BAB II

LANDASAN TEORI

II.1 Konsep Pemrograman Berbasis Objek

Pemrograman Object Oriented(OO) adalah suatu konsep pemrograman yang menggunakan objek sebagai building block dalam pengembangan aplikasinya(Weisfeld, 2008). Dalam pemrograman OO, objek memiliki data dan behaviour. Data adalah atribut yang melekat pada objek, yang akan merepresentasikan state objek tersebut. Sedangkan behaviour adalah fungsi-fungsi yang dapat dilakukan oleh objek, yang selanjutnya merepresentasikan sifat-sifat atau perilaku objek.

Objek didefinisikan dalam suatu Kelas. Kelas inilah yang bertindak sebagai cetak-biru dalam pembentukan suatu objek. Objek dari suatu kelas akan mengikuti kontrak yang tertulis dalam definisi kelas tersebut, kontrak itu meliputi data dan behaviour yang didefinisikan kelas sehingga objek dapat disebut juga sebagai instance of a class.

Pemrograman OO memiliki beberapa konsep dasar yang digunakan dalam pengembangan aplikasi berbasis OO. Konsep-konsep tersebut antara lain adalah encapsulation, inheritance, polymorphism, dan composition.

II.1.1 Encapsulation

Sebuah objek memiliki atribut dan fungsi yang merepresentasikan state dan behaviour-nya. Encapsulation

adalah konsep menyembunyikan atribut suatu objek dari objek lain. Komunikasi antar objek harus dilakukan melalui antarmuka yang telah ditentukan dalam definisi/kontrak yang tertulis dalam Kelas. Antarmuka ini berupa fungsi-fungsi yang didefinisikan oleh kelas.

Sebagai contoh, Objek dari kelas A memiliki atribut x, atribut ini hanya dapat diakses oleh objek lain melalui fungsi setX() dan getX() yang didefinisikan oleh kelas A. Dengan begitu objek lain tidak perlu mengetahui detail implementasi yang diterapkan kelas A untuk mengakses data atribut x yang dimilikinya. Objek lain cukup mengetahui fungsi apa yang disediakan kelas A untuk mengakses atribut tersebut.

II.1.2 Inheritance

Keuntungan dari pemrograman OO adalah kemampuan code reuse, memanfaatkan kode yang telah ditulis untuk kegunaan lain, inheritance adalah salah satu penerapannya. Inheritance adalah suatu cara untuk membentuk Kelas baru, berdasarkan Kelas yang telah dibentuk. Kelas baru ini nantinya akan mewarisi atribut dan fungsi yang terdefinisi dalam Kelas yang telah dibentuk sebelumnya.

Kelas baru yang terbentuk ini selanjutnya disebut sebagai subclass atau kelas anak. Sedangkan Kelas pembentuk subclass disebut sebagai superclass atau kelas induk.

Atribut dan fungsi-fungsi yang diwarisi kelas anak dari kelas induknya dapat juga diubah atau ditambah. Ini menjadikan kelas anak sebagai versi kelas induk yang memiliki perilaku dan atribut yang lebih khusus.

Selain mewarisi atribut dan fungsi kelas induk, kelas anak juga mewarisi type dari kelas induk. Sehingga kelas

anak memiliki *is-a relationship* dengan kelas induk. Contohnya, Kelas Segitiga dan Lingkaran merupakan kelas yang dibentuk dari Kelas Bentuk2D. Kelas Segitiga dan Lingkaran akan mewarisi atribut dan fungsi dari Kelas Bentuk2D. Selain itu, Segitiga dan Lingkaran dapat dikatakan sebagai Bentuk2D, dengan kata lain 'is-a' Bentuk2D. Objek Segitiga memiliki type kelas Segitiga dan juga type kelas induknya, yaitu Bentuk2D. Ini yang selanjutnya disebut sebagai polymorphism(bahasa yunani: memiliki banyak bentuk).

II.1.3 Polymorphism

Polymorphism berkaitan erat dengan konsep inheritance, suatu kelas dapat memiliki banyak bentuk(polimorfis) dengan mewarisi bentuk atau type dari kelas induknya melalui inheritance.

Kelas anak yang merupakan spesialisasi dari kelas induk dapat mengimplementasi fungsi kelas induk melalui method overriding. Contohnya, kelas Bentuk2D memiliki fungsi hitungLuas() yang dapat di-override kelas turunannya. Kelas Lingkaran sebagai turunan kelas Bentuk2D dapat mengimplementasikan fungsi hitungLuas() sesuai kebutuhannya. Dalam bahasa pemrograman Java hal itu dapat ditulis sebagai berikut.

```
// Bentuk2D.java
public abstract class Bentuk2D{
 protected double area;
 public abstract double hitungLuas();
}
// Lingkaran.java
public class Lingkaran extends Bentuk2D{
 private double jejari;
 public Lingkaran(double jejari){
```

```
   super();
   this.jejari=jejari;
  }
@Override
 public double hitungLuas(){
   this.area = 3.14d * jejari * jejari ;
   return area;
  }
}
```

