LAPORAN PENELITIAN

PEMODELAN DAN IMPLEMENTASI ANTARMUKA *WEB SERVICES* SISTEM INFORMASI UNPAR

Oleh:

Muhammad Ghifary, S.T., M.T.

Gede Karya, S.T, M.T.

Jurusan Teknik Informatika

Fakultas Teknologi Informasi dan Sains

Universitas Katolik Parahyangan

Desember 2011

Pemodelan dan Implementasi Antarmuka *Web Services* Sistem Informasi UNPAR

Muhammad Ghifary
Fakultas Teknologi Informasi dan Sains
Teknik Informatika
Universitas Katolik Parahyangan
Jl. Ciumbuleuit 94 Bandung 40141
Email: mghifary@unpar.ac.id, mghifary@gmail.com

Abstrak

Saat ini di Universitas Parahyangan (UNPAR) telah berjalan berbagai sistem informasi (SI) perangkat lunak berbasis *web* yang membantu keberlangsungan kegiatan akademik dan administrasi. Sistem-sistem tersebut antara lain SI Akademik, SI Keuangan, SI Kepegawaian, dan SI Perpustakaan. Seperti pada umumnya perangkat lunak *web*, layanan yang diberikan bersifat *machine-to-human* yaitu perangkat lunak menyediakan sajian informasi kepada pengguna yang dapat diakses melalui *web browser*. Penelitian ini mencoba memodelkan sebuah jalur protokol komunikasi *machine-to-machine* antara sistem informasi yang ada di UNPAR dengan perangkat lunak lainnya, terutama *mobile application*, dengan memanfaatkan teknologi *web services* serta tetap menjaga independensi dan keamanan sistem.

Terdapat dua hal yang menjadi fokus analisis utama dari model *web services* untuk SI UNPAR yang akan dirancang. Hal pertama adalah mengenai pemilihan jenis teknologi *web services* yang sesuai. Ada dua jenis teknologi *web services* yang lazim digunakan yaitu SOAP/Big Web Services dan REST Web Services yang masing-masing memiliki kelebihan dan kekurangan, serta diperuntukkan untuk keperluan yang berbeda pula. Selain itu, diperlukan pula pemilihan yang sesuai mengenai format pesan pertukaran yang digunakan sebagai media komunikasi antar perangkat lunak luar dengan SI di UNPAR. Hal kedua adalah mengenai mekanisme pengamanan jaringan *web services* agar tetap menjamin tiga prinsip keamanan informasi yaitu *authentication*, *confidentiality*, dan *integrity*.

Berdasarkan hasil analisis yang telah dilakukan, dirancang sebuah model *web services* yang menggunakan kedua teknologi “SOAP” dan “REST” di jalur komunikasi yang berbeda. Sedangkan, untuk format pesan pertukaran digunakan format pesan JavaScript Object Notation (JSON). Yang menjadi elemen utama dari model tersebut adalah *Web Services Server* berbasis Java 2 Enterprise Edition (J2EE) yang dibangun terpisah dengan aplikasi SI di UNPAR. Aplikasi tersebut akan berfungsi sebagai pintu gerbang yang menghubungkan perangkat lunak luar dengan SI UNPAR. Untuk mekanisme autentikasi, digunakan mekanisme Single Sign-On yang didukung oleh sebuah *web server* bernama Central Authentication Service (CAS). Untuk menguji apakah model tersebut dapat diimplementasikan, dibangun tiga prototipe perangkat lunak, yaitu Web Services Server, CAS Server, dan aplikasi klien uji yang mensimulasikan cara kerja dari model. Kumpulan prototipe tersebut menjalankan dua buah layanan yang dimiliki oleh SI UNPAR yaitu “Melihat Jadwal Kuliah” dan “Mengubah Nomor Telepon Mahasiswa”. Diharapkan prototipe tersebut dapat dikembangkan menjadi perangkat-perangkat lunak utuh sehingga bisa membuka semua layanan yang diperlukan agar dapat diakses oleh aplikasi perangkat lunak lainnya.

Kata Pengantar

Laporan penelitian ini merupakan rangkuman dari sejumlah eksplorasi kami dalam memodelkan antarmuka Web Services dan mengimplementasikannya dalam bentuk prototipe. Penelitian ini diperlukan agar sistem informasi perangkat lunak di Unpar dapat berkomunikasi secara *machine-to-machine* dengan perangkat lunak lainnya. Penelitian ini didanai oleh Unpar melalui LPPM.

Terima kasih kepada Ketua Jurusan Teknik Informatika, Dekan Fakultas Teknologi Informasi dan Sains atas dukungan dan arahan serta persetujuan atas usulan penelitian ini. Terima kasih juga kepada Ketua LPPM atas pendanaan yang diberikan. Selain itu juga Penulis sampaikan terima kasih kepada Sdr. Ingvarius atas keikutsertaannya dalam mengimplementasikan salah satu modul hasil rancangan dalam bahasa PHP & Java, dan rekan-rekan staf Biro Teknologi Informasi atas kesertaannya dalam proses pembuatan prototipe.

Akhir kata semoga laporan ini dapat memberikan gambaran motivasi, metodologi dan hasil dari penelitian yang telah dilakukan.

Ketua Peneliti,

Muhammad Ghifary, S.T., M.T.

Daftar Isi

Daftar Gambar

BAB 1 PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, tujuan dan hasil, metodologi penelitian, serta sistematika pembahasan.

1.1 Latar Belakang

Saat ini di Universitas Parahyangan telah berjalan berbagai sistem informasi (SI) perangkat lunak berbasis *web* yang membantu keberlangsungan kegiatan akademik dan administrasi. Sistem-sistem tersebut antara lain SI Akademik, SI Keuangan, SI Kepegawaian, dan SI Perpustakaan. Seperti pada umumnya perangkat lunak *web*, layanan yang diberikan bersifat *machine-to-human* yaitu perangkat lunak menyediakan sajian informasi kepada pengguna yang dapat diakses melalui *web browser*. Pengguna dapat menggunakan layanannya melalui komputer manapun yang memiliki *web browser* dan terhubung dengan jaringan internet. Dengan kemajuan teknologi *mobile device*, saat ini layanan aplikasi *web* juga dapat dinikmati melalui berbagai perangkat *mobile*/handphone yang semakin memberikan kemudahan kepada pengguna.

Seiring dengan perkembangan teknologi web dan perangkat *mobile device*, layanan pada web dapat disediakan secara *machine-to-machine* yang memungkinkan aplikasi perangkat lunak lainnya untuk menggunakan layanan atau fitur pada aplikasi web yang telah ada (*service reusability*). Teknologi tersebut dikenal dengan istilah *Web Services*. Dengan adanya *Web Services*, pengembang perangkat lunak tidak perlu membuat ulang suatu modul yang pernah dibuat sebelumnya. Misalnya, pada suatu perangkat lunak akan dibuat modul *search engine* seperti Google. Yang perlu dilakukan pengembang adalah membuat program untuk mengirimkan *request* ke dan menerima *response* dari Google. Pengerjaan pencarian sebenarnya diserahkan sepenuhnya kepada Google. Berbagai aplikasi web yang cukup ternama seperti Google, Facebook, Twitter, dan lain-lain, saat ini sudah menyediakan layanan berupa *Aplication Programming Interface* (API) agar bisa digunakan oleh aplikasi lainnya. Dengan keberadaan API tersebut, muncullah berbagai perangkat lunak Google, Facebook, ataupun Twitter yang berjalan pada platform yang berbeda-beda, khususnya pada platform mobile device diantaranya J2ME, Android, iOS, BlackBerry OS, dan sebagainya. Performa perangkat lunak tersebut jauh lebih baik dibandingkan jika pengguna menggunakan *web browser*.

Agar SI berbasis web yang dimiliki Unpar juga makin berkembang, perlu untuk dikembangkan API agar layanannya dapat digunakan oleh aplikasi lainnya yang berjalan pada platform yang berbeda. Pada penelitian ini akan dikembangkan suatu model antarmuka *Web Services* yang dapat merealisasikan komunikasi *machine-to-machine* antara SI di Unpar dengan aplikasi lainnya yang tetap menjaga independensi, reliabilitas, dan

sekuritas. Model tersebut diharapkan pula dapat memudahkan pengembang selanjutnya untuk melakukan revisi atau penambahan modul API yang baru.

Untuk memastikan apakah model tersebut dapat berjalan sesuai hasil rancangan, akan diimplementasikan prototipe perangkat lunak-perangkat lunak yang mensimulasikan elemen-elemen pada model rancangan. Sebagai penyedia antarmuka *web services*, akan dikembangkan prototipe perangkat lunak Web Service Server dalam platform J2EE yang dapat berkomunikasi dengan aplikasi *client* dan juga aplikasi *web server* utama. Sebagai peminta layanan kepada *web server* utama, akan dikembangkan sebuah prototipe aplikasi *client* dalam bahasa pemrograman Java. Untuk kebutuhan keamanan, akan dikembangkan pula sebuah aplikasi *web server* untuk autentikasi pengguna yang berbasis *single sign-on*.

1.2 Rumusan Masalah dan Batasan

Berdasarkan latar belakang di atas, maka pada penelitian ini dirumuskan masalah yang akan ditangani adalah:

1. Bagaimana memodelkan antarmuka *web services* untuk kebutuhan sistem informasi-sistem informasi yang ada di UNPAR, mencakup pemilihan teknologi yang akan digunakan dan isu keamanan, agar layanannya dapat dimanfaatkan oleh pihak/aplikasi/sistem lain?
2. Bagaimana merealisasikan model antarmuka *web services* yang telah dirancang pada Sistem Informasi Akademik UNPAR dalam bentuk prototipe?

Agar lebih fokus, dalam penelitian ini dibatasi pada hal-hal berikut:

1. Layanan SI yang akan dibuka hanya 2 fitur pada *student portal*, yaitu 1 operasi *write/PUT* dan 1 operasi *read/GET*
2. Bahasa pemrograman yang digunakan adalah Java dengan teknologi *Java 2 Enterprise Edition* (J2EE) untuk membangun perangkat lunak Java berbasis Web.

1.3 Tujuan dan Hasil yang Diharapkan

Berdasarkan rumusan dan latar belakang di atas, maka tujuan yang ingin dicapai pada penelitian ini adalah menghasilkan sebuah model antarmuka *web services* beserta prototipenya agar sistem informasi-sistem informasi yang berada di Unpar dapat berkomunikasi secara machine-to-machine dengan perangkat lunak lainnya.

Oleh karena itu, hasil akhir yang diharapkan adalah perangkat lunak uji yang dapat menghasilkan output berupa:

1. Diagram model Web Services

2. Prototipe perangkat lunak yang terdiri atas berbagai modul program yang berbeda platform

1.4 Metodologi Penelitian

Penelitian ini dilakukan dengan metode rekayasa produk, khususnya produk perangkat lunak. Tahapan yang dilalui antara lain:

1. Studi dan eksplorasi tentang referensi yang diperlukan sesuai kajian pustaka.
2. Analisis kebutuhan penggunaan fitur yang akan dibuka melalui *web services*
3. Perancangan model antarmuka *web services* yang dibuat berdasarkan kebutuhan penggunaan fitur atau data pada sistem terkait.
4. Rekayasa perangkat lunak prototipe *web services*
5. Pengujian prototipe yang telah dibangun

1.5 Sistematika Pembahasan

Laporan penelitian ini disajikan dengan sistematika sebagai berikut:

1. Bab 1 Pendahuluan dijelaskan tentang latar belakang, rumusan masalah, tujuan, hasil, metodologi dan sistematika pembahasan.
2. Bab 2 Studi Pustaka, berisi kajian tentang aplikasi telematika, multi threading dan sistem aplikasi berbasis web.
3. Bab 3 Analisis Masalah dan Desain Solusi, berisi pembahasan masalah dan gambaran solusi serta disain dari solusi yang diajukan.
4. Bab 4 Implementasi dan Pengujian, membahas hasil implementasi dan pengujian pernagkat lunak yang dilakukan.
5. Bab 5 Kesimpulan dan Potensi Pengembangan, berisi kesimpulan yang diambil berdasarkan hasil penelitian dan potensi pengembangan yang mungkin untuk penelitian selanjutnya.

BAB 2 STUDI PUSTAKA

Pada bagian ini dibahas tentang hasil studi pustaka yang berhubungan dengan peneltian ini.

2.1 Sistem Informasi di UNPAR

Untuk mendukung kegiatan akademik maupun administratif universitas, saat ini Universitas Katolik Parahyangan (UNPAR) telah memiliki beberapa aplikasi sistem informasi berbasis *web*, yaitu :

1. Sistem Informasi Akademik
   Sistem informasi yang khusus menangani segala bentuk kegiatan akademik yang dilangsungkan pada Unpar. Pengguna utama dari sistem ini adalah dosen dan mahasiswa. Di dalam SI Akademik, terdapat sebuah subsistem bernama *studentportal* yang khusus untuk mengelola segala kegiatan kemahasiswaan. Beberapa fitur yang dimiliki pada SI Akademik antara lain:
   - Pengelolaan mata kuliah
   - Pengelolaan jadwal kuliah
   - Pengelolaan data nilai
   - Fitur FRS dan PRS
2. Sistem Informasi Keuangan
   Sistem informasi yang khusus digunakan oleh biro keuangan untuk pengelolaan *cashflow* yang ada di Unpar.
3. Sistem Informasi Kepegawaian
   Sistem informasi yang khusus digunakan oleh biro kepegawaian untuk pengelolaan data personil seluruh *civitas academia* di Unpar.
4. Sistem Informasi Perpustakaan
   Sistem informasi yang khusus digunakan oleh biro perpustakaan untuk pengelolaan data buku yang ada di perpustakaan pusat Unpar.

2.2 Konsep Web Services

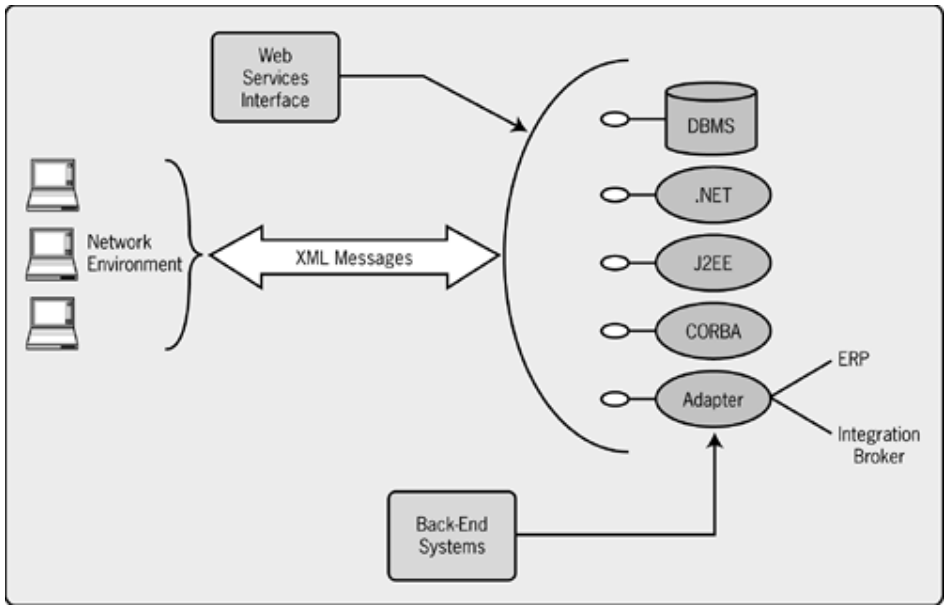
Pada bagian ini akan dijelaskan konsep dasar dari *Web Services* yang mencakup definisi, arsitektur, operasi, dan jenis teknologi.

2.2.1 Definisi Web Services

World Wide Web Consortium (W3C) mendefinisikan *Web Services* sebagai suatu sistem perangkat lunak yang dirancang untuk mendukung interoperabilitas dan interaksi antar mesin/sistem pada suatu jaringan [9]. *Web service* digunakan sebagai suatu fasilitas yang

disediakan oleh suatu web site untuk menyediakan layanan (dalam bentuk informasi) kepada sistem lain, sehingga sistem lain dapat berinteraksi dengan sistem tersebut melalui layanan-layanan (*services*) yang disediakan oleh suatu sistem yang menyediakan web service. Web service menyimpan data informasi dalam format pesan universal (misal: XML dan JSON), sehingga data ini dapat diakses oleh sistem lain walaupun berbeda platform, sistem operasi, maupun bahasa *compiler*.

Web service bertujuan untuk meningkatkan kolaborasi antar pemrogram dan perusahaan, yang memungkinkan sebuah fungsi di dalam *web services* dapat dipinjam oleh aplikasi lain tanpa perlu mengetahui detil pemrograman yang terdapat di dalamnya. Ilustasi mengenai posisi *web services* terhadap aplikasi lainnya dijelaskan pada gambar di bawah ini.



Gambar 2-1 Antarmuka Web Services terhadap Sistem Lainnya [3]

Beberapa alasan mengapa digunakannya *web services* adalah sebagai berikut:

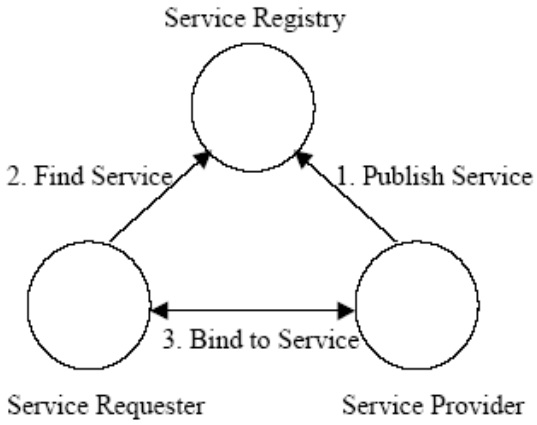
1. Web service dapat digunakan untuk mentransformasikan satu atau beberapa bisnis logic atau class dan objek yang terpisah dalam satu ruang lingkup yang menjadi satu, sehingga tingkat keamanan dapat ditangani dengan baik.
2. Web service memiliki kemudahan dalam proses deployment-nya, karena tidak memerlukan registrasi khusus ke dalam suatu sistem operasi. Web service cukup di-upload ke web server dan siap diakses oleh pihak-pihak yang telah diberikan otorisasi.
3. Web service berjalan di port 80 yang merupakan protokol standar HTTP, dengan demikian web service tidak memerlukan konfigurasi khusus di sisi firewall.

4. Web service dapat meminimalisir kesalahan entri data/informasi yang timbul pada komunikasi *machine-to-human* karena komunikasi dapat dilakukan secara *machine-to-machine*.

2.2.2 Arsitektur Web Services

Web service memiliki tiga entitas dalam arsitekturnya, yaitu:

1. **Service Requester** (peminta layanan)
2. **Service Provider** (penyedia layanan)
3. **Service Registry** (daftar layanan)



Gambar 2-2 Arsitektur umum *Web Services* [6]

- **Service Provider**: Berfungsi untuk menyediakan layanan/service dan mengolah sebuah registry agar layanan-layanan tersebut dapat tersedia.

- **Service Registry**: Berfungsi sebagai lokasi central yang mendeskripsikan semua layanan/service yang telah di-register.

- **Service Requestor**: Peminta layanan yang mencari dan menemukan layanan yang dibutuhkan serta menggunakan layanan tersebut.

2.2.3 Operasi-Operasi *Web Services*

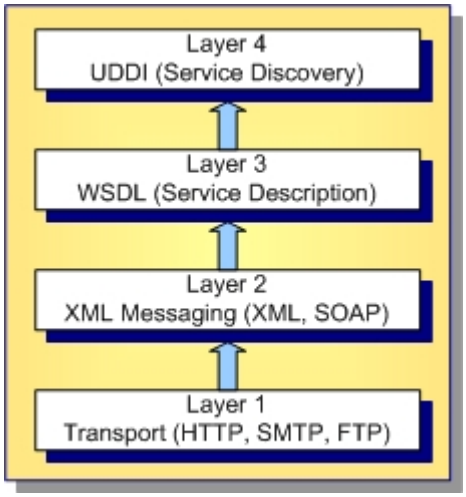
Secara umum, web service memiliki tiga operasi yang terlibat di dalamnya, yaitu:

1. **Publish/Unpublish**: Menerbitkan/menghapus layanan ke dalam atau dari registry.

2. **Find**: Service requestor mencari dan menemukan layanan yang dibutuhkan.
3. **Bind**: Service requestor setelah menemukan layanan yang dicarinya, kemudian melakukan binding ke service provider untuk melakukan interaksi dan mengakses layanan/service yang disediakan oleh service provider.

2.2.4 Big Web Services

Istilah "*Big Web Services*" mengacu pada teknologi *web services* yang paling awal dikembangkan, yaitu menggunakan format pesan dalam bentuk *eXtensible Markup Language* (XML) dan menggunakan protokol SOAP untuk penukaran pesan yang sebelumnya telah menjadi standar di sistem *enterprise* tradisional. Awalnya teknologi berbasis Service-Oriented Architecture (SOA) ini hanya disebut dengan istilah *Web Services* saja, namun karena saat ini berkembang jenis *web services* lainnya, maka istilah untuk teknologi ini ditambahkan dengan kata "Big" untuk membedakannya dengan jenis lain tersebut.



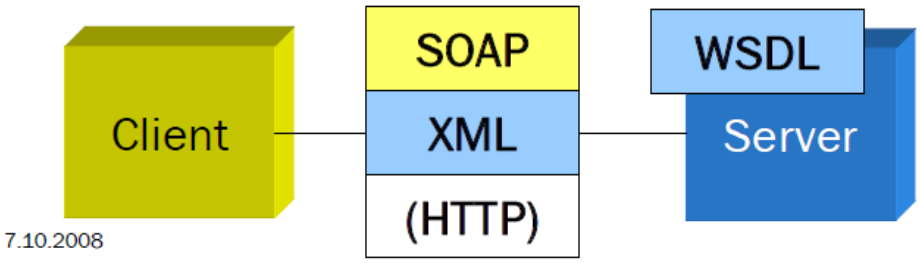
Gambar 2-3 Layer Komponen pada Big Web Services [10]

Big Web Services secara keseluruhan memiliki empat layer komponen seperti pada gambar di atas, yaitu:

1. Layer 1: Protokol internet standar seperti HTTP, TCP/IP, SMTP, FTP
2. Layer 2: Simple Object Access Protocol (SOAP), merupakan protokol akses objek berbasis XML yang digunakan untuk proses pertukaran data/informasi antar layanan.
3. Layer 3: Web Service Definition Language (WSDL), merupakan suatu standar bahasa dalam format XML yang berfungsi untuk mendeskripsikan seluruh layanan yang tersedia.

4. Layer 4: Universal Description, Discovery, and Integration (UDDI), merupakan sebuah metode/protokol yang merepresentasikan cara untuk mempublikasikan (*publish*) dan menemukan (*find*) *web services* di World Wide Web.

Jika diilustrasikan dengan lebih sederhana dari sudut pandang client-server, Big Web Services dapat digambarkan seperti di bawah ini :



Gambar 2-4 Skema Client-Server dari Big Web Services

2.2.5 REST Web Services

REST merupakan singkatan dari Representational State Transfer. Istilah REST atau RESTful pertama kali diperkenalkan oleh Roy Fielding pada disertasinya di tahun 2000 [11]. REST bukanlah sebuah standar protokol *web services*, melainkan hanya sebuah gaya arsitektur. Ide dasar dari arsitektur REST adalah bagaimana menghubungkan jalur komunikasi antar mesin/aplikasi melalui HTTP sederhana. Sebelum adanya REST, komunikasi antar mesin/aplikasi dilakukan dengan menggunakan beberapa mekanisme atau protokol *middleware* yang cukup kompleks seperti DCE, CORBA, RPC, ataupun SOA.

Arsitektur REST mampu mengeksploitasi berbagai kelebihan dari HTTP yang digunakan untuk kebutuhan *web services*. Walaupun Big Web Services juga dapat menggunakan protokol HTTP, namun hanya terbatas untuk kebutuhan *transport* saja. HTTP sendiri merupakan sebuah protokol standar di dunia World Wide Web yang berbasis *synchronous request/response*. Protokol tersebut sangat sederhana: *client* mengirimkan sebuah *request message* yang mencakup HTTP method yang akan diinvokasi, lokasi *resource* dalam format URI, serta pilihan format pesan (pada dasarnya dapat berupa format apa saja seperti HTML, plain text, XML, JSON, ataupun data binary), kemudian *server* akan mengirimkan response sesuai dengan spesifikasi yang diminta oleh *client*. Selama ini, yang berfungsi sebagai aplikasi *client* adalah sebuah *web browser* yang memfasilitasi komunikasi antara mesin dengan manusia. Dengan adanya REST, aplikasi *client* dapat berupa aplikasi apa saja hanya dengan memanfaatkan HTTP.

Berikut ini beberapa prinsip arsitektur dari REST yang dikutip dari sebuah buku berjudul “RESTful Java with JAX-RS” [] :

Addressability

Addressability merupakan sebuah ide dimana setiap objek dan *resources* pada suatu sistem dapat dicapai hanya dengan melalui sebuah *unique identifier*. Pada dunia REST, addressability dikelola dengan penggunaan Uniform Resource Identifier (URI). Format sebuah URI telah distandardisasi seperti di bawah ini :

```
scheme://host:port/path?queryString#fragment
```

"Scheme" merupakan nama prokotol yang akan digunakan. Untuk REST web services, prokotol yang biasa digunakan adalah HTTP atau HTTPS. "Host" merupakan nama DNS atau IP address dan diikuti sebuah "port" yang bersifat opsional. Berikut ini contoh pengaksesan resource melalui URI :

```
https://example.com/customers?lastName=Ghifar&zipcode=40293
```

Constrained & Uniform Interface

Ide dari prinsip ini adalah menyediakan berbagai layanan melalui antarmuka atau pemanggilan *method/procedure* yang seragam. Pada sistem CORBA ataupun SOAP, pengembang *client* harus mengetahui *method* apa saja yang disediakan oleh web service server. Pemanggilan method tersebut dikenal dengan istilah RPC (Remote Procedure Call). Namun pada sistem REST, *method*/*procedure* yang digunakan untuk layanan apapun hanyalah method-method yang disediakan pada HTTP. Istilah yang biasa digunakan untuk menyatakan prinsip *uniform interface* pada REST adalah CRUD (Create, Read, Update, Delete). Berikut ini ulasan detail mengenai method-method tersebut :

a. **GET**
   GET merupakan operasi *read-only* yang digunakan untuk meminta informasi spesifik pada server dalam bentuk *query*. Karakteristik dari operasi GET adalah *idempotent* dan *safe*. Idempotent berarti sebanyak-banyak apapun operasi ini dilakukan, hasilnya akan tetap sama. Sedangkan, *safe* berarti ketika operasi ini diinvokasi tetap tidak mengubah *state* di server.
b. **PUT**
   PUT merupakan operasi untuk meminta kepada server agar membuat sebuah resource baru.
c. **DELETE**
   DELETE digunakan untuk menghapus suatu *resource* tertentu.
d. **POST**
   POST merupakan operasi untuk membuat *resource* baru (PUT) ataupun memodifikasi resource yang telah ada.

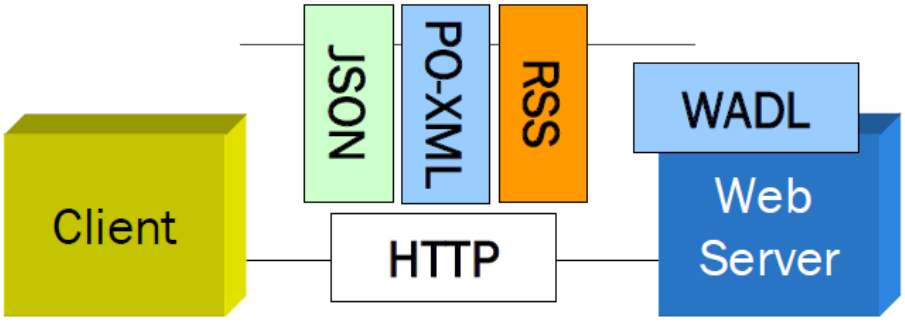
e. **HEAD**
   HEAD merupakan operasi yang mirip dengan GET, namun *response message* yang dikembalikan hanyalah berupa *response code* dan *message header*.
f. **OPTIONS**
   OPTIONS merupakan operasi yang mengembalikan informasi mengenai berbagai HTTP method yang didukung oleh suatu server, berguna untuk mengecek fungsionalitas sebuah server sebelum melakukan operasi sesungguhnya.

Adapun operasi-operasi HTTP lainnya yang tidak disebutkan di atas yaitu CONNECT dan TRACE, namun kedua operasi tersebut tidak penting ketika mendesain dan mengimplementasi REST Web Services. Alasan mengapa antarmuka yang seragam itu penting karena meningkatkan kualitas dari elemen-elemen berikut: *familiarity*, *interoperability*, dan *scalability*.

Stateless Communication

Pada dunia REST, seperti halnya pada dunia World Wide Web, *stateless* berarti tidak ada *client session data* yang disimpan pada server. Server hanya menyimpan dan mengelola *state* dari resource yang digunakan. Jika terdapat kebutuhan informasi yang *session-specific*, hal tersebut seharusnya diatur pada sisi *client*. Karakteristik tersebut memberikan kemudahan dari sisi *scalability* suatu server yang berbasis *cluster*. Untuk mengembangkan ukuran *cluster*, yang perlu dilakukan hanyalah menambah mesin baru serta tidak perlu memikirkan kepemilikan data terhadap *client* tertentu.

Jika diilustrasikan dengan lebih sederhana dari sudut pandang client-server, RESTful Web Services dapat digambarkan seperti di bawah ini :



Gambar 2-5 Skema Client-Server REST Web Services

2.2.6 Format Pesan Pertukaran

Saat ini terdapat 2 buah format pesan yang dipertukarkan (data interchange format) yang digunakan pada *web services*, yaitu XML dan JSON. XML adalah singkatan dari Extensible Markup Language yang merupakan turunan dari format Standard Generalized Markup Language (SGML) [15]. XML dirancang sebagai alternatif dari SGML untuk mengatasi masalah kompleksitas pada SGML itu sendiri. Pertimbangan fundamental akan penggunaan

XML adalah mencakup unsur *simplicity* dan *human readability* [16]. Contoh penulisan sintaks XML adalah sebagai berikut :

```
<name>
    <first>Muhammad</first>
    <last>Ghifary</last>
</name>
```

Gambar 2-6 Contoh penulisan sintaks XML

Sedangkan, JSON singkatan dari JavaScript Object Notation yang merupakan objek asli bawaan JavaScript [17]. JSON didesain sebagai format pesan pertukaran yang *human readable* serta mudah dibaca (*parsing*) oleh program komputer. Pada aplikasi berbasis JavaScript, penggunaan JSON sebagai format pesan pertukaran akan membawa dampak performa yang cukup signifikan dibandingkan bila menggunakan XML, karena penggunaan XML melibatkan *library* tambahan untuk membaca data dari Document Object Model (DOM) [18]. Berikut ini contoh penulisan sintaks dalam JSON :

```
{
   "firstname" : "Muhammad"
   "lastname" : "Ghifary"
}
```

Gambar 2-7 Contoh penulisan sintaks JSON

2.3 Keamanan Web Services

2.3.1 Teknologi SSL

Secure Socket Layer (SSL) merupakan sebuah teknologi yang memungkinkan web browser dengan web server untuk berkomunikasi melalui koneksi yang aman. Koneksi aman yang dimaksud adalah bahwa data yang dikirimkan akan dilakukan enkripsi terlebih dahulu kemudian akan didekripsi oleh penerima sebelum data tersebut diproses. Karena pada REST Web Services tidak memiliki fitur sekuritas khusus sebagaimana layaknya Big Web Service, SSL dapat dikatakan sebagai langkah pertama yang harus diimplementasikan untuk mengamankan komunikasi REST Web Service melalui jaringan HTTP.

SSL menjamin 3 pertimbangan sekuritas, yaitu sebagai berikut :

1. *Authentication*
2. *Confidentiality*
3. *Integrity*

Untuk menginstalasi dan mengkonfigurasi SSL pada sebuah *stand-alone web server*, komponen-komponen yang harus dimiliki adalah sebagai berikut:

- *Server certificate keystore*
- *HTTPS connector*

2.3.2 Instalasi dan Konfigurasi SSL pada Java

Hal yang pertama kali dilakukan untuk memasang SSL pada web server adalah membuat sebuah *server certificate keystore*. Berikut ini langkah-langkah yang dapat dilakukan untuk mendefinisikan *keystore* pada platform Java :

1. Meng-*generate server certificate*
   Perintah Java yang dapat digunakan untuk melakukan hal tersebut adalah *keytool*.

```
keytool -genkey -alias gif.bti.services -keyalg RSA -keypass changeit
-storepass changeit -keystore keystore.jks
```

   Setelah perintah tersebut dieksekusi, akan muncul *prompt* yang akan meminta informasi *first and last name*, *organization unit*, *locality*, *state*, dan *country code*. Yang perlu diperhatikan adalah field *first and last name* harus diisi dengan *server-name* (mis. localhost).

2. Eksport *server certificate* yang sudah terbentuk di keystore.jks ke the server.cer, dengan perintah sebagai berikut:

```
keytool -export -alias gif.bti.services -storepass changeit –file
server.cer –keystore keystore.jks
```

3. Untuk membuat sebuah file *trust-store* bernama *cacerts.jks* lalu menambahkan *server certificate* ke *trust-store*, jalankan kembali *keytool* pada direktori yang sama dengan lokasi server certificate dan keystore yang telah dibuat sebelumnya. Gunakan parameter-parameter sebagai berikut :

```
keytool -import -v –trustcacerts -alias gif.bti.services -file
server.cer  -keystore cacerts.jks -keypass changeit -storepass
changeit
```

Setelah mendefinisikan sebuah *server certificate keystore*, hal berikutnya yang harus dilakukan adalah mengkonfiguras *HTTPS connector* pada *web server*. Jika menggunakan *web server* Apache Tomcat, konfigurasi tersebut diatur pada file **$CATALINE_HOME/conf/server.xml**. ($CATALINE_HOME mengacu kepada direktori tempat instalasi Apache Tomcat). Secara *default*, HTTPS Connector pada Tomcat tidak dijalankan. UNtuk mengaktifkan connector tersebut, hanya dengan membuka sintaks komentar pada potongan kode server.xml sebagai berikut:

```
<Connector port="8443" maxThreads="150" scheme="https" secure="true"
SSLEnabled="true" clientAuth="false" sslProtocol="TLS" keyAlias="gif.bti.services"
```

```
keystoreFile="keystore.jks" keystorePass="changeit"/>
```

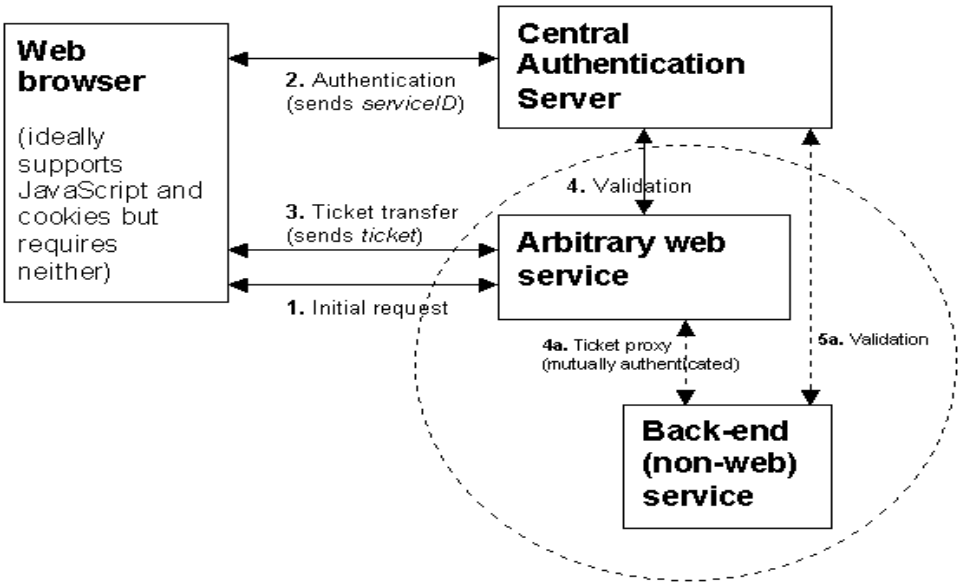
Gambar 2-8 Potongan Kode Konfigurasi HTTPS Connector pada Tomcat

2.4 Teknologi Central Authentication Service (CAS)

Central Authentication Service (CAS) merupakan solusi Single Sign-On yang bersifat *open-source* untuk layanan *web* [12]. Single Sign-On berarti pengguna dapat mengakses berbagai *resources* pada jaringan hanya dengan menggunakan satu akun pengguna saja. CAS dirancang untuk memenuhi kebutuhan-kebutuhan sebagai berikut [13]:

- Memfasilitasi fitur SSO pada berbagai aplikasi *web.*
- Memungkinkan layanan yang *untrusted* yang disediakan oleh suatu organisasi untuk dapat mengautentikasi pengguna tanpa harus mengakses *password* asli dari pengguna tersebut.
- Mempermudah prosedur yang harus diikuti suatu aplikasi untuