:

Tabel 3 Pemetaan relasi UML ke Java

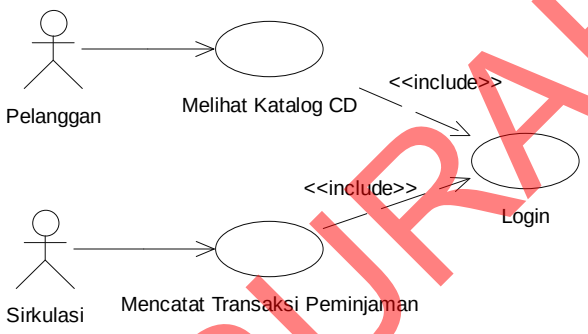
Relasi di UML	Kode Java
Assoiasi dua arah (bidirectional association)	Instanisasi level kelas di kelas yang mengacuh
Assoiasi dua arah (bidirectional association)	Instanisasi level kelas di kedua kelas yang terlibat relasi
Dependensi (dependency)	Ada 3 yaitu: o Kelas yang diacuh diinstan di level metode dari kelas yang mengacuh.

	o Kelas yang diacuh dijadikan tipr parameter dari metode di kelas yang mengacuh. o Kelas yang mengacuh dijadikan nilai balik dari sebuah metode kelas yang mengacuh.
Aggregasi (aggregation)	Instanisasi sebagaimana relasi assosiasi satu arah.
Realisasi (realize realization)	Implements
Generalisasi	Extends

HASIL DAN PEMBAHASAN

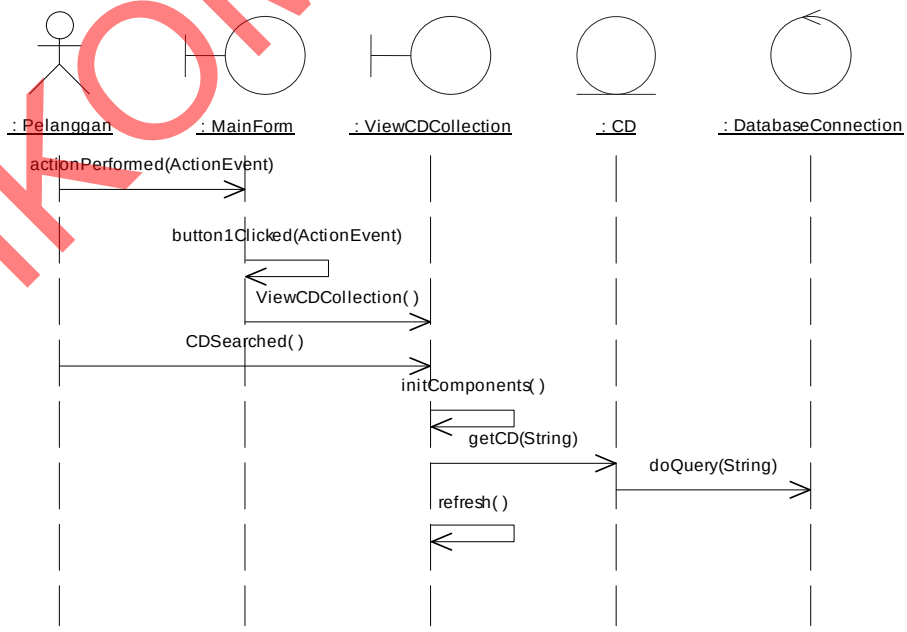
Untuk menguji beberapa skema pemetaan elemen-elemen UML ke Java akan dilakukan dengan memberikan contoh pemodelan sederhana dimana diberikan dalam potongan diagram use case, kemudian dari satu use case dibuat diagram sekuensial. Selanjutnya dari diagram sekuensial dibuat diagram kelas, dan dari diagram kelas dapat dilakukan pembangkitan kode dengan memperhatikan skema pemetaan elemen-elemen UML ke Java sebagaimana diberikan di tabel 1, 2 dan 3.

Potongan contoh pemodelan yang digunakan adalah sistem peminjaman Compact Disk(CD), dimana dalam sistem peminjaman CD diberikan potongan diagram use case pada Gambar 2.