Dalam contoh diatas kelas Lingkaran mewarisi atribut area dan fungsi hitungLuas dari kelas Bentuk2D. Kelas Lingkaran mengimplementasikan fungsi hitungLuas() untuk menghitung luas lingkaran. Berikut contoh program main yang menunjukkan polymorphism.

```
//PolymorphDemo.java
public class PolymorphDemo{
 public static void main(String[] args){
   Bentuk2D bentuk = new Lingkaran(5);
   System.out.println(Luas:  + bentuk.hitungLuas());
  }
}
```

II.1.4 Composition

Selain Inheritance, Composition juga dapat digunakan sebagai mekanisme pembangunan objek. Merupakan hal yang wajar jika kita berpikir bahwa sebuah objek dibangun dari beberapa objek yang lain.

Composition adalah cara membangun sebuah objek baru dari beberapa objek lainnya. Berbeda dari Inheritance, dimana kelas anak memiliki is-a relationship dengan kelas induk, dalam Composition kelas bentukan(kelas baru) memiliki has-a relationship dengan kelas pembentuknya.

II.2 Enterprise Application Integration

II.2.1 Integrasi Sistem

Perangkat lunak telah menjadi sarana untuk mengotomatisasi proses bisnis dan administrasi di perusahaan. Setiap departemen di perusahaan memiliki aplikasi yang dibangun berdasarkan kebutuhannya. Tetapi seringkali aplikasi ini dibangun tanpa mempertimbangkan aspek integrasi dengan aplikasi dari departemen lainnya.

Seiring berjalannya waktu, cukup banyak pasokan informasi dari setiap departemen yang terkumpul. Pasokan informasi tersebut terisolasi di tiap departemen karena tidak adanya kemampuan aplikasi untuk berinteraksi dengan entitas luar. Data-data yang tersedia di tiap departemen berguna bagi pengambilan keputusan strategis perusahaan, namun ketidaktersediaan antarmuka integrasi menjadi salah satu kendala. Karena terisolasi, dibutuhkan banyak waktu dan tenaga untuk mengumpulkan informasi dari tiap departemen. Integrasi Sistem merupakan salah satu cara untuk mengatasi masalah tersebut.

II.2.2 Tahapan Dalam Integrasi Sistem

Menurut Mostafa H. Sherif (2010) Integrasi sistem dapat dicapai dengan melewati tahap-tahap berikut :

1. Interconnectivity
2. Functional Interoperability
3. Semantic Interoperability
4. Optimization and Innovation

II.2.2.1 Interconnectivity

Interconnectivity merupakan tahapan awal dari integrasi sistem. Tahapan ini bergantung pada infrastruktur telekomunikasi untuk memadukan aplikasi yang terpisah, sehingga dapat berdampingan dan saling bertukar informasi melalui gateway, adaptor, dan/atau transformer (sesuai dengan metodologi yang digunakan). Tahap ini dapat disebut juga "loose-integration" dimana setiap aplikasi dan fungsionalitasnya tidak terikat satu sama lain.

II.2.2.2 Functional Interoperability

Tahapan ini mengacu pada kemampuan untuk membuat setiap aplikasi dapat bekerja satu dengan yang lain secara langsung. Tahapan ini membutuhkan kompatibilitas teknis dan fungsional diantara protokol jaringan, aplikasi, dan format data. Keberhasilan dalam pertukaran data melibatkan, antara lain:

- Telecommunication networks
- Protocols for exchanging data
- Format of the messages exchanged
- Security procedures

Interoperabilitas dapat dicapai jika setiap aplikasi mengimplementasi antarmuka yang sama dan jika diperlukan, menambahkan fungsi yang tidak mempengaruhi interoperabilitas.

II.2.2.3 Semantic Interoperability

Konsistensi semantik dicapai ketika elemen data dan maknanya dapat digunakan setiap aplikasi bersama-sama (semantic

unification). Seluruh aplikasi harus menggunakan model yang sama, atau setidaknya satu model dapat dipetakan ke model yang lain tanpa menimbulkan perubahan makna.

II.2.2.4 Optimization and Innovation

Dalam tahap ini, integrasi sistem menjadi sarana perubahan yang sistematis dalam teknologi yang digunakan, organisasi, atau keduanya. Perlakuan terhadap informasi dapat di optimalkan melalui kontrol statistik dan aktifitas pelaporan yang lebih baik.

II.2.3 Aplikasi Enterprise

Doganata et al.(2010) menulis bahwa aplikasi enterprise adalah aplikasi perangkat lunak yang dibangun untuk mengelola operasi bisnis, aset, dan sumber daya perusahaan. Proses pembangunannya setidaknya meliputi empat kelompok: programmer yang fokus di koding business logic sebagai solusi masalah bisnis; database manager yang membangun data model dan mengelola pengaksesan, keamanan, dan penyimpanan data; GUI developer yang bertanggung jawab terhadap desain dan pembangunan widget untuk mempermudah pengguna; dan application integrator yang menangani integrasi dari aplikasi yang sudah ada (legacy system).

Dalam prinsipnya, tidak terdapat perbedaan antara aplikasi enterprise dan aplikasi reguler selain tujuan aplikasi dibangun. Aplikasi enterprise dibangun untuk memenuhi suatu tujuan bisnis yang spesifik.

Sifat dasar sasaran dan proses bisnis adalah perubahan, maka solusi perangkat lunak untuk setiap masalah bisnis pun berbeda-beda. Akibatnya, pengelolaan sistem enterprise secara keseluruhan menjadi sangat kompleks. Sebuah blue-

print sebagai template standar aplikasi untuk sebuah perusahaan dapat mengurangi kompleksitas tersebut. Untuk menjawab kebutuhan ini batasan, antarmuka, dan aturan untuk membangun aplikasi enterprise telah didokumentasikan. Blueprint mengenai bagaimana aplikasi enterprise harus dibangun ini disebut Enterprise Application Architecture. Arsitektur ini mendefinisikan sebuah struktur untuk mengorganisasi element dan sumber daya aplikasi, hubungan, dan peranannya dalam sebuah perusahaan.

Aplikasi Enterprise biasanya dibangun secara terpisah satu sama lain dan setiap aplikasi mengelola data mereka sendiri di sebuah sistem database. Hal ini menimbulkan heterogenitas dan tidak efisiennya data karena elemen data yang sama disimpan di database yang berbeda. Perbedaan struktur dan semantik data juga dapat terjadi. Salah satu tantangan dalam aplikasi enterprise saat ini adalah bagaimana mengintegrasikan berbagai aplikasi yang mungkin menggunakan OS dan database yang berbeda. Hohpe dan Woolf (2004) menulis bahwa Enterprise Application Integration adalah sebuah tugas untuk membuat berbagai aplikasi yang telah dibangun yang mungkin berjalan di berbagai platform dapat bekerja sama untuk saling berbagi data dan proses bisnis.

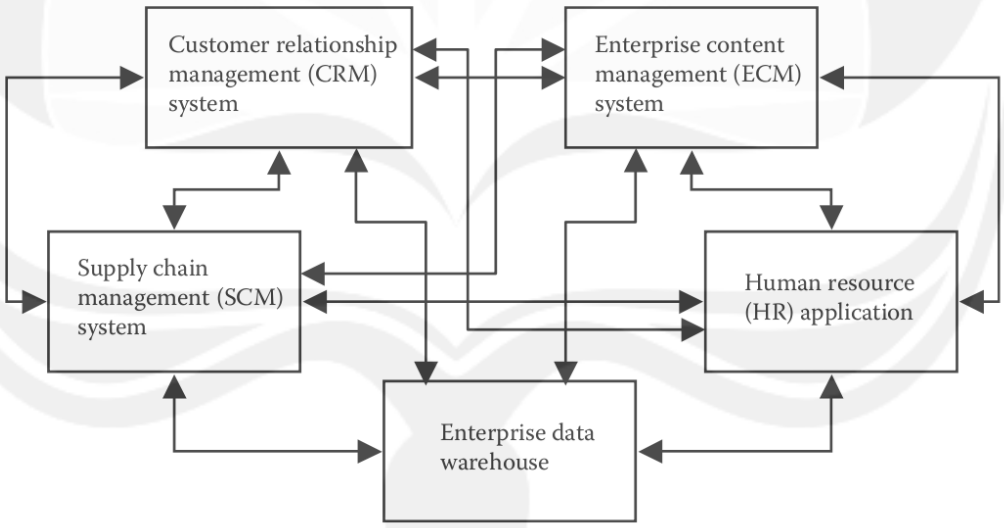
II.2.4 EAI Patterns

Integration pattern dapat dikelompokkan berdasarkan caranya terhubung, menjadi point-to-point integration dan hub-and-spoke integration. Pendekatan yang pertama adalah menghubungkan aplikasi-aplikasi yang ada secara langsung (directly connected), sedangkan yang kedua pertukaran message dilakukan melalui third-party application sebelum

mencapai tujuan akhirnya. Keduanya dapat direalisasikan dengan menggunakan pattern yang terdapat pada buku Enterprise Integration Patterns yang ditulis Hohpe dan Woolf (2004).

II.2.4.1 Point to point Integration

Point-to-point integration merupakan cara termudah untuk mengintegrasikan aplikasi yang saling terpisah. Seperti dapat terlihat di gambar II.1, setiap aplikasi terhubung secara langsung satu sama lain. Dibutuhkan antarmuka untuk setiap koneksi antar aplikasi. Jenis integrasi ini akan bekerja dengan baik jika tidak terlalu banyak aplikasi yang akan diintegrasikan. Jika ada N aplikasi yang akan diintegrasikan maka dibutuhkan N x N antarmuka.



Gambar II.1: Point to point Integration

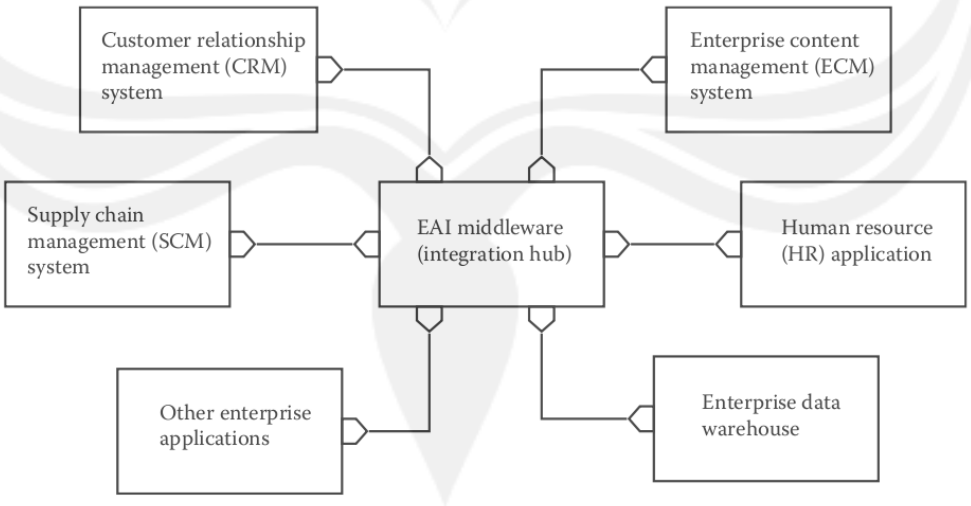
Masalah lain dalam integrasi ini adalah tidak adanya kemampuan untuk merespon perubahan secara cepat. Hal ini terjadi karena antarmuka terpatri di tiap aplikasi, sehingga perubahan infrastruktur informasi menimbulkan perubahan pula di tiap antarmuka aplikasi.

II.2.4.2 Message-oriented Integration

Dalam solusi message-oriented integration, setiap aplikasi saling berkomunikasi dengan menerima dan mengirim message melalui sebuah middleware yang mengelola message queue tiap aplikasi. Integrasi dilakukan dengan saling mengirim message ke queue, dan middleware akan memastikan message yang dikirim sampai ke tujuan.

II.2.4.3 Spoke-Hub Integration

Spokehub integration menghapuskan kebutuhan untuk menerjemahkan letak penerima pesan. Sebuah enterprise application middleware yang terpusat mengarahkan message ke tujuannya berdasarkan isi dan format pesan yang dikirim. Seluruh aplikasi terhubung ke pusat integrasi. Gambar II.2 memperlihatkan berbagai aplikasi enterprise yang terhubung ke sebuah middleware menggunakan spoke-hub integration.



Gambar II.2: Spoke-hub integration

II.3 RESTful Web Service

II.3.1 Web Service

Menurut D. Booth et. al. (2004) Web Service merupakan sebuah sistem perangkat lunak yang didesain untuk menunjang kemampuan interaksi antar mesin melalui sebuah jaringan. Sedangkan Richardson dan Ruby (2007) mengkategorikan web service sebagai programmable web, penyedia data bagi mesin dalam sebuah jaringan berskala Internet.

Terdapat tiga buah arsitektur web service yang sering digunakan: RESTful resource-oriented, RPC-style, dan REST-RPC hybrid.

II.3.1.1 RESTful, Resource-oriented Architecture

Arsitekur ini menerapkan prinsip-prinsip REST yang ditulis Roy T. Fielding (2000) dalam disertasinya yaitu addressability, statelessnes, uniform interfaces, representations, dan connectedness (hypermedia as the engine of application state). Contoh dari aplikasi yang dibangun dengan prinsip ini adalah:

- Service yang menggunakan Atom Publishing Protocol dan GData
- Amazon Simple Storage Service (S3)
- Sebagian besar web service yang dimiliki Yahoo!

II.3.1.2 RPC-style Architecture

Web service yang menggunakan RPC-style Architecture menerima sebuah envelope yang berisi data dari client, yang kemudian mengirimkan envelope yang sama kembali ke

client. Fungsi dan informasi yang ada disimpan di dalam envelope. Ada beberapa jenis envelope yang dapat digunakan, antara lain HTTP dan SOAP. Karena menggunakan konsep RPC, maka setiap sistem yang dibangun menggunakan konsep ini selalu menciptakan fungsi baru yang selalu berbeda di tiap implementasinya. Berbeda dengan prinsip REST yaitu uniform interface, yang menggunakan kumpulan fungsi yang sama untuk setiap resource-nya.

II.3.2 Resource-oriented Architecture (ROA)

ROA menerapkan prinsip-prinsip Representational State Transfer (REST) yang ditulis Roy T. Fielding (2000) dalam disertasinya. REST merupakan suatu arsitektur perangkat lunak untuk sistem hypermedia yang terdistribusi. Contoh sistem yang mengimplementasikan REST adalah world wide web.

Arsitektur REST memiliki beberapa dasar antara lain : keadaan dan fungsionalitas aplikasi diabstraksikan menjadi resource, setiap resource secara unik diakses dengan menggunakan URI, setiap resource menggunakan antarmuka yang sama untuk pengiriman state antara klien dan resource, menggunakan protocol yang mendukung client-server, stateless, cacheable, dan layered. Dengan menerapkan dasar-dasar dari REST, maka implementasi komponen menjadi lebih sederhana, meningkatkan kinerja, dan meningkatkan skalabilitas komponen server.

Batasan yang perlu diterapkan dalam implementasi REST seperti yang telah disebutkan beberapa di atas, yaitu:

1. Client-server

   Klien terpisah dari server. Dengan pemisahan ini, berarti sisi klien tidak perlu memiliki hubungan

secara langsung dengan tempat penyimpanan data, dan server tidak perlu mengurus antarmuka pengguna.

2. Stateless

   Keadaan klien dalam pemanggilan layanan di server tidak diperlukan. Hal ini berarti tiap permintaan yang dikirimkan klien berisi semua informasi yang dibutuhkan server untuk memberikan layanan yang diminta oleh klien.

3. Cacheable

   Klien dapat menyimpan response yang dikirimkan server. Sehingga, response yang dikirimkan server harus didefinisikan cacheable untuk mencegah klien menggunakan data yang dikirimkan pada response tersebut dengan tidak sah.

4. Uniform Interface

   Antarmuka yang seragam, menyederhanakan arsitektur Representational State Transfer (REST).

5. Layered system

   Sebuah klien tidak bisa mengakses langsung ke server secara biasa. Klien harus melalui lapisan-lapisan sistem yang diperlukan untuk mengakses server.

6. Code on demand

   Server dapat mengubah fungsionalitas klien dengan mengirimkan suatu urutan logika yang dapat dikerjakan klien.

II.4 NetKernel 4 Standard Edition

II.4.1 Deskripsi

NetKernel adalah sebuah platform perangkat lunak yang menerapkan prinsip Resource Oriented Computing(ROC). ROC memungkinkan arsitektur dan implementasi(code) decoupled.

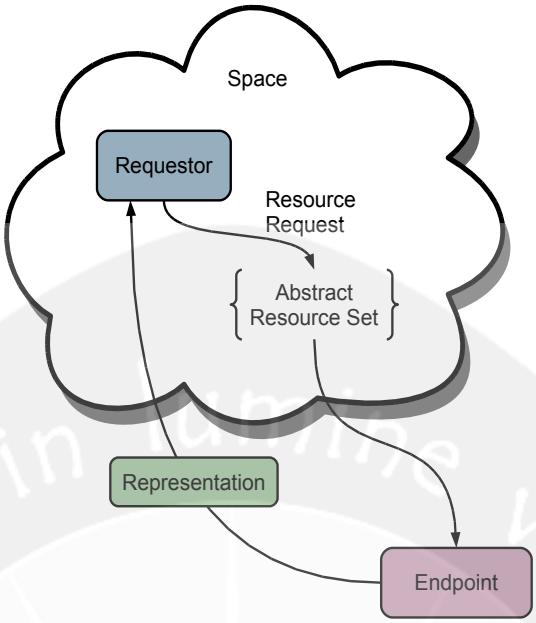
Resource Oriented Computing menyediakan beberapa konsep sederhana, – space, resource, endpoint, request, yang digunakan dalam melakukan komposisi arsitektur dari sebuah aplikasi.

- Request diproses oleh service endpoint yang terdapat di dalam space.
- Endpoint dapat melakukan request ke endpoint yang lain.
- Endpoint bersifat dinamis.

II.4.2 Konsep Resource Oriented Computing

Resource Oriented Computing adalah sebuah abstraksi komputasi yang bersifat umum. ROC dapat dikatakan sebagai generalisasi dari konsep REST dan Unix. Seperti yang dapat dilihat di diagram II.3, dalam ROC informasi dimodelkan dengan Resource. Contohnya, sebuah Sistem Informasi Seminar akan memiliki sebuah resource yang memodelkan informasi mengenai peserta seminar, dan resource yang memodelkan informasi pembicara seminar.

Setiap Resource dapat di akses melalui identifer. Uniform Resource Identifier (URI) biasanya digunakan sebagai identifier sebuah resource. Contohnya untuk resource pembicara digunakan URI *res:/seminar/pembicara* sedangkan untuk peserta *res:/seminar/peserta*.



Gambar II.3: Resource Oriented Computing

II.4.2.1 Space

Space menyediakan context yang digunakan untuk melakukan resolve terhadap suatu request. Di dalam NetKernel, cara request di resolve dan di evaluasi dapat dikonfigurasi secara deklaratif melalu space configuration.

II.4.2.2 Endpoint

Endpoint adalah *physical code* yang melakukan proses terhadap resource, dan berpotensi mengembalikan sebuah *resource representation*. Endpoint diklasifikasi berdasarkan kegunaannya, yaitu

- Transport, merupakan *event handler* yang menginisialisasi request berdasarkan suatu event.
- Accessor, bertugas memberikan response terhadap request yang masuk, dapat membuat request baru ke resource lain.

- Transreptor, mengubah informasi dari satu jenis representasi ke lainnya. Transreptor hanya merubah representasi fisik dari informasi, sehingga tidak ada informasi yang hilang.
- Overlay, bertindak sebagai *bridge* antar *space*

II.4.2.3 Resource

Resource adalah abstraksi dari sebuah informasi, seperti daftar mahasiswa, pesan email, tagihan mahasiswa, dan berbagai jenis informasi lainnya. Resource dapat diakses melalui identifier, dan akan mengembalikan sebuah representasi ketika diakses. Representasi merupakan nilai konkret dari sebuah resource.

II.4.2.4 Request

Sebuah request terdiri dari identifier dan verb. Identifier menentukan informasi resource, sedangkan verb menentukan tindakan yang akan dilakukan terhadap resource. Identifer adalah sebuah token yang mengidentifikasi resource. NetKernel menggunakan URI sebagai resource identifier-nya. Tindakan yang akan dilakukan terhadap resource ditentukan dari verb, dalam NetKernel verb yang digunakan antara lain, SOURCE, SINK, NEW, EXISTS, dan DELETE. Verb NetKernel ini berbeda dengan yang digunakan oleh protocol HTTP, yaitu GET, PUT, POST, HEAD, dan DELETE.

II.4.3 Request Processing di NetKernel

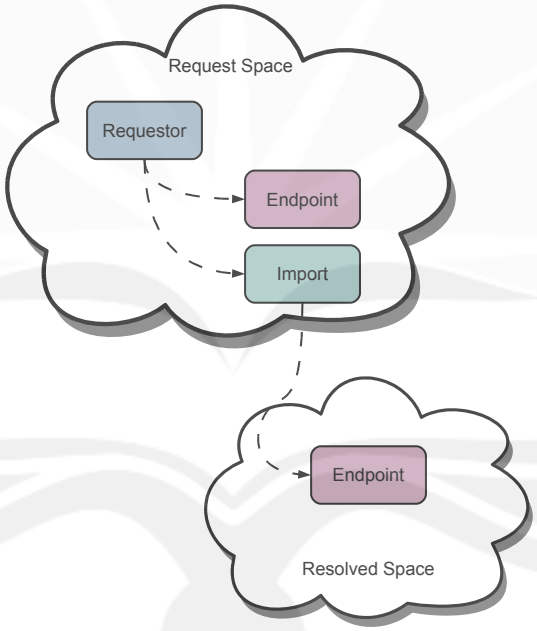
Bagian ini akan menjelaskan bagaimana Netkernel meng-implementasi pemrosesan Request ROC. Bagian Logical Request Processing akan menjelaskan abstraksi dari per-

spektif logical, sedangkan bagian Physical Request Processing menjelaskan implementasi low-level mekanisme request processing yang dilakukan oleh NetKernel.

II.4.3.1 Logical Request Processing

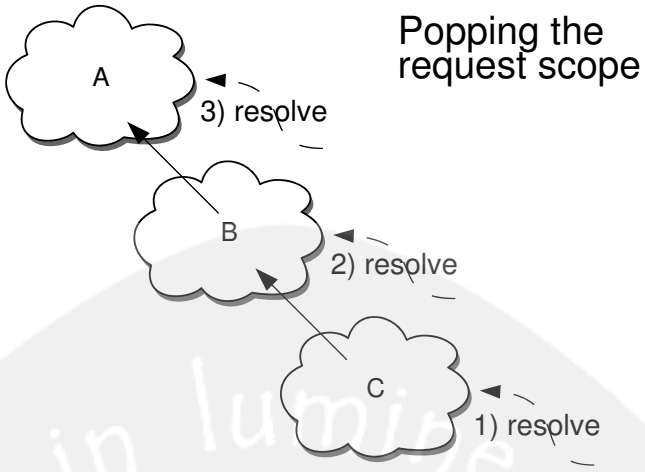
Dalam perspektif ini Request Processing terbagi menjadi dua fase yaitu Request Resolution dan Request Evaluation.

Ketika memasuki fase Request Resolution proses pencarian dilakukan untuk mendapatkan endpoint yang akan menerima request yang sedang diproses. Request Delegation dapat dilakukan dalam proses ini, sehingga proses pencarian endpoint dapat di delegasikan ke space yang lain.



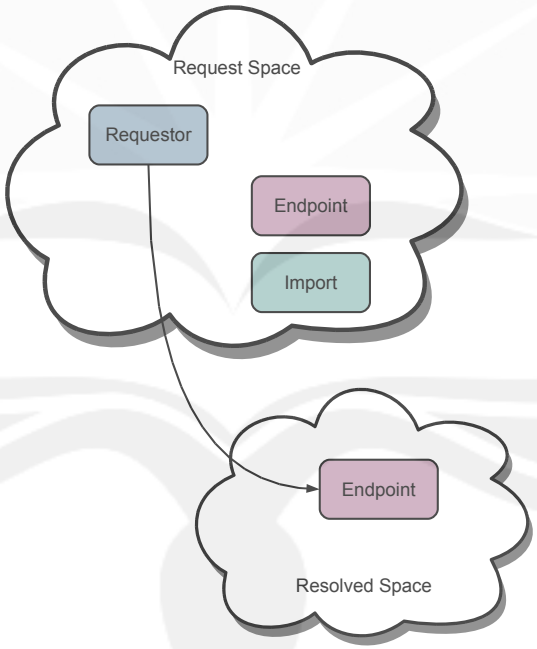
Gambar II.4: Request Delegation di NetKernel

Jika tidak ditemukan resolusi terhadap request yang diproses maka NetKernel akan mencari di lingkup terdekat dari request yang diproses. Proses ini akan terus berlanjut sampai endpoint ditemukan. Jika tidak ada endpoint yang memenuhi request, proses ini akan menampilkan error.



Gambar II.5: Perpindahan Scope(lingkup) Request

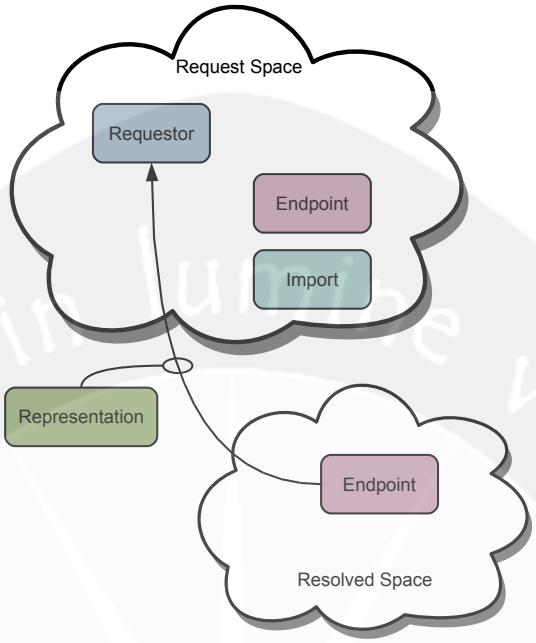
Setelah ditemukan Endpoint dalam proses Request Resolution, maka request akan diproses oleh endpoint tersebut dalam proses Request Evalution.



Gambar II.6: Request Evaluation

Endpoint dapat melakukan beberapa hal ketika memproses request, antara lain menciptakan sebuah resource lalu mengembalikannya ke requestor. Seringkali request berisi SOURCE verb, yang dalam hal ini akan meminta endpoint untuk

menciptakan resource representation dan menyertakannya di dalam response yang akan diterima requestor.



Gambar II.7: Request Evaluation oleh Endpoint

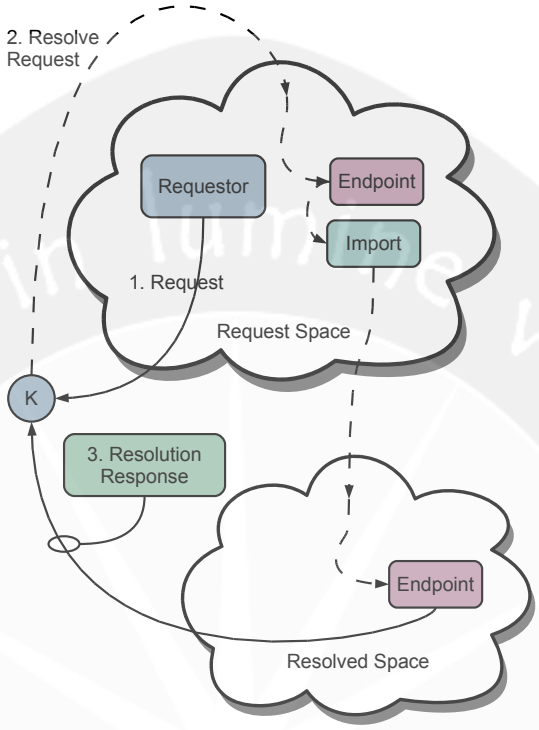
II.4.3.2 Physical Request Processing

Di *physical level*, endpoint melakukan inisialisasi proses dengan membangun sebuah request dan mengirimkannya ke kernel. Request ini memiliki sebuah identifier, verb, dan *resource scope*. *Resource scope* adalah sebuah *linked-list* dari beberapa space yang dapat digunakan request.

Kernel kemudian akan melakukan mediasi dan mengirim resolution request ke space. Space akan melakukan cek terhadap request dan masuk dan mencocokan identifiernya dengan endpoint yang tersedia, jika tidak ditemukan endpoint yang cocok maka, kernel akan melanjutkan dengan mengirim resolution request ke space berikutnya (yang terdapat) di dalam *resource scope*.

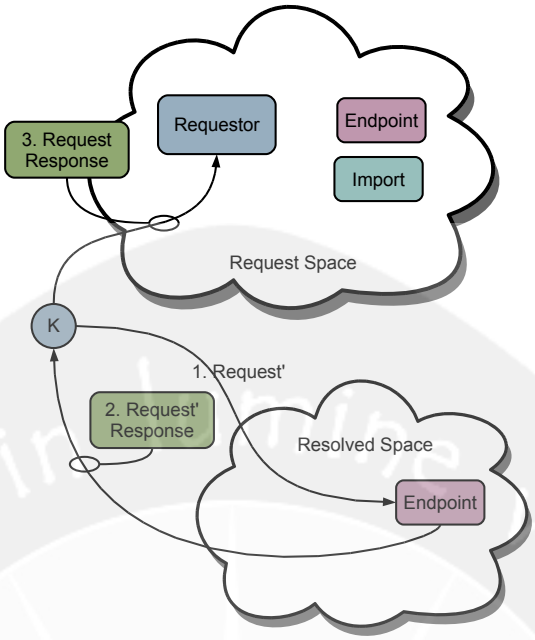
Jika seluruh endpoint yang ada di dalam *resource scope* tidak dapat memenuhi request resolution maka kernel akan

melemparkan resolution exception. Endpoint yang menerima request resolution akan mengembalikan response resolution ke kernel. Interaksi ini dapat dilihat di diagram berikut.



Gambar II.8: Physical Request Resolution

Pada saat evalution phase, endpoint yang telah di resolve saat fase resolusi dipanggil oleh kernel menggunakan request awal. Jika endpoint dapat menyelesaikan request, maka sebuah response yang berisi representasi resource akan dikembalikan ke kernel. Kernel selanjutnya akan mengembalikan response tersebut ke requestor.



Gambar II.9: Physical Request Evaluation

II.5 Tinjauan Pustaka

Service Oriented Architecture merupakan salah satu mekanisme integrasi aplikasi yang dapat digunakan (Hery, 2009). Service oriented architecture digunakan untuk menyediakan layanan(service) pada suatu sistem yang dapat digunakan sistem lain sesuai dengan kebutuhan. Arsitektur ini, jika dipenuhi maka akan membungkus fungsionalitas sebagai sebuah layanan(service). Service orientation bertujuan untuk memberikan layanan yang dapat diakses sistem lain, sehingga mendukung integrasi antar aplikasi.

Dalam pendekatannya, Service Oriented Architecture menggunakan web service sebagai jembatan untuk memberikan layanan yang bisa diakses oleh sistem lain. Web Service mendukung integrasi antar sistem dengan kemampuannya dalam hal memberikan layanan yang dapat diakses oleh sistem lain.

Resource Oriented Architecture(ROA) memiliki pendekatan yang sedikit berbeda dibandingkan SOA. ROA lebih mene-

kankan sebuah service sebagai resource yang dapat di ambil. Resource ini dapat berupa data atau layanan, dan dapat saling bertautan satu dengan yang lainnya.

Beberapa riset mengenai Service Oriented Architecture sudah pernah dilakukan. Salah satunya adalah Analisis dan Implementasi Integrasi Sistem Informasi Universitas Atma Jaya Yogyakarta Dengan Service-Oriented Architecture(SOA) (Hery). Dalam risetnya, penulis membangun sebuah sistem yang mengintegrasikan sistem informasi di Universitas Atma Jaya Yogyakarta bagian kepegawaian, aktivitas akademik dosen, dan penjaminan mutu. Sistem-sistem yang dapat diintegrasikan tersebut dapat bertukar informasi sehingga data yang diperoleh lebih valid dan dapat lebih mendukung pengambilan keputusan.

Riset lain mengenai web service yang pernah dilakukan adalah Pembangunan Sistem Informasi Geografis Rumah Sakit Wilayah DIY Berbasis Web(Christiana, Rika). Dalam riset ini, penulis membangun sebuah sistem informasi geografis dan sistem informasi untuk rumah sakit. Kedua sistem ini terintegrasi dengan menggunakan web service. Hasil dari riset ini adalah sebuah sistem informasi geografis yang mampu memberikan informasi klinik dan jadwal praktek dokter di rumah sakit tertentu.