

## **BAB II**

### **DASAR TEORI**

Pada Bab 2 ini akan dibahas beberapa teori yang digunakan dalam pengerjaan tugas akhir ini. Sesuai dengan judul tugas akhir ini, maka sistem yang akan dibentuk merupakan sistem terdistribusi dengan komponen pembentuknya antara lain *web service* dan *Business Process Execution Language* (BPEL). Dalam sistem tersebut harus diperhatikan pula konsep ACID yang dijalankan jika terjadi suatu kegagalan dalam proses bisnis perusahaan. Prinsip ACID tersebut merupakan suatu mekanisme yang ditangani dalam BPEL yang dihasilkan.

#### **2.1 Web Service**

*World Wide Web Consortium* (W3C) mendefinisikan *web service* sebagai suatu sistem perangkat lunak yang didesain untuk mendukung operasi interaksi antara mesin dengan mesin melalui suatu jaringan [WEE05]. *Web service* hanyalah suatu *Application Programming Interface* (API) yang dapat diakses melalui suatu jaringan seperti internet, dan dieksekusi pada sistem *remote* yang merupakan pemilik dari *services* yang diminta. *Web Services* ini memiliki dua spesifikasi dasar yang harus dipenuhi [ERL04], yaitu :

1. Berkomunikasi melalui protokol internet (secara umum adalah HTTP).
2. Mengirim dan menerima data dalam format dokumen XML.

Seperti yang telah dijelaskan pada bab II.1, *web service* adalah salah satu cara untuk merealisasikan SOA selain dengan cara pendekatan tradisional seperti CORBA dan DCOM. Keuntungan yang dimiliki oleh *web service* jika dibandingkan dengan CORBA dan DCOM adalah aspek *loose coupling* yang dimiliki oleh arsitekturnya. Pengertian *loose coupling* adalah fleksibilitas terhadap *platform*, format, dan protokol yang digunakan dalam komunikasi antara *service* dengan peminta *service* [WEE05]. Keunggulan yang lain yaitu penggunaan teknologi dan spesifikasi yang dikembangkan secara terbuka dan *vendor-neutral*. Jadi teknologi dan spesifikasi tersebut dapat digunakan secara fleksibel terhadap aplikasi karena hanya menggunakan standard yang merupakan fondasi dari komunikasi internet. Secara umum, teknologi *web services* ini dibagi menjadi dua yaitu *first-generation web services* dan *second-generation web services*.

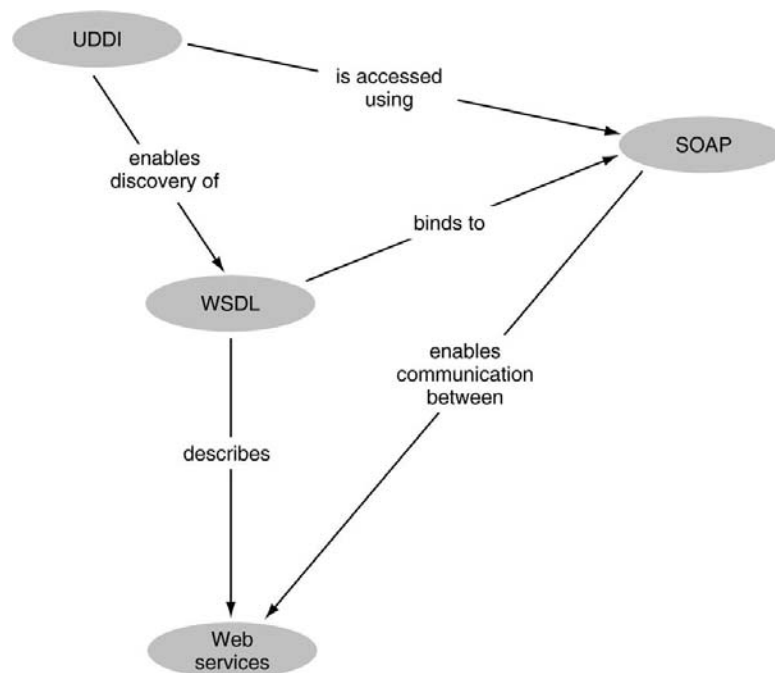
##### **2.1.1 First Generation Web Service [ERL04]**

*First-generation web service* menjadi suatu standard tersendiri yang digunakan dalam teknologi ini. Arsitektur dasar *web service* terdiri dari spesifikasi yang mendukung interaksi dari peminta

*web service* dengan penyedia *web service*, cara mendefinisikan *web service*, dan cara menemukan *web service* yang disediakan. Spesifikasi tersebut antara lain adalah WSDL, SOAP, dan UDDI [WEE05]. Keterkaitan antar tiga komponen dasar di atas dapat dilihat pada Gambar II-1. Penjelasan lebih lanjut tentang WSDL, SOAP, dan UDDI dapat dilihat pada subbab-subbab di bawah ini.

### 2.1.1.1 Web Service Definition Language (WSDL)

WSDL adalah spesifikasi dari W3C yang menyediakan bahasa untuk mendeskripsikan definisi dari *web service* [ERL04]. Dalam definisi tersebut termasuk juga deskripsi layanan dan fungsi-fungsi yang disediakan oleh *web service*.



**Gambar II-1. Keterkaitan Antara Spesifikasi *First Generation Web Service* [ERL04]**

Struktur umum dari WSDL adalah sebagai berikut [ERL04] :

```

<definitions>
  <interface>
    <operation>
      ---
    </operation>
  </interface>
  <message>
    <part>
      ---
    </part>
  </message>

```

```

    <service>
      <endpoint>
        ---
      </endpoint>
    </service>
  <binding>
    ---
  </binding>
</definitions>

```

Komponen *interface* dan *message* mendefinisikan antarmuka dari *service* sedangkan komponen *service* dan *binding* mendefinisikan detail implementasi dari *service*. Penjelasan lengkap komponen WSDL adalah sebagai berikut :

1. Interface mendefinisikan antarmuka dari web service dan terdiri dari beberapa operasi yang dimiliki.
2. Operations merepresentasikan fungsi dan operasi yang dimiliki oleh web service dan dapat mengacu pada banyak komponen message.
3. Messages mendefinisikan koleksi dari parameter masukan dan keluaran dan dapat mengandung banyak komponen parts.
4. Parts merepresentasikan data masukan dan keluaran yang digunakan komponen operation.
5. Services merupakan kumpulan dari endpoints yang didefinisikan dalam masing-masing komponen endpoint.
6. Endpoint mengandung data dari endpoint, termasuk alamat fisik dan informasi protokol.

*Binding* menghubungkan enam komponen di atas dengan komponen *operation*.

### 2.1.1.2 Simple Object Access Protocol (SOAP)

Dalam komunikasi antara *web service* diperlukan suatu standard format pesan antara peminta *web services* dengan penyedia *web services*. SOAP adalah format pesan yang digunakan untuk komunikasi tersebut. SOAP mendefinisikan mekanisme pembungkusan yang mengatur pertukaran pesan antara *web services* [WEE05]. Pesan SOAP adalah dokumen XML yang mengandung tiga elemen yaitu *envelope*, *header*, *body*. Struktur umum SOAP adalah sebagai berikut [ERL04]:

```

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    ---
  </env:Header>
  <env:Body>
    ---
  </env:Body>
</env:Envelope>

```

*Envelope* adalah elemen utama dari pesan SOAP. Elemen ini mengandung elemen *header* yang opsional dan elemen *body* yang harus ada. Elemen *header* adalah mekanisme umum untuk mendefinisikan fitur tambahan pada SOAP. Elemen *body* berisi deskripsi pesan aktual ditujukan untuk penerima akhir dan akan diproses. Komunikasi pesan SOAP antar *web service* ini dilakukan melalui protokol HTTP.

### 2.1.1.3 Universal Description, Discovery, and Integration (UDDI)

UDDI adalah spesifikasi yang telah diakui secara luas untuk pendaftaran suatu *web service*. UDDI mendefinisikan metadata dan protokol yang digunakan untuk mengetahui dan mengubah informasi dari suatu *web service* [WEE05]. Langkah pertama dalam menemukan *web service* adalah dengan meminta alamat tempat penyimpanan dari *web service* yang akan dipakai yang biasa disebut dengan *directory*.

Setelah menemukan *directory*, peminta *web service* dapat mengirimkan permintaan lagi untuk mendapatkan informasi detail tentang *web service* (misalnya penyedia *web service* dan dimana diletakkan). Selanjutnya perangkat lunak menggunakan informasi yang didapat untuk secara dinamik mengakses *web service* yang diinginkan.

Tempat penyimpanan UDDI dapat dibagi dalam tiga cara [WEE05], yaitu :

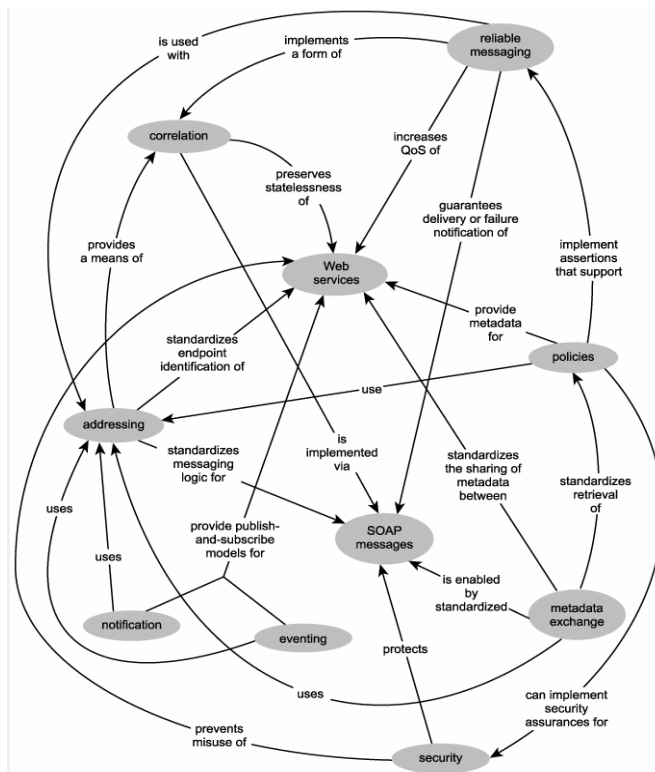
1. *Public UDDI*, dianggap sebagai resource bagi internet-based web service. Salah satu contohnya adalah UDDI Business Registry (UBR) yang dimiliki oleh grup yang terdiri dari IBM, Microsoft, SAP.
2. *Intra Enterprise UDDI*, merupakan tempat penyimpanan yang private yang menyediakan service bagi kalangan internal suatu perusahaan.
3. *Inter Enterprise UDDI*, merupakan repository yang dapat diakses oleh perusahaan-perusahaan yang merupakan partner dalam suatu bisnis.

Seperti yang telah dijelaskan pada bab II.1, *service discovery* memainkan peranan yang penting dalam SOA. Banyak cara yang dapat dipakai untuk mekanisme ini, tapi dalam teknologi *web service*, UDDI menyediakan standard yang baik dan fleksibel untuk *web service discovery*.

### 2.1.2 Second Generation Web Service [ERL04]

Motivasi untuk menambah kapabilitas dari *first-generation web service* adalah untuk mendukung terbentuknya SOA dan juga meningkatkan kemampuan untuk menangani fungsi bisnis yang dimiliki oleh suatu perusahaan. Oleh karena itu muncul *second-generation web service* yang menambahkan ekstensi-ekstensi pada *web service* yang sebelumnya [ERL04].

Ekstensi-ekstensi yang merupakan spesifikasi pokok dari generasi *web service* ini antara lain adalah WS-Coordination, WS-Transaction, Business Process Execution Language for Web Services (BPEL4WS), WS-ReliableMessaging, WS-Addressing, WS-Policy, WS-PolicyAssertion, WS-PolicyAttachments, WS-Attachments, SOAP with Attachments (SwA). Keterkaitan antar standard pada *first-generation* dengan *second-generation web service* dapat dilihat pada gambar II-2.



**Gambar II-2. Keterkaitan Antar Standard *First Generation* dengan *Second Generation Web Service* [ERL04]**

Untuk mengetahui pentingnya ekstensi-ekstensi di atas, perlu dilihat pada beberapa kebutuhan fungsi bisnis yang belum dapat dipenuhi oleh *first-generation web services*. Beberapa kebutuhan itu antara lain [ERL04] :

1. Manajemen transaksi dan konteks. *First-generation web service* tidak memiliki kemampuan untuk manajemen transaksi. Dengan tidak adanya kemampuan ini, web service berjalan secara *independent* dan tidak mendukung transaksi yang terdistribusi. *WS-Coordination* menyediakan suatu mekanisme untuk manajemen konteks yang dapat diterapkan untuk mendukung transaksi yang *atomic* dan *long-running*. Mekanisme ini menggunakan *protocol* yang dideskripsikan dalam spesifikasi *WS-Transaction*.

2. Proses bisnis. Untuk menyusun beberapa *web service* agar menjadi *workflow* yang terstruktur, suatu standard sangatlah diperlukan. *Business Process Execution Language for Web Services* (BPEL4WS) menyediakan standard tersebut yang dapat digunakan oleh aplikasi yang mendukung orkestrasi *web services*.
3. Keamanan. Faktor inilah yang menjadi perbedaan yang cukup mencolok antara *first-generation* dengan *second-generation web service*. Secara umum suatu organisasi enggan untuk membuka proses bisnis mereka melalui internet, sehingga faktor keamanan sangatlah penting. WS-Security merupakan spesifikasi yang ditujukan dalam masalah keamanan.
4. *Reliable messaging*. Untuk menciptakan komunikasi yang baik, dibutuhkan suatu mekanisme yang menjamin status terkirimnya suatu pesan ke tujuan dan termasuk juga cara untuk menangani pengiriman yang gagal. WS-ReliableMessaging menyediakan sistem ini yang dapat digunakan pada proses bisnis yang *delivery-related*.
5. *Policies*. Dalam perusahaan yang *service-oriented*, kebutuhan untuk menggambarkan aturan bisnis, aturan keamanan, dan property lainnya dalam suatu kebijakan sangatlah dibutuhkan. WS-Policy merupakan cara untuk menangani kebutuhan ini.
6. *Attachments*. Penggunaan SOAP sebagai standard format pesan, dapat membatasi tipe data yang dapat dikirimkan. WS-Attachment mendukung SOAP dengan menyediakan mekanisme untuk pengiriman data dalam format yang tidak didukung SOAP dan juga file sebagai suatu *attachment* pada pesan SOAP.

## **2.2 Business Process Execution Language (BPEL)**

Teknologi *web service* telah memberikan suatu keuntungan tersendiri dalam pengembangan aplikasi. Dalam perusahaan pastinya terdapat suatu proses bisnis dan digambarkan dengan pemodelan interaksi antar *web service* yang dimiliki aplikasi. Mekanisme inilah yang ditangani oleh *Web Service Business Process Execution Language* (WS-BPEL) atau yang lebih dikenal dengan nama BPEL.

BPEL adalah bahasa yang berbasis pada *workflow* yang menggabungkan *service* yang berperan dalam suatu interaksi yang dapat berupa orkestrasi dan koreografi *service* [WEE05]. BPEL memanfaatkan deskripsi WSDL untuk mendefinisikan fungsi yang disediakan oleh *service* yang ada dan model interaksi yang dibentuk. XPath merupakan standard yang dapat digunakan untuk mengakses dan memanipulasi data yang digunakan dalam koreografi BPEL. Untuk manajemen *endpoint* dari *service* yang disusun dapat digunakan standard WS-Addressing.

Dalam lingkungan aplikasi yang menggunakan teknologi *web service*, terdapat beberapa kebutuhan yang harus dipenuhi untuk mendefinisikan model komposisi dari *web service* [WEE05], antara lain:

1. *Flexibel Integration*, model yang terbentuk harus dapat beradaptasi terhadap perubahan interaksi antar *web service* yang disebabkan oleh perubahan proses bisnis.
2. *Recursive Composition*, model interaksi harus dapat dengan mudah dikomposisikan dengan service yang telah ada sehingga meningkatkan nilai *scalability* dan *reuse*.
3. *Separation and Composeability*, Logik dari proses bisnis dan komposisi service harus dapat dipisahkan dengan mekanisme teknis yang mengatur fungsionalitas service. Informasi tersebut dapat ditambahkan sebagai suatu layer ataupun attachment pada bagian yang terpisah dengan definisi service.
4. *Statefull Conversation and Lifecycle Management*, model harus dapat merepresentasikan *lifecycle* model dari proses bisnis sehingga dapat menangani mekanisme *long-running conversations* yang terjadi pada interaksi antar service.
5. *Recoverability*, model harus dapat mendukung mekanisme *fault-handling* dan *compensation* yang akan menangani suatu kondisi *error* yang mungkin terjadi selama eksekusi proses.

BPEL dapat menangani semua kebutuhan di atas dengan beberapa fungsionalitas yang disediakan. BPEL merupakan kombinasi dari dua teknik terdahulu yang menangani mekanisme komposisi dari *web service*, yaitu *IBM Web Service Flow Language (WSFL)* dan *Microsoft XLANG (XLANG)* [WEE05]. BPEL menggabungkan dua teknik yang ada pada masing-masing bahasa di atas. Proses dalam BPEL dapat dibuat dengan menggunakan pemodelan yang *graph-oriented* yang disediakan oleh WSFL dan mekanisme pertukaran pesan yang ada pada XLANG. *Service* yang berpartisipasi dalam proses disebut dengan nama *partner service*. Ketika suatu proses BPEL dibangkitkan, *instance*-nya akan tetap ada selama eksekusi belum selesai. Ada dua cara pembentukan proses yaitu *partner service* membangkitkan *instance* yang baru dari proses atau *partner service* berinteraksi dengan *instance* yang telah ada [ERL04].

Deskripsi dari proses BPEL merupakan suatu *workflow* yang terdiri dari urutan aktivitas sesuai dengan kondisi dan logik dari proses bisnis yang ada. Secara umum ada dua aktivitas dalam proses BPEL [ERL04], yaitu :

1. *Basic Activities*

Aktivitas ini menyediakan fungsi primitif dari BPEL, antara lain *receive*, *invoke*, *reply*, *throw*, dan *wait* [ERL04]. Tiga fungsi *receive*, *invoke*, dan *reply* memungkinkan interaksi antar proses service dengan *partner service*. Jika ada suatu kondisi yang dikenali oleh

proses, suatu *exception handling* akan dieksekusi. Fungsi *throw* akan melempar eksekusi pada proses yang sesuai dengan kondisi *fault*. Fungsi *wait* dapat digunakan untuk menyediakan mekanisme eksekusi yang terjadwal. Fungsi ini akan menunda eksekusi dari proses dalam waktu yang dispesifikasikan.

## 2. *Structured Activities*

Aktivitas ini menyediakan mekanisme untuk pembentukan logik dari *workflow* proses bisnis. Beberapa fungsi dalam aktivitas ini antara lain adalah *sequence*, *flow*, *switch*, dan *while* [ERL04]. Fungsi *sequence* adalah untuk mendefinisikan bagaimana urutan suatu aktivitas dieksekusi, sedangkan *flow* menangani mekanisme eksekusi aktivitas yang dapat dilakukan secara konkuren. *Switch* dan *while* memiliki fungsi sama seperti yang ada pada pemrograman biasa yaitu untuk menangani *conditional*.

Cara untuk mendefinisikan suatu proses dalam BPEL adalah sebagai berikut [ERL04].

```
<process name="...">
  <partnerLinks>
    <partnerLink name="...">
      ---
    </partnerLink>
  </partnerLinks>
  <variables>
    <variable ... />
    ---
  </variables>
  <faultHandlers>
    <catch ...>
      ---
    </catch>
    <catchAll>
      ---
    </catchAll>
  </faultHandlers>
  <sequence>
    <receive ...>
      ---
    </receive>
    <invoke ...>
      ---
    </invoke>
    <reply ...>
      ---
    </reply>
  </sequence>
</process>
```

Penjelasan masing-masing elemen di atas adalah sebagai berikut :

1. *PartnerLinks* terdiri dari beberapa elemen *partnerLink* yang digunakan untuk merepresentasikan partner service. Elemen ini diisi dengan nama dari services yang digunakan dalam proses dan juga alokasi dari peran yang dipegang.
2. *Variables* digunakan untuk manajemen informasi dan status proses. Hal ini digunakan untuk mekanisme penanganan *long-running process* yang membutuhkan penyimpanan suatu status proses pada setiap kondisinya. Dalam elemen ini terdapat konstruktor

- individu variable yang didefinisikan berdasar *scope*-nya. Nilai dari variable biasanya berisi pesan yang mengandung informasi status suatu proses.
3. *faultHandlers* menangani mekanisme penanganan *fault condition*. Pada saat proses bisnis yang *long-running* gagal, maka proses akan ditangani oleh *compensation* yang terenkapsulasi dalam elemen ini. Untuk penjelasan yang lebih detail dapat dilihat pada bab II.5.
  4. *Sequence* merupakan elemen untuk mengatur urutan eksekusi aktivitas yang didefinisikan dalam *basic activities* seperti *receive*, *invoke*, dan *reply*.

### **2.3 Aplikasi e-Commerce [WIK07]**

*E-Commerce* yang merupakan singkatan dari *Electronic Commerce* dapat dianalogikan dengan tempat pemasaran melalui fasilitas internet. Termasuk di dalamnya adalah distribusi, membeli, menjual, memasarkan dan melayani permintaan produk atau jasa melalui sistem elektronik semisal internet dan juga jaringan komputer yang lain. *E-Commerce* biasanya menggunakan teknologi komunikasi elektronik dari *World Wide Web* pada beberapa poin dalam siklus transaksinya, meskipun pastinya aplikasi ini juga bergantung pada teknologi lain, semisal *database* dan *e-mail*. Teknologi non-komputer juga memiliki peran penting seperti misalnya transportasi untuk barang yang terjual melalui proses *e-commerce*.

Dalam banyak kasus, perusahaan *e-commerce* akan bertahan tidak hanya dengan produknya, tetapi dengan memiliki tim manajemen yang kompeten, layanan yang memuaskan, struktur bisnis yang terorganisasi dengan baik, infrastruktur jaringan yang baik dan cukup aman, *website* yang didesain dengan baik. Perusahaan yang ingin sukses harus mempertimbangkan dua hal utama yaitu aspek teknik dan organisasi dan juga prinsip *customer-oriented*. Penjelasan dua hal tersebut adalah sebagai berikut :

1. Aspek teknik dan organisasi
 

Beberapa hal dalam aspek ini yang dapat mendukung suksesnya perusahaan *e-commerce* antara lain :

  - a. Tim manajemen handal yang memiliki strategi dalam teknologi informasi. Strategi tersebut sebaiknya adalah bagian dari re-design proses bisnis perusahaan.
  - b. Menyediakan suatu cara yang mudah dan aman bagi konsumen untuk melakukan transaksi. Sebagai contoh adalah cara menangani mekanisme pembayaran dengan kartu kredit melalui fasilitas internet.
  - c. Menyediakan suatu jaminan kehandalan dan keamanan sistem.

- d. Mengatur suatu relationship yang baik antara komponen dalam proses bisnis yaitu pegawai, konsumen, penyuplai, dan partner lain yang berkaitan.
  - e. Menyiapkan agar organisasi dapat memberikan respon yang cepat terhadap segala macam perubahan baik dari segi ekonomi, social, dan lingkungan sekitar.
  - f. Menyediakan website yang atraktif.
  - g. Mengefisienkan proses bisnis melalui *re-engineering* dan pemanfaatan teknologi informasi.
  - h. Mengerti dengan detail tentang produk dan layanan yang ditawarkan.
2. Prinsip *customer-oriented*

Organisasi e-commerce yang sukses harus menyediakan pelayanan yang menyenangkan dan memuaskan terhadap konsumennya. Beberapa faktor yang dapat mempengaruhi kepuasan konsumen antara lain :

- a. Memberikan suatu nilai tambah kepada konsumen. Hal ini dicapai dengan menawarkan produk yang menarik konsumen potensial pada harga yang kompetitif.
- b. Menyediakan layanan dan performansi yang baik.
- c. Menyediakan dorongan kepada konsumen untuk membeli dan terus kembali kepada produk perusahaan. Hal ini dilakukan dengan promosi, pemberian diskon, ataupun adanya kupon-kupon khusus.
- d. Memberikan suatu perhatian personal kepada konsumen. Website, saran untuk pembelian, dan penawaran special kepada personal orang merupakan beberapa cara yang dapat dilakukan.
- e. Membebaskan konsumen untuk melayani diri sendiri. Penyediaan dari situs self-serve yang mudah untuk digunakan tanpa menggunakan semacam bantuan dapat dimanfaatkan untuk mencapai faktor ini.

Banyak perusahaan *e-commerce* yang sukses adalah dalam bentuk usaha yang *virtual*. Perusahaan tersebut biasanya berurusan dengan music, film, edukasi, komunikasi, perangkat lunak, fotografi, dan transaksi finansial. Sebagai contoh perusahaan yang sukses dalam bidang di atas adalah *Google*, *eBay*, dan *Paypal*.