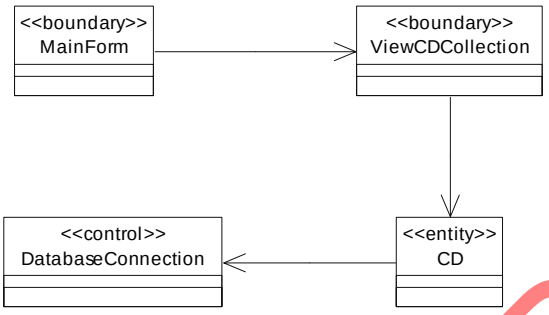
Gambar 2 Potongan diagram use case

Dari potongan diagram use case dibuat diagram sekuensial untuk use case “Melihat Katalog CD” sebagaimana diberikan pada Gambar 3.



Gambar 3 Diagram sekuensial dari use case “Melihat katalog CD”

Dari diagram sekuensial yang sudah dibentuk dengan memberikan justifikasi daftar atttribut pada masing-masing kelas didapatkan kelas-kelas dan diagram kelas pada Gambar 4.



Gambar 4 Potongan diagram kelas

Kemudian dengan menggunakan pemetaan sebagaimana diberikan di bagian metode, maka didapatkan struktur kelas-kelas di Java pada Gambar 5.

```
import java.awt.event.ActionEvent;
public class CD
{
   String id;
   String title;
   public DatabaseConnection theDatabaseConnection;

   /**
    * @roseuid 44E16FD301F9
    */
   public CD()
   {
   }
   /**
    * @param cd
    * @return CD
    * @roseuid 44E16C090000
    */
   public CD getCD(String cd)
   {
    return null;
   }
}

class ViewCDCollection
{
   public CD theCD;

   /**
    * @roseuid 44E16B8D00FC
    */
   public ViewCDCollection()
   {
   }

   /**
    * @roseuid 44E16BC0039E
    */
   public void CDSearched()
   {
   }

   /**
    * @roseuid 44E16BF5010F
    */
   public void initComponents()
   {
```

```
        }

        /**
         * @roseuid 44E16C6701FA
         */
        public void refresh()
        {
        }
    }

    class DatabaseConnection
    {

        /**
         * @roseuid 44E16FD30285
         */
        public DatabaseConnection()
        {
        }

        /**
         * @param sql
         * @roseuid 44E16C5301A1
         */
        public void doQuery(String sql)
        {
        }
    }

    class MainForm
    {
        public ViewCDCollection theViewCDCollection;

        /**
         * @roseuid 44E17075006C
         */
        public MainForm()
        {

        }

        /**
         * @param ActionEvent
         * @roseuid 44E16AE70284
         */
        public void actionPerformed(ActionEvent e)
        {

        }

        /**
         * @param ActionEvent
         * @roseuid 44E16B4E012D
         */
        public void button1Clicked(ActionEvent e)
        {
        }
    }
```

Gambar 5 Struktur kode sumber hasil pembangkitan dari diagram kelas menggunakan rational rose 2002

Dari gambar 5 diperoleh beberapa elemen-elemen UML yang dipetakan ke kode sumber Java antara lain: kelas, attribut, operasi, parameter operasi, dan nilai balik metode dari sebuah kelas. Dengan pembangkitan ini maka jika dalam proyek sistem informasi skala rekayasa perangkat lunak, dimana seorang analisis bekerja dengan beberapa orang programmer, maka akan didapatkan pola dan gaya pemrograman yang identik. Disamping itu, dengan konsep pembangkitan ini akan ada dua pekerjaan yang disisakan buat programmer yaitu: membuat rangcangan antar muka yang efektif dan melengkapi metode-metode dari suatu kelas.

SIMPULAN

Pembangkitan kode sumber dari elemen-elemen UML membantu menemukan rancangan pola terbaik terhadap aplikasi yang akan dibangun. Jika seorang analis bekerja dengan banyak programmer dalam membangun sistem yang besar, maka para programmer akan bekerja dengan pola dan gaya pemrograman yang sama.

DAFTAR RUJUKAN

Boggs, W., and Boggs, M. 2002. *Mastering UML with Rational Rose 2002*. Alameda: Sybex.

Gonzalo, G., Carlos, R. C., and Juan L. 2003. Mapping UML Associations into JavaCode. *Journal Of Object Technology*, Vol. 2, No. 5, September-October 2003.

Kirk, K. 2001. UMLJavaReference.pdf. *Modeling Java Applications using UML Language Mappings.*

Lethbridge, T. C., and Laganiere, R. 2002. *Object-Oriented software Engineering, International edition.* Maidenhead: McGrawHill.

Yaldiz, S. 2004. *Roundtrip Engineering for Classes: Mapping between UML Diagram and Java Structures based on Poseidon for UML™ and the Eclipse™ Platform*. Submitted in partial fullfilment of the requirements for the degree Master of Science in Information and Media Technologies.