PENERAPAN METODOLOGI RAISE DALAM VERIFIKASI PERANGKAT LUNAK SECARA FORMAL

Dhanny Setiawan

Sekolah Tinggi Manajemen Informatika dan Komputer LIKMI
Jl. Ir. H. Juanda 96 Bandung 40132

E-mail:dhanny@likmi.ac.id

Abstrak

Abstrak— Penelitian ini mencoba untuk meneliti tentang menspesifikasikan kebutuhan perangkat lunak secara formal dengan metodologi RAISE. Pada Penelitian ini juga dicoba untuk meneliti bagaimana metode berorientasi objek dapat diterapkan untuk membantu spesifikasi awal untuk kemudian dipetakan menjadi metode formal. Sebagai studi kasus, maka digunakan suatu sistem informasi yang mempunyai sebuah basis data, dan dua buah entitas. Karena suatu basis data akan dapat diakses oleh banyak *user* secara bersama-sama, maka akan timbul masalah konkurensi

Kata-kata Kunci : metode formal, metode berorientasi objek, RAISE, konkurensi.

1. PENDAHULUAN

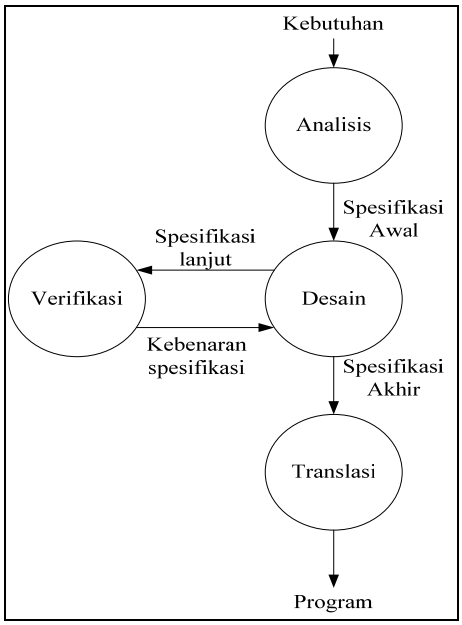
Pengembangan perangkat lunak secara umum mengikuti mempunyai tahapan-tahapan yang dilalui, yaitu pendefinisian masalah, analisis kebutuhan, tahapan desain, tahapan pembuatan program (implementasi), tahapan integrasi, tahapan pengujian, tahapan operasional dan tahapan perawatan[6]. Tahapan analisis memegang peranan yang sangat penting, karena di tahapan inilah spesifikasi kebutuhan dihasilkan. Kesalahan dalam melakukan spesifikasi kebutuhan dapat berakibat sangat fatal, bahkan bagi sistem tersebut dapat mengakibatkan kehilangan nyawa. Oleh karena itu, sebaiknya kesalahan spesifikasi ditemukan dalam tahap analisis tersebut Untuk mengurangi kesalahan yang mungkin terjadi pada tahap analisis maka diperlukan

suatu tahapan verifikasi. Verifikasi adalah proses untuk memeriksa bahwa proses pengembangan perangkat lunak sudah tepat, termasuk formulasi dan justifikasi relasi formal antara langkah-langkah pengembangan[5].

Ada beberapa metode yang digunakan untuk pengembangan perangkat lunak, yaitu metode pengembangan terstruktur, maupun metode pengembangan berbasis objek. Tetapi kedua metode ini masih memerlukan tahapan pengujian yang membutuhkan biaya yang cukup besar untuk melakukan verifikasi terhadap kebutuhan perangkat lunak yang sudah dispesifikasikan. Oleh karena itu, diperlukan suatu metode yang mempunyai tingkat akurasi yang sangat tinggi dalam membuat spesifikasi kebutuhan perangkat lunak tersebut. Metode yang baru berkembang ini dikenal dengan nama metode formal. Metode formal adalah metode untuk membantu dalam meningkatkan ketepatan dan kehandalan dari sebuah sistem, karena metode formal menggunakan beberapa pendekatan matematika, logika, dan pendekatan lainnya untuk menentukan ketepatan dan kehandalan sistem[5].

Ada beberapa metodologi yang dapat diguanakan untuk mengembangkan metode formal, salah satunya adalah metodologi RAISE. RAISE adalah salah satu metode formal yang dirancang untuk membantu dalam melakukan analisis kebutuhan *software* pada kalangan industri. RAISE dikembangkan untuk memperbaiki metode VDM (*Vienna Development Model*), yaitu dengan menambahkan pendekatan aljabar[8]. Spesifikasi perangkat lunak dengan menggunakan metodologi RAISE dapat ditulis dalam *RAISE Specification Language* (RSL), yang dapat melakukan spesifikasi sampai pada tahap yang terperinci sehingga dapat ditranslasikan dengan mudah ke dalam bahasa pemrograman tertentu. Walaupun demikian, RAISE terfokus pada analisis spesifikasi kebutuhan sehingga belum ada metode verifikasi yang tepat untuk digunakan.

Tahapan pengembangan perangkat lunak menggunakan RAISE dapat digambarkan seperti pada Gambar I.1.



Gambar I. 1 Tahapan RAISE

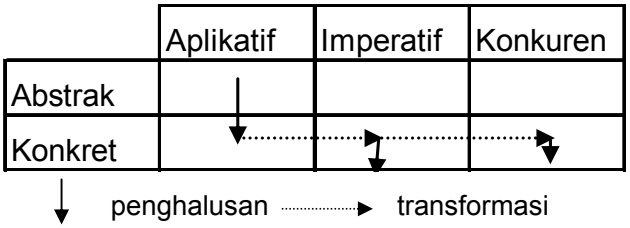
2. TINJAUAN PUSTAKA

RAISE adalah salah satu metode rekayasa perangkat lunak, sehingga tahapan – tahapan pengembangannya mempunyai kemiripan dengan metoda pengembangan rekayasa perangkat lunak pada umumnya. Tahapan – tahapan pengembangan dalam RAISE adalah proses analisis, desain, verifikasi dan translasi, seperti terlihat pada gambar I.1

Sebagai awal untuk tahap analisis, maka dapat digunakan metode pengembangan perangkat lunak berbasis objek. Tahapan pengembangan dengan basis objek diawali dengan analisis diagram *use case*, kemudian dilanjutkan dengan pembuatan skenario *use case*, *state-transition diagram*, *entity-relationship diagram*. Setelah pembuatan diagram-diagram tersebut, maka dapat dilanjutkan dengan pembuatan spesifikasi dengan metodologi RAISE. Hasil dari tahap analisis adalah spesifikasi awal perangkat lunak yang dapat ditulis dengan menggunakan RSL. Spesifikasi awal ini merupakan kebutuhan perangkat lunak yang ditulis dalam bahasa yang digunakan sehari-hari. Spesifikasi awal merupakan masukan bagi tahapan desain untuk menghasilkan spesifikasi akhir yang dapat ditulis juga menggunakan RSL. Kemudian spesifikasi akhir akan diterjemahkan menjadi program dalam bahasa pemrograman tertentu pada tahap translasi.

Biasanya, spesifikasi awal ditulis dengan gaya abstrak aplikatif sekuensial, selanjutnya spesifikasi awal akan diperhalus dengan membuat tipe data menjadi lebih

konkret. Dengan adanya tipe yang lebih konkret, fungsi – fungsi yang menggunakan tipe tersebut juga dapat didefinisikan secara lebih konkret. Selanjutnya, tipe yang sudah konkret aplikatif ini ditransformasi untuk mendekati bahasa pemrograman sehingga ditransformasikan menjadi konkret imperatif. Kemudian, jika ada tahapan konkurensi, maka konkurensi dispesifikasikan setelah tahapan imperatif. Tahapan-tahapan ini dapat digambarkan seperti pada Gambar II.1



Gambar II.1 Rute Pengembangan Perangkat Lunak dalam RAISE[5]

3. TEORI RSL

RSL adalah sebuah bahasa untuk membuat spesifikasi perangkat lunak secara formal. Pembahasan RSL disini akan melingkupi komponen-komponen yang terdapat dalam RSL, kemudian modul-modul yang terdapat dalam RSL, tipe-tipe data di dalam RSL, dan fungsi dalam RSL

Sebagai sebuah bahasa spesifikasi formal, RSL mempunyai 4 komponen, yaitu aturan-aturan sintaks, aturan-aturan yang *well-formedness*, semantik, dan logika[6]. Aturan – aturan sintaks menentukan simbol – simbol yang digunakan dan juga menentukan bagaimana simbol – simbol tersebut dapat bergabung untuk membentuk struktur yang lebih besar. Aturan – aturan *well-formedness* berkaitan dengan *scope*, visibilitas, dan pemeriksaan tipe. Dengan RSL dapat didefinisikan suatu identifier atau operator.

Ada dua jenis modul yang terdapat dalam RAISE, yaitu modul sistem dan modul pendukung[5]. Modul sistem adalah modul yang menghasilkan sebagian besar spesifikasi. Setiap modul sistem akan mempunyai sebuah tipe yang disebut *type of interest*, yang diberi nama sesuai dengan nama sistem yang akan dibangun dan menggambarkan status sistem tersebut. Pembuatan modul sistem dimulai dengan membuat sebuah modul abstrak aplikatif yang mempunyai *type of interest* yang menggambarkan status keseluruhan dari sistem. Kemudian status ini ditransformasikan sehingga menjadi lebih konkret. Proses ini selesai jika semua tipe

sudah mengandung tipe konkret RSL atau *type of interest* dari modul standard.

Ada tiga macam modul pendukung, yaitu modul tipe, modul tambahan, dan modul parameter. Modul tipe digunakan untuk mendefinisikan setiap tipe yang akan digunakan dalam spesifikasi, disertai dengan fungsi – fungsi aplikatif yang memanipulasi tipe – tipe tersebut. Modul tipe biasanya dibangun dengan cara yang sangat sederhana, seperti penambahan tipe dan fungsi yang akan dipakai selanjutnya. Modul tambahan digunakan untuk mendefinisikan fungsi – fungsi generik pada sebuah tipe data yang konkret. Modul parameter digunakan untuk mendefinisikan parameter – parameter bagi modul lainnya.

RSL digunakan untuk pembuatan spesifikasi perangkat lunak secara moduler, oleh karena itu perlu dibuat modul-modul yang saling berkaitan satu sama lain. Setiap modul dalam RSL mendefinisikan sebuah kelas. Secara umum, suatu kelas dalam RSL dapat berbentuk[6] :

id =
  class
    $deklarasi_1$
    .
    .
    .
    $deklarasi_n$
  end

untuk $n \geq 0$, di mana suatu deklarasi dapat diawali dengan kata kunci **type, value, axiom** yang mengindikasikan jenis deklarasi yang akan muncul, diikuti dengan sejumlah definisi, yang dipisahkan dengan koma.

Tipe adalah sekumpulan nilai yang berkaitan secara logis. Kata kunci **type** berarti mendeklarasikan semua tipe bentukan yang akan dipakai di dalam kelas tersebut. Di dalam RSL, sudah tersedia beberapa tipe yang bisa langsung digunakan. Untuk mendefinisikan sebuah tipe yang baru, ada dua macam cara yang dapat dilakukan, yaitu :

1. Secara abstrak, dengan hanya menyebutkan nama untuk tipe tersebut tanpa operator yang terdefinisi sebelumnya.
2. Dengan menggunakan tipe – tipe yang sudah terdefinisi sebelumnya

Sebuah nilai pada suatu kelas dapat diberi nama dalam deklarasi nilai dengan diawali kata kunci **value.** Bentuk sederhananya adalah :

id : type_expr

Identifier id digunakan untuk menyatakan suatu nilai dalam tipe yang direpresentasikan oleh ekspresi tipe. Ada dua cara yang dapat digunakan untuk mendefinisikan suatu nilai dari suatu identifier, yaitu :

1. Secara eksplisit, atau langsung setelah deklarasinya
2. Secara implisit, atau dengan menggunakan aksioma.

Aksioma menyatakan sifat – sifat khas dari identifier / nama dari suatu nilai. Suatu aksioma dideklarasikan dengan didahului kata kunci **axiom**. Setiap aksioma adalah suatu ekspresi Boolean, yang harus menghasilkan nilai benar.

Suatu fungsi adalah pemetaan dari nilai – nilai yang bertipe tertentu menjadi nilai – nilai yang bertipe lain. Secara umum, ada dua jenis fungsi yaitu fungsi total dan fungsi sebagian. Suatu fungsi *f*, yang memetakan nilai – nilai bertipe T_1 ke nilai – nilai bertipe T_2 disebut fungsi total jika untuk setiap nilai di T_1, *f* mengembalikan sebuah nilai yang unik di T_2. Sedangkan fungsi *f* disebut parsial jika *f* tidak dapat mengembalikan suatu nilai di T_2 atau aplikasi yang berbeda dari *f* akan menghasilkan nilai yang berbeda pula.

Ekspresi dari fungsi total dapat berbentuk :

$\text{type\_expr}_1 \rightarrow \text{type\_expr}_2$

Sedangkan ekspresi dari fungsi sebagian berbentuk :

$\text{type\_expr}_1 \xrightarrow{\infty} \text{type\_expr}_2$

Contoh penulisan fungsi, misalnya pada basis data pemilihan umum, yaitu :

value

register : Person x Database → Database
check : Person x Database → Bool
number : Database → Nat

atau pada suatu sistem koordinat :

value

distance : Position x Position (real

Ada tiga macam tipe data pada RSL, yaitu : tipe abstrak, tipe terdefinisi, dan tipe bentukan. Tipe abstrak adalah yang paling sederhana, karena cukup dengan menyebutkan nama tipe tersebut tanpa perlu didefinisikan. Tipe terdefinisi adalah tipe yang sudah tersedia di dalam RSL. Tipe bentukan adalah tipe yang dibentuk dari gabungan tipe – tipe yang sudah terdefinisi sebelumnya. Penggabungan ini dilakukan dengan melakukan operasi *product*, fungsi, himpunan, *list*, *map*, subtipe atau varian.

Tipe terdefinisi pada RSL adalah Bool, Int, Real, Char, Nat, Text dan Unit. Bool adalah tipe boolean yang hanya mengandung nilai benar atau salah. Int adalah bilangan bulat. Real adalah bilangan real termasuk bilangan desimal. Char adalah karakter yang diawali dan diakhiri dengan ‘. Nat adalah subtipe dari Int, yaitu bilangan cacah (bilangan bulat tak negatif). Text adalah string karakter yang diawali dan diakhiri dengan “. Tipe Unit hanya mempunyai 1 nilai yaitu ‘()’, yang dipakai pada spesifikasi yang bergaya imperatif dan konkuren.

Sebuah himpunan adalah sekumpulan nilai yang berbeda dengan tipe yang sama, yang tak berurutan dan mungkin kosong. Bentuk umum suatu ekspresi himpunan adalah

T – **set**

yang berarti himpunan berhingga dengan elemen – elemen bertipe *T*, atau

T – **infset**

yang berarti himpunan tak berhingga dengan elemen – elemen bertipe *T*.

Ekspresi nilai himpunan dapat direpresentasikan sebagai berikut :

1. Ekspresi himpunan terenumerasi, yaitu cara penulisan secara eksplisit dengan bentuk umum [value_expr$_1$, ..., value_expr$_n$] dengan n ≥ 0 di mana value_expr$_i$ mempunyai tipe maksimal yang sama
2. Ekspresi himpunan bernarasi (*comprehended*), yaitu cara penulisan secara implisit dengan memberikan predikat yang mendefinisikan anggota himpunan tersebut, mempunyai bentuk umum [value_expr$_1$| typing$_1$, ..., typing$_n$ • value_expr$_2$] dengan n ≥ 1 di mana value_expr$_2$ pasti bernilai Boolean
3. Ekspresi himpunan berjangkauan, yaitu cara penulisan dengan memberikan himpunan bilangan dalam jangkauan dari batas bawah sampai batas atas,

dengan bentuk umum [value_expr$_1$ ... value_expr$_2$], dimana value_expr$_1$ dan value_expr$_2$ merupakan ekspresi integer.

Sebuah list adalah sekumpulan nilai yang bertipe sama dan memungkinkan adanya duplikasi maupun kosong. Bentuk umum ekspresi tipe list adalah *type_expr** yang merepresentasikan list berhingga, atau *type_expr*$^\omega$ yang merepresentasikan list tak berhingga. Ekspresi nilai list dapat direpresentasikan sebagai berikut :

1. Ekspresi list terenumerasi, yaitu penulisan ekspresi secara eksplisit dengan bentuk umum ⟨value_expr$_1$, ..., value_expr$_n$⟩ dengan n ≥ 0 di mana value_expr$_i$ mempunyai tipe maksimal yang sama.
2. Ekspresi list bernarasi (*comprehended*), yaitu penulisan ekspresi secara implisit dengan memberikan predikat yang mendefinisikan anggota list tersebut, mempunyai bentuk umum ⟨value_expr$_1$| typing$_1$, ..., typing$_n$ • value_expr$_2$⟩ dengan n ≥ 1 di mana value_expr$_2$ pasti bernilai Boolean.
3. Ekspresi list berjangkauan, yaitu penulisan ekspresi dengan memberikan list bilangan dalam jangkauan dari batas bawah sampai batas atas, dengan bentuk umum ⟨value_expr$_1$ ... value_expr$_2$⟩, di mana value_expr$_1$ dan value_expr$_2$ merupakan ekspresi integer.

Map adalah suatu struktur yang mirip tabel, yang memetakan nilai – nilai dari suatu tipe menjadi nilai – nilai dari tipe yang lain. Nilai – nilai yang didefinisikan oleh suatu map disebut sebagai *domain* dari map tersebut, sedangkan nilai – nilai yang dipetakan oleh map tersebut disebut sebagai *range*. Bentuk umum ekspresi tipe pemetaan adalah type_expr$_1$ $\xrightarrow{m}$ type_expr$_2$. Ekspresi nilai map dapat direpresentasikan sebagai berikut:

1. Ekspresi pemetaan terenumerasi, yaitu cara penulisan secara eksplisit dengan bentuk umum [value_expr$_1$ |→ value_expr$_1$', ..., value_expr$_n$ |→ value_expr$_n$'] dengan n ≥ 0 .
2. Ekspresi pemetaan bernarasi (*comprehended*), yaitu cara penulisan secara implisit dengan memberikan predikat yang mendefinisikan anggota list tersebut, mempunyai bentuk umum [value_expr$_1$ |→ value_expr$_2$| typing$_1$, ..., typing$_n$ • value_expr$_3$] dengan n ≥ 1 di mana value_expr$_3$ pasti bernilai Boolean.

4. TEORI BASIS DATA

Sebuah sistem basis data pada dasarnya adalah sebuah sistem penyimpanan data secara komputerisasi, atau dengan kata lain, sebuah sistem komputerisasi yang tujuan utamanya adalah untuk mengelola informasi dan untuk membuat informasi tersebut dapat digunakan pada saat dibutuhkan. Sebuah sistem basis data akan melibatkan empat komponen utama, yaitu data, perangkat keras, perangkat lunak dan pengguna[3].

Sebuah sistem basis data dapat merupakan sistem *single-user* yang berarti hanya ada satu pengguna yang dapat mengakses basis data pada saat tertentu, atau merupakan sistem *multi-user* yang berarti basis data akan dapat diakses oleh beberapa pengguna pada saat yang bersamaan.

Secara umum, data di dalam basis data akan dapat diintegrasi dan *shared*. Dengan diintegrasi maka berarti basis data dapat diasumsikan sebagai gabungan dari beberapa file data yang berbeda. Dengan *sharing* maka sebuah data dapat digunakan bersama-sama oleh pengguna yang berbeda.

Di antara pengguna sistem dan basis data ada perangkat lunak yang digunakan, yaitu *database manager* atau *database management system* (DBMS). Setiap permintaan dari pengguna untuk mengakses basis data, seperti menambah atau menghapus tabel atau file, mengambil data dari basis data, dan merubah data dapat ditangani oleh DBMS.

Pengguna dari sistem dapat dibedakan menjadi tiga kelas, yaitu *application programmer*, yaitu orang yang membuat sebuah program yang menggunakan basis data tersebut, *end user*, yaitu orang berinteraksi dengan sistem melalui terminal atau *online* workstation, dan *database administrator*, yaitu orang bertugas untuk mengelola basis data.

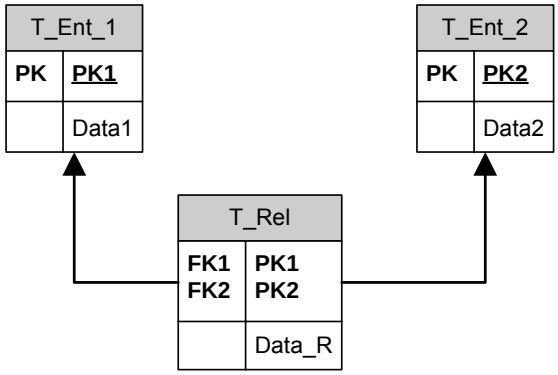
Sebuah basis data terdiri dari beberapa data yang bersifat tetap yang digunakan oleh sistem aplikasi. Sebuah basis data terdiri dari entitas, yaitu objek yang dapat dibedakan dari objek lain yang terwakili dalam basis data, dan *relationship* yang menghubungkan entitas – entitas tersebut. Selain itu, ada juga atribut yang merupakan kelengkapan dari entitas maupun relationship.

Keuntungan dari penggunaan basis data adalah dapat mengurangi redundancy, dapat menghindari inkonsistensi, data dapat di-*shared*, standard dapat ditetapkan oleh administrator, pembatasan keamanan dapat diaplikasikan, integritas dapat dipelihara,

dan pertentangan kebutuhan dapat diseimbangkan.

5. PEMODELAN BASIS DATA DENGAN METODE FORMAL

Sebuah basis data dapat mempunyai beberapa tabel yang saling berelasi. Sebagai contoh untuk memodelkan relasi suatu tabel, maka digunakan sebuah basis data yang mempunyai dua buah entitas dan sebuah hubungan (*relationship*). Setiap entitas mempunyai sebuah *primary key* dan sebuah data, sedangkan *relationship* menjadi tabel tersendiri yang mempunyai sebuah *primary key* dan dua buah *foreign key* yang terhubung ke dua entitas tersebut. Setiap entitas akan menjadi sebuah tabel tersendiri, dan relasi juga akan mempunyai sebuah tabel tersendiri. Diagram hubungan antar entitas dapat digambarkan seperti pada gambar V.1



Gambar V.1. Diagram Hubungan Antar Tabel

Fungsi-fungsi yang umum terdapat dalam tabel adalah:

1. Fungsi untuk memastikan bahwa data yang akan dimasukkan belum ada di dalam basis data, sehingga tidak terjadi redundancy data (*is_wf_database*),
2. Fungsi untuk memasukkan suatu *record* baru ke dalam basis data (fungsi *insert*),
3. Fungsi untuk menentukan bahwa kunci tertentu sudah ada di dalam basis data (fungsi *defined*),
4. Fungsi untuk menghapus suatu data berdasarkan kunci tertentu (fungsi *remove*),
5. Fungsi untuk meng-*update* data berdasarkan kunci tertentu (fungsi *update*),
6. Fungsi lain yang didefinisikan tersendiri oleh pengguna

Dari diagram hubungan antar tabel, maka dapat dibentuk pemodelan formal,

dimana setiap entitas dapat menjadi sebuah *class*, sementara setiap atribut dapat menjadi sebuah *type*. Di bagian ini akan diberikan pemodelan abstrak yang umum sehingga untuk setiap entitas dapat digambarkan seperti pada Gambar V.2:

```
scheme T_ENT_X =
/* X menandakan nomor entitas */
        class
                type
/* Definisikan atribut pk dan data sebagai type */
                PK_X, Data_X, T_Ent_X,

        /* Definisikan bahwa sebuah record EntitasX */
        /* terdiri dari PK_X dan Data_X */
                        Ent_X = PK_X x Data_X

                value
                        /* Konstanta yang berisi tabel kosong */
                        empty : T_EntX,

                        /* Fungsi untuk memasukkan suatu record (fungsi no. 2)*/
                        insert_X : PK_X x Data_X × T_Ent_X → T_Ent_X,

                        /* Fungsi untuk mencari apakah suatu key tertentu */
                        /* sudah ada di dalam tabel (fungsi no. 3) */
                        defined_X : PK_X × T_Ent_X → Bool,


                        /* Fungsi untuk menghapus suatu record berdasarkan */
                        /* key tertentu (fungsi no. 4) */
                        remove_X : PK_1 × T_Ent_X → T_Ent_X,

                        /* Fungsi untuk meng-update suatu record berdasarkan */
                        /* key tertentu (fungsi no 5) */
                        update_X : PK_X x Data_X × T_Ent_X → T_Ent_X

                axiom forall pkx:PK_X, dx:Data_X, tex:T_Ent_X, rex:Ent_X •

                /* Untuk memasukkan data */
                        defined_X(pkx,insert_X(rex,tex)) ≡ pkx = pkx1 ∨ defined_X(pkx,tex),

                        /* Untuk mencari apakah key sudah ada di tabel */
                        defined_X(pkx,defined_X(pkx1,tex)) ≡ pkx = pkx1 ∨ defined_X(pkx,tex),

                        /* Untuk menghapus data */
                        remove_X(pkx,remove_X(pkx,tex)) ≡
                                if pkx=pkx1 then remove_X(pkx,tex),

                        /* Untuk mengubah data */
                        remove_X(pkx,update_X(rex,tex)) ≡
                                if pkx=pkx1 then remove_X(pkx,tex) else
                                        update_X(rex,remove_X(pkx,tex)),

        end
```

Gambar V.2. Kode Formal Untuk Pemodelan *Abstract Applicative* Entitas

Berdasarkan pemodelan *abstract applicative*, maka dapat dibentuk pemodelan *concrete applicative* dengan menambahkan tipe dari tabel yang diinginkan. Tipe data untuk tabel dapat menggunakan tipe list, tipe set, tipe map. Pada pemodelan di Penelitian ini dipilih tipe himpunan (set) karena pemodelannya tidak terlalu sulit. Maka untuk setiap entitas dapat dibuat menjadi konkret seperti pada Gambar V.3.

```
scheme T_ENT_X =
/* X menandakan nomor entitas */
    class
        type
            /* Definisikan atribut key dan data sebagai type */
            PK_X, Data_X,


            /* Definisikan bahwa sebuah record EntitasX */
            /* terdiri dari PKX dan DataX */
            RecEnt_X = PK_X x Data_X,

            /* Definisikan tipe dari tabel DataEnt1 sebagai set */
            T_Ent_X = {| tex: RecEnt_X –set • is_wf_T_Ent_X(tex) |}

        value
            /* Fungsi untuk mengecek redundancy dalam tabel */
            is_wf_T_Ent_X : RecEnt_X -set → Bool,

            empty : T_Ent_X,

            /* Fungsi untuk memasukkan suatu record (fungsi no 3)*/
            insert_X : PK_X x DataX × T_Ent_X → T_Ent_X,

            /* Fungsi untuk mencari apakah suatu key tertentu */
            /* sudah ada di dalam tabel (fungsi no 4) */
            defined_X : PK_X × T_Ent_X → Bool,

            /* Fungsi untuk menghapus suatu record berdasarkan */
            /* key tertentu (fungsi no 5) */
            remove_X : PK_X × T_Ent_X → T_Ent_X,

            /* Fungsi untuk meng-update suatu record berdasarkan */
            /* key tertentu (fungsi no 6) */
            update_X : RecEnt_X × Data_X × T_Ent_X → T_Ent_X

        axiom forall kx:PK_X, dx:Data_X, tex:T_Ent_X, rex:RecEnt_X –set •

            is_wf_T_Ent_X(rex) ≡ ( ∀ kx : PK_X, dx1,dx2 : Data_X •
                ((kx,dx1) ∈ tex ∧ (kx,dx2) ∈ tex) ⇒ dx1 = dx2),

            empty = {},

            insert_X(kx,dx,tex) as tex1
            post (kx,dx) ∈ tex1
            pre ~defined_X(kx,tex),

            defined_X(kx,tex) ≡ ( ∃ dx : Data_X • (kx,dx) ∈ tex),

            remove_X(kx,tex) as tex1
            post ~defined_X(kx,tex1)
            pre defined_X(kx,tex),

            update_X((kx,dx),dx1,tex) as tex1
            post (kx,dx1) ∈ tex1 ∧ (kx,dx) ∉ tex1
            pre defined(kx,tex)

    end
```

Gambar V.3 Kode Formal Untuk Pemodelan *Concrete Applicative* Entitas

6. STUDI KASUS

Pada Penelitian ini, studi kasus yang akan digunakan adalah sistem pemesanan *resources* yang dilakukan dengan mengunjungi situs web sistem pemesanan tersebut.

Tahapan yang terdapat dalam studi kasus ini adalah pelanggan melakukan pemesanan *resources*, mengkonfirmasi pemesanan, mengeluarkan (*issued*) tiket, menggunakan *resources*. Selain itu ada juga tahapan jika pelanggan ingin melakukan pembatalan pemesanan.

Pada tahap pemesanan *resources*, pelanggan harus memilih atribut yang penting yang merupakan ciri dari sistem tersebut, biasanya merupakan jadwal. Contoh pada sistem bioskop, yaitu pelanggan harus memilih lokasi bioskop, judul film, dan tanggal. Contoh pada sistem penerbangan, yaitu pelanggan harus memilih kota asal, kota tiba, dan tanggal keberangkatan.

Jadwal dapat tidak tersedia dikarenakan perusahaan tidak mempunyai jadwal pada tanggal tersebut, atau karena pemesanan secara *online* sudah tidak dapat dilakukan karena pemesanan dilakukan kurang dari waktu yang sudah ditentukan.

Jika jadwal tidak tersedia karena perusahaan tidak mempunyai jadwal pada tanggal tersebut, maka sistem akan menampilkan pesan bahwa perusahaan tidak mempunyai jadwal pada tanggal tersebut. Tetapi, jika jadwal tidak tersedia karena pelanggan melakukan pemesanan kurang dari waktu yang sudah ditentukan, maka pelanggan diberi pesan untuk menghubungi *customer service* untuk melakukan pemesanan secara langsung. Jika jadwal tersedia, maka sistem akan menampilkan jadwal lengkap dengan beberapa atribut yang perlu diketahui oleh pelanggan.

Setelah pelanggan memilih jadwal yang diinginkan, maka sistem akan memeriksa apakah *resources* (kapasitas) masih ada atau tidak. Jika *resources* (kapasitas) sudah tidak ada, maka pelanggan diberi pesan bahwa kapasitas sudah penuh. Kapasitas biasanya dilebihi ±10%, hal ini dilakukan untuk mengatasi jika ada yang membatalkan pemesanan.

Jika kapasitas masih ada, maka sistem akan menanyakan data pelanggan (nomor KTP/Passport), dan nama pelanggan. Selanjutnya sistem akan menyimpan data pemesanan ke dalam basis data transaksi dengan status ***booked*** dan memberikan nomor *booking* kepada pelanggan.

Pada tahapan konfirmasi, pelanggan harus memasukkan nomor *booking*. Kemudian sistem akan memeriksa apakah nomor *booking* tersebut terdapat dalam basis data atau tidak. Jika tidak ada, maka sistem akan memberikan pesan bahwa nomor *booking* tersebut salah. Jika ada, maka sistem akan melakukan pemeriksaan apakah kapasitas masih mencukupi atau tidak.

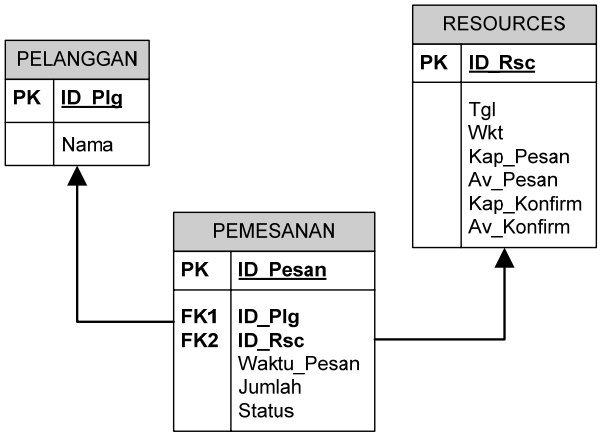
Pada tahapan ini, jumlah yang dapat melakukan konfirmasi tidak boleh melebihi kapasitas sesungguhnya. Jika kapasitas tidak mencukupi, maka pelanggan diberi pesan bahwa kapasitas sudah penuh, dan status pelanggan adalah *waiting list*, kemudian pelanggan menentukan apakah ingin tetap melakukan pesanan atau tidak. Jika tidak, maka pelanggan dianggap melakukan pembatalan pesanan, jika pelanggan ingin tetap melanjutkan konfirmasi, maka status pelanggan dalam basis data diubah menjadi *waiting list* dan pelanggan akan diberitahukan jika ada tempat kosong. Setelah itu, sistem akan melanjutkan ke tahapan pembayaran. Pembayaran hanya dapat dilakukan melalui kartu kredit. Maka, sistem akan menanyakan nomor kartu kredit pelanggan.

Setelah melakukan konfirmasi, maka tahapan berikutnya adalah mencetak tiket. Untuk mencetak tiket, maka pelanggan dapat memasukkan nomor *booking*, kemudian nomor tersebut akan diperiksa keabsahannya. Jika nomor tersebut tidak valid, maka akan dikeluarkan pesan kepada pelanggan bahwa nomor *booking* tersebut tidak valid, sedangkan jika valid maka sistem akan menampilkan tiket yang berisi nomor tiket, jadwal, dan nama pelanggan. Kemudian, pelanggan dapat mencetak tiket tersebut.

Setelah pelanggan mendapatkan tiket, maka tahapan selanjutnya adalah tahapan menggunakan *resources* tersebut. Pada tahapan ini, pelanggan memberikan tiket yang sudah dicetak kepada petugas untuk dapat menggunakan *resources* tersebut.

Sistem pemesanan tempat akan mempunyai beberapa *use case*, yaitu *use case* untuk pelanggan yang terdiri dari memesan *resources* (UCP1), mengkonfirmasi pemesanan (UCP2), mengeluarkan (*issued*) tiket (UCP3), menggunakan *resources* (UCP4), membatalkan pemesanan (UCP5). Sedangkan *use case* untuk administrator akan terdiri dari memasukkan data pelanggan (UCA1), menghapus data pelanggan (UCA2), mengubah data pelanggan (UCA3), memasukkan data *resources* (UCA4), menghapus data *resources* (UCA5), mengubah data *resources* (UCA6).

Hubungan antar tabel untuk kasus ini dapat digambarkan pada Gambar VI. 1.



Gambar VI.1 Diagram Hubungan Antar Tabel untuk Studi Kasus

7. PEMETAAN METODE BERORIENTASI OBJEK MENJADI METODE FORMAL

Dari diagram hubungan antar tabel pada bab 4, dapat dibuat tiga buah tabel, yaitu tabel Pelanggan, tabel Resources, dan tabel Pemesanan. Suatu entitas dapat menjadi sebuah ***class*** pada metode formal, sedangkan atribut dapat menjadi **tipe** pada metode formal. *Use Case* pada pemodelan berorientasi objek akan menjadi **fungsi** pada metode formal.

Pemetaan dari pemodelan berorientasi objek menjadi formal dapat diberikan pada tabel V.1 :

Tabel V.1. Pemetaan Metode Berorientasi Objek Menjadi Metode Formal

Tabel	**Use Case**	**Operasi**
Pelanggan	UCA1	- insert_plg - defined_plg
	UCA2	- remove_plg - defined_plg
	UCA3	- update_plg - defined_plg
Resources	UCA4	- insert_rsc - defined_rsc
	UCA5	- remove_rsc - defined_rsc
	UCA6	- update_rsc - defined_rsc
Pemesanan	UCP1	- memesan - defined - is_ada_jdw - is_enough_pesan
	UCP2	- konfirmasi

Tabel	**Use Case**	**Operasi**
		- defined - is_enough_konfirm - is_valid - want_wait issue_Rsc
	UCP3	- defined - use_Rsc
	UCP4	- defined - cancel_booked
	UCP5	- remove

Spesifikasi pada metode formal dimulai dengan mem bangun abstraksi sistem, maka disusun modul *abstract applicative* untuk PELANGGAN, RESOURCES, dan PEMESANAN. *Scheme* PELANGGAN merupakan representasi formal untuk tabel pelanggan yang mempunyai atribut ID_Plg dan Nama seperti pada Gambar VII.1.

```
scheme PELANGGAN = class
/* Untuk tabel pelanggan */
        type
                /* Definisikan tipe ID_Plg dan Nama yang diambil */
                /* dari atribut pada entitas Pelanggan pada ERD */
                ID_Plg, Nama, T_Plg,

                /* Definisikan bahwa sebuah record Pelanggan */
                /* terdiri dari ID_Plg dan Nama */
                RecPlg = ID_Plg x Nama

        value

                /* Fungsi-fungsi yang sama dengan template */
                empty : T_Plg,

                /* Fungsi untuk mencari apakah */
                /* suatu key ada di dalam tabel */
                defined_Plg : ID_Plg x T_Plg → Bool,

                /* Fungsi untuk memasukkan data pelanggan */
                insert_Plg: ID_Plg x Nama x T_Plg → T_Plg,

                /* Fungsi untuk menghapus data pelanggan */
                remove_Plg : ID_Plg x T_Plg → T_Plg,

                /* Fungsi untuk mengubah data pelanggan */
                update_Plg: ID_Plg x Nama x T_Plg → T_Plg

                axiom forall IDP,IDP1:ID_Plg, n:Nama, tp:T_Plg,rp:RecPlg •

                        /* Untuk memasukkan data */
                        defined_Plg(IDP,insert_Plg(rp,tp)) ≡ IDP = IDP1 ∨ defined_Plg(IDP,tp),

                        /* Untuk mencari apakah key sudah ada di tabel */
                        defined_Plg(IDP,defined_Plg(IDP1,tp)) ≡ IDP = IDP1 ∨ defined_Plg(IDP,tp),

                        /* Untuk menghapus data */
                        remove_Plg(IDP,remove_Plg(IDP,tp)) ≡
                                if IDP=IDP1 then remove_Plg(IDP,tp),

                        /* Untuk mengubah data */
                        remove_Plg(IDP,update_Plg(rp,tp)) ≡
                                if IDP=IDP1 then remove_Plg(IDP,tp) else
                                        update_Plg(rp,remove_Plg(IDP,tp)),

        end
```

Gambar VII.1 Implementasi *Abstract Applicative* Entitas Pelanggan

Setelah spesifikasi dalam *abstract applicative* disusun, maka selanjutnya adalah membuat spesifikasi menjadi konkret. Maka, perlu ditambahkan tipe yang akan digunakan. Dalam hal ini, digunakan tipe data **–set** (himpunan). Selain itu, perlu ditambahkan juga sebuah fungsi is_wf yang dapat menjamin bahwa setiap data dalam tabel tidak ada duplikasi.

Untuk tabel pelanggan ditambahkan fungsi is_wf_T_Plg untuk menjamin tidak adanya duplikasi data. Maka PELANGGAN akan menjadi seperti Gambar VII.2

```
scheme PELANGGAN = class
    type ID_Plg, Nama,
            RecPlg = ID_Plg x Nama,
            T_Plg = {| rp: RecPlg–set • is_wf_T_Plg(rp) |}
            /* Penulisan T_Plg menggunakan subtype */
            /* yang berarti T_Plg merupakan sebuah himpunan (set) */
            /* dari RecPlg yang memenuhi is_wf_T_Plg */

    value
            is_wf_T_Plg : RecPlg–set → Bool,
            empty : T_Plg,
            defined_Plg : ID_Plg x T_Plg → Bool,
            insert_Plg : ID_Plg x Nama x T_Plg → T_Plg,
            remove_Plg : ID_Plg x T_Plg → T_Plg,
            update_Plg: ID_Plg x Nama x T_Plg → T_Plg

    axiom forall tp:T_Plg, rp:RecPlg–set, IDP:ID_Plg, n,n1:Nama •

            /* Fungsi-fungsi yang sama dengan template */
            /* Memeriksa redundancy data */
            is_wf_T_Plg(tp) ≡
                    (∀ IDP:ID_Plg, n1,n2:Nama • ((IDP,n1) ∈ tp ∧ (IDP,n2) ∈ tp) ⇒ (n1)=(n2)),

            empty ≡ {},

            insert_plg(IDP,n,tp) as tp1
            post (IDP,n) ∈ tp1
            pre ~defined_Plg(IDP,tp),

            defined_Plg(IDP,tp) ≡ (∃IDP:ID_Plg,n: Nama • (IDP,n) ∈ tp),

            remove_Plg(IDP,tp) as tp1
            post ~defined(IDP,tp1)
            pre defined_Plg(IDP,tp),

            update_Plg((idp,n),n1,tp) as tp1
            post (IDP,n) ∉ tp1 ∧ (IDP,n1) ∈ tp1
            pre defined_Plg(IDP,tp)

    end
```

Gambar VII.2 Implementasi *Concrete Applicative* Entitas Pelanggan

Untuk implementasi *concrete imperative*, dapat diambil dari hasil *concrete applicative*, tapi diubah menjadi versi imperative dimana variabel sudah digunakan. Maka untuk tabel pelanggan menjadi seperti pada Gambar VII.3.

```
scheme PELANGGAN = class
    type ID_Plg, Nama,
        RecPlg = ID_Plg x Nama

    variabel
        tp = {| rp: RecPlg–set • is_wf_T_Plg(rp) |}

    value
        is_wf_T_Plg : RecPlg–set → Bool,
        empty : Unit write tp Unit,
        defined_Plg : ID_Plg → read tp Bool,
        insert_Plg : ID_Plg x Nama → write tp Unit,
        remove_Plg : ID_Plg → write tp Unit,
        update_Plg: ID_Plg x Nama → write tp Unit

    axiom forall rp:RecPlg-set, IDP:ID_Plg, n:Nama •

        empty() ≡ tp := {},

        insert_plg(IDP,n) ≡
            if ~defined_Plg(IDP)
            then tp := tp ∪ {(IDP,n)},

        defined_Plg(IDP) ≡ IDP ∈ tp,

        remove_Plg(IDP) ≡
            if defined_Plg(IDP)
            then tp := tp \ {(IDP,n) | n : Nama • true},

        /* Mengubah data pelanggan */
        update_Plg(IDP,n) ≡ if defined_Plg(IDP)
            then tp := remove_Plg(IDP) ∪ {(IDP,n)}

    end
```

Gambar VII.3 Implementasi *Concrete Imperative* Entitas Pelanggan

Sebagai sebuah basis data yang dapat diakses oleh beberapa pengguna sekaligus, maka harus menjaga kemungkinan terjadinya konkurensi. Pada umumnya, spesifikasi *concrete concurrent* dapat dibentuk dari spesifikasi *concrete imperative* dengan menambahkan definisi *channel* untuk setiap fungsi yang akan mengakses basis data, kemudian definisikan masukan dan keluaran *channel* dari setiap fungsi.

Maka, implementasi *concrete concurrent* untuk tabel pelanggan adalah seperti pada Gambar VII.4.

```
scheme PELANGGAN = class
    type ID_Plg, Nama,
        RecPlg = ID_Plg x Nama

    variable
        tp = {| rp: RecPlg–set • is_wf_T_Plg(rp) |}

    channel
            Empty : Unit,
            Insert_Plg: ID_Plg x Nama,
            Defined_Plg,Remove_Plg : ID_Plg,
            Defined_Plg_res : Bool,

    value
        /* Fungsi yang berlangsung di table */
        CTP : tp → in Empty, Insert_Plg, Update_Plg, Defined_Plg,  Remove_Plg out Defined_Plg_res
Unit,

        /* Fungsi yang berlangsung di interface user */
        empty : Unit → out Empty Unit,
        defined_Plg : ID_Plg → out Defined_Plg in Defined_Plg_res Bool,
        insert_Plg : ID_Plg x Nama → out Insert_Plg Unit,
        remove_Plg : ID_Plg → out Remove_Plg Unit,
        update_Plg : ID_Plg x Nama → out (Remove_Plg,Insert_Plg) Unit

    axiom forall ctp:tp, IDP:ID_Plg, n:Nama, rp:RecPlg-set •
        CTP(ctp) ≡
            Empty? ; CTP({}),
            []
            let (IDP,n) = Insert_plg? in CTP(ctp ∪ {(IDP,n)} end
            []
            let IDP = Defined_Plg? in Defined_Plg_res!(IDP ∈ ctp); CTP(ctp) end
            []
            let IDP = Remove_Plg? in CTP(ctp \ {(IDP,n) | n : Nama • true) end,

        empty() ≡ Empty!(),

        insert_plg(IDP,n) ≡ if ~defined_Plg(IDP) then Insert_Plg!(IDP,n),

        defined_Plg(IDP) ≡ Defined_Plg!IDP ; Defined_Plg_res?,

        remove_Plg(IDP) ≡ if defined_Plg(IDP) then Remove_Plg!IDP,

        update_Plg(IDP) ≡ if defined_Plg(IDP) then remove_Plg(IDP); insert_Plg(IDP,n)

    end
```

Gambar VII.4 Implementasi *Concrete Concurrent* Entitas Pelanggan

8. KESIMPULAN

Kesimpulan yang didapat setelah penelitian adalah :

1. Metode berorientasi objek dapat digunakan untuk membantu metode formal dalam melakukan analisis awal.
2. Metode formal mempunyai keunggulan dalam mengatasi masalah konkurensi.

9. DAFTAR PUSTAKA

[1]. Bauer, Robert T., “Formal Method Used In Software Verification”, 2000

[2]. Booch, Grady, James Rumbaugh and Ivar Jacobson, “The Unified Modelling Language User Guide”, 1999, Addison-Wesley

[3]. Date, C. J., “An Introduction To Database System 6th ed”, 1995, Addison-Wesley Publishing Company, Inc

[4.] Gomaa, Hassan, “Designing Concurrent, Distributed, And Real-Time Applications with UML”, 2000, Addison-Wesley

[5] Nugraheni, Cecilia E., “Diktat Kuliah Metode Formal”, 2005

[6] Pressman, Roger S., 2005, “Software Engineering : A Practitioner’s Approach 7th ed”, McGraw-Hill

[7]. Quoc Dang, and Richard Moore, “Formal Modelling of Future Demand Forecasting and Frequent Flyer Programs”, 1999, International Institute for Software Technology

[8]. Raise Method Group, “ The RAISE Development Method “, 1995, Prentice Hall International

[9]. Silberschatz, Abraham, Henry F. Korth, S. Sudarshan, “Database System Concepts 4th ed.”, 2002, The McGraw-Hill Companies Inc.