Dalam pengerjaan Tugas Akhir ini terdapat batasan pengertian *e-commerce* yang digunakan. *E-commerce* yang digunakan hanya merupakan suatu subsistem dari *e-commerce* secara umum. Pengerjaan Tugas Akhir ini hanya menerapkan subsistem yang menangani proses transaksi pembelian. Subsistem lain seperti manajemen pegawai ataupun pengaturan stock buku tidak dimasukkan dalam pengertian *e-commerce* ini.

## 2.4 Konsep ACID dalam BPEL

Konsep dalam bab ini sangat erat kaitannya dengan dua ekstensi dari *web service* yang telah disebutkan pada bab II.2.2 yaitu WS-Coordination dan WS-Transaction. Oleh karena itu, awal pembahasan bab ini adalah mengenai dua ekstensi di atas dan dilanjutkan dengan kombinasinya terhadap BPEL sehingga dapat dijaga konsep ACID.

### 2.4.1 WS-Coordination [ERL05]

Semakin rumit aktivitasnya maka semakin banyak informasi konteks yang harus dijaga. Kerumitan aktivitas dapat dipengaruhi oleh beberapa faktor, antara lain :

1. Jumlah *services* yang berperan dalam aktivitas.
2. Lamanya suatu aktivitas.
3. Frekuensi perubahan suatu aktivitas.
4. Banyaknya jumlah *instances* dari aktivitas yang berjalan secara konkuren.

Suatu mekanisme diperlukan untuk mengatur informasi konteks dalam aktivitas yang rumit dan WS-Coordination muncul sebagai suatu solusi masalah ini. WS-Coordination membentuk suatu *service* umum yang akan menjadi suatu *coordinator*. *Service Coordinator* ini akan mengendalikan komposisi dari *service* lain yang memiliki peran tertentu dalam manajemen *context data*. Beberapa *service* tersebut antara lain [ERL05]:

1. *Activation service*, bertanggung jawab dalam pembuatan konteks yang baru dan menghubungkan konteks ini dengan aktivitas yang sesuai.
2. *Registration service*, memungkinkan *service* yang berpartisipasi dalam aktivitas untuk menggunakan informasi konteks yang diterima dari *activation service* dengan mendaftarkan *service* tersebut.
3. *Protocol-specific service*, merepresentasikan protokol yang didukung oleh *coordinator*.

Setiap *coordinator* memiliki suatu tipe koordinasi yang menspesifikasikan suatu kebiasaan dan logik dari aktivitas yang diatur oleh informasi konteks. Tipe koordinasi ini merupakan spesifikasi yang terpisah dan dapat ditambahkan pada WS-Coordination. Dua tipe koordinasi yang secara umum dipakai adalah WS-AtomicTransaction dan WS-BusinessActivity. Pembahasan lebih lanjut tentang dua tipe ini ada pada bab II.5.2. Tipe koordinasi ini merupakan sekumpulan protokol yang berisi aturan yang harus diikuti oleh partisipan yang terdaftar dalam suatu aktivitas. Konteks yang dibuat oleh *activation service* mengandung koleksi informasi yang merepresentasikan aktivitas dan data tambahan. Contoh data yang ada dalam konteks antara lain identitas unik dari aktivitas, nilai *expiration*, dan informasi dari *coordination type*. *Service* yang

ingin ambil bagian dalam aktivitas yang diatur oleh WS-Coordination harus meminta *coordination context* melalui *activation service* dan untuk selanjutnya melakukan *registration*.

Suatu *coordination service* dimulai ketika *application service* memanggil *activation service*. Dengan melalui pesan *CreateCoordinationContext*, maka *activation service* menghasilkan data konteks yang baru. Setelah itu pesan *ReturnContext* dikirimkan dan *application service* dapat mengundang *service* lain untuk berpartisipasi dalam aktivitas. Semua *service* yang berpartisipasi harus mendaftarkan alamatnya pada *registration service*. Setelah proses *registration* berhasil, maka lokasi dari *coordinator service* diberikan pada semua partisipan yang berinteraksi. Pada saat yang sama, *coordinator service* juga dikirim alamat dari partisipan yang baru.

*Application service* dapat mengindikasikan bahwa *coordination* telah selesai dengan mengirimkan pesan *completion* kepada *coordination service*. Selanjutnya *coordinator service* mengirimkan pesan *completion* kepada semua partisipan dan setiap partisipan membalasnya dengan pesan *completion acknowledgement*.

## 2.4.2 WS-Transaction

Seperti yang telah dijelaskan pada bab II.5.1 bahwa ada spesifikasi khusus yang menangani beberapa tipe koordinasi. WS-Transaction merupakan spesifikasi yang menangani permasalahan di atas. Secara umum ada dua tipe transaksi yaitu *atomic transaction* dan *long-running transaction* [ERL04]. Untuk masing-masing jenis transaksi di atas, terdapat spesifikasi khusus yang menanganinya yaitu WS-AtomicTransaction dan WS-BusinessActivity. Kedua spesifikasi tersebut merupakan perluasan dari spesifikasi WS-Transaction yang terlebih dahulu didefinisikan.

### 2.4.2.1 WS-Atomic Transaction [ERL05]

Transaksi merupakan suatu proses untuk mencapai solusi dari otomatisasi sistem komputer. Ketika terlibat dengan beberapa data, kebutuhan untuk membungkus perubahan ke dalam suatu aksi tunggal merupakan dasar dalam proses bisnis. Transaksi atomik menerapkan suatu fitur *commit* dan *rollback* untuk mendukung transaksi yang *cross-service*.

Protokol yang disediakan oleh WS-AtomicTransaction secara umum dapat disamakan dengan fitur transaksi ACID secara umum dikenal. Untuk lebih jelasnya, penjelasan karakteristik ACID adalah sebagai berikut [ERL05] :

1. *Atomic*, karakteristik ini memastikan apakah semua perubahan dari transaksi berjalan dengan sukses atau tidak ada satupun yang sukses. Karakteristik ini menjalankan

- mekanisme *rollback* untuk mengembalikan semua perubahan yang merupakan akibat dari transaksi yang gagal menjadi kondisi aslinya.
2. *Consistent*, karakteristik ini memastikan tidak ada perubahan data sebagai akibat dari transaksi yang dapat melanggar validitas dari model data yang digunakan.
  3. *Isolated*, jika ada beberapa transaksi berjalan secara konkuren, maka tidak boleh ada gangguan satu sama lain. Setiap transaksi harus menjamin lingkungan eksekusi yang terisolasi.
  4. *Durable*, selama akhir dari transaksi yang sukses, perubahan yang dibuat sebagai hasil transaksi dapat bertahan dari kegagalan yang selanjutnya.

Untuk berpartisipasi dalam transaksi ini, *service* pertama kali menerima *coordination context* dari *activation service*. Selanjutnya *service* mendaftarkan pada protokol transaksi yang disediakan, antara lain [ERL05] :

1. *Completion protocol*, digunakan untuk menginisiasi status *commit* atau *abort* dari transaksi.
2. *Durable 2PC protocol*, untuk *service* yang merepresentasikan tempat penyimpanan data permanen yang harus didaftarkan.
3. *Volatile 2PC protocol*, digunakan untuk *service* yang menggunakan data temporer.

Saat protokol WS-AtomicTransaction digunakan, *coordinator* akan beralih peran menjadi *atomic transaction coordinator*. Koordinator ini berperan dalam mengatur partisipan dari transaksi dan memutuskan keluaran final dari transaksi. Dalam menentukan keluaran dari transaksi, koordinator menggunakan *feedback* yang diterima dari semua partisipan transaksi.

Pengumpulan dari *feedback* ini dibagi menjadi dua fase. Selama fase *prepare*, semua partisipan dinotifikasi oleh koordinator dan setiap partisipan diminta untuk bersiap dan mengeluarkan suatu *vote*. Setiap *vote* dapat mengandung antara permintaan *commit* ataupun *abort*.

Setelah *vote* dikumpulkan, koordinator memasuki fase *commit*. Dalam fase ini, akan dilakukan *review* terhadap semua *vote* dan memutuskan apakah transaksi akan *commit* atau *rollback*. Jika semua *votes* yang diterima adalah *commit*, maka koordinator memutuskan bahwa transaksi sukses. Akan tetapi jika satu *vote* saja adalah *abort* atau ada partisipan yang tidak memberikan respon, maka transaksi dinyatakan gagal dan dilakukan *rollback*.

#### **2.4.2.2 WS-Business Activity [ERL05]**

Spesifikasi ini mengatur aktivitas yang sifatnya *long-running*. Selama durasi menunggu dalam jam, harian, atau bahkan mingguan, aktivitas dapat menjalankan beberapa proses yang melibatkan banyak partisipan. Faktor yang membedakan antara *business activity* dengan aktivitas kompleks

yang biasa adalah bahwa partisipannya diharapkan untuk mengikuti aturan yang didefinisikan oleh protokol. *Business activity* juga dapat dibedakan dengan *atomic transaction* dalam hal penanganannya terhadap *exception* dan *constraints* yang didefinisikan dalam aturan protokol.

Secara umum, protokol dalam *business activity* tidak menawarkan kemampuan *rollback*. Akan tetapi untuk tetap menjaga semantik ACID, maka tersedia proses *compensation* yang akan menjalankan suatu proses lain jika terjadi kondisi *exception*.

WS-BusinessActivity menyediakan dua macam protokol yang mirip, yang masing-masing mengatur bagaimana partisipan bertindak dalam aktivitas bisnis. Protokol tersebut antara lain [ERL05] :

1. *BusinessAgreementWithParticipantCompletion protocol*, yang memungkinkan partisipan untuk menentukan kapan partisipan tersebut menyelesaikan bagiannya dalam aktivitas bisnis.
2. *BusinessAgreementWithCoordinationCompletion protocol*, yang membutuhkan partisipan untuk bergantung pada coordinator dalam menentukan bahwa tidak ada tanggung jawab proses lagi terhadap partisipan tersebut.

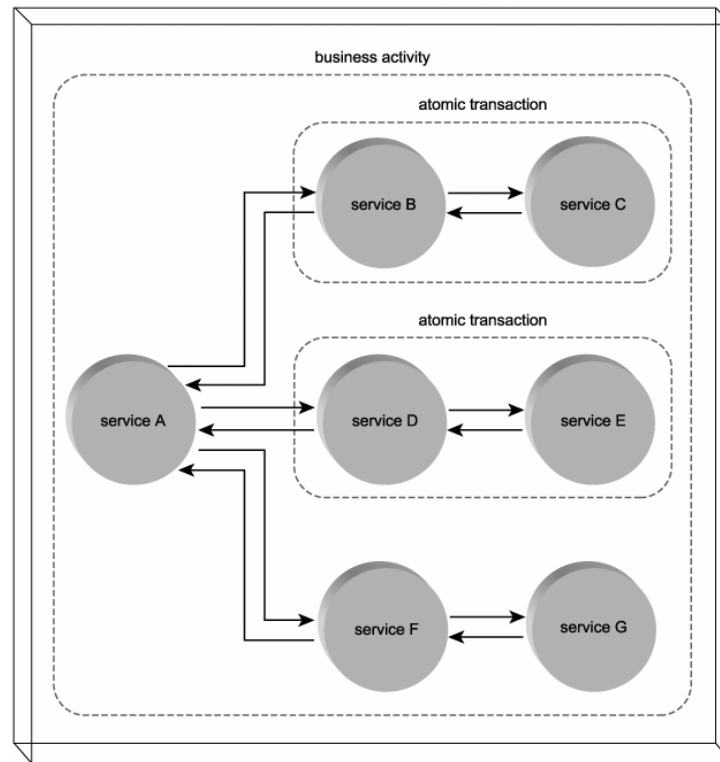
Selama *lifecycle* dari aktivitas bisnis, *coordinator* dan partisipan akan melalui transisi dalam beberapa *state*. Waktu dilakukannya transaksi diketahui melalui suatu pesan notifikasi yang dikirimkan antara *service* yang berperan. Sebagai contoh, partisipan dapat mengindikasikan telah menyelesaikan prosesnya dengan mengirimkan *completed notification*. Hal ini mengubah *state* aktivitas dari *active* menjadi *completed*. *Coordinator* mungkin meresponnya dengan pesan *close* untuk menginformasikan kepada partisipan bahwa aktivitas bisnis telah selesai dengan sukses. Partisipan dapat juga masuk dalam *compensation state* jika diperlukan penanganan terhadap *exception*. *Compensation* berbeda dengan mekanisme pada *atomic transaction* yaitu bukan dalam bentuk *rollback* dari perubahan yang dilakukan *service* yang berkaitan, akan tetapi lebih kepada eksekusi suatu proses lain jika proses normal gagal. Selain itu, *cancelled state* dapat dilakukan. Hal ini akan menghasilkan suatu *termination* dalam semua proses yang sedang berlangsung.

Hal yang membedakan antara *business activities* dari *atomic transaction* adalah fakta bahwa *service* yang berpartisipasi tidak dibutuhkan untuk menjadi partisipan selama durasi dari aktivitas. *Service* tersebut dapat saja meninggalkan aktivitas bisnis setelah tugas individunya telah selesai. Pada saat meninggalkan aktivitas bisnis, partisipan memasuki *exit state* dengan mengirimkan pesan *exit* kepada *coordinator*.

Penggunaan *business activity* tidak dapat dipisahkan dari penggunaan *atomic transaction*. Pada Gambar II-3 dapat dilihat bahwa *business activity* merupakan penggabungan eksekusi beberapa *atomic transaction* selama masa aktifnya.

### 2.4.3 Penerapan WS-Coordination dan WS-Transaction Dalam BPEL [WEE05]

Karena BPEL terdiri dari susunan *web service*, maka harus ditangani kesalahan yang dikarenakan pemanggilan *web service*. Selain itu, kesalahan juga dapat disebabkan oleh logik dalam bisnis yang dapat terjadi selama eksekusi proses BPEL. Proses bisnis tidak dapat selalu dieksekusi sebagai operasi tunggal yang *atomic* karena secara umum proses tersebut merupakan *long-running process*. Oleh karena itu, hasil dari proses *intermediate* dibuat menjadi *persistent* dan dapat diakses oleh proses lain. Saat kegagalan terjadi pada proses bisnis yang *long-running*, maka aksi *undo* dilakukan terhadap proses yang telah selesai. Dalam BPEL, mekanisme menangani kegagalan merupakan tanggung jawab dari *fault handler* dan pekerjaan *undo* dilakukan oleh *compensation handler*.



Gambar II-3 Kaitan *Business Activities* Dengan *Atomic Transaction*

Sebagai tambahan, kegagalan juga sangat mungkin terjadi antara beberapa proses bisnis. Dalam lingkungan sistem yang terdistribusi, mekanisme penanganan kegagalan ini merupakan tanggung jawab dari WS-AtomicTransaction dan WS-BusinessActivity yang telah dijelaskan pada bab II.5.2.1 dan bab II.5.2.2. WS-BusinessActivity mendeskripsikan model untuk mengkoordinasikan aksi *compensation* dalam beberapa aplikasi bisnis yang berbeda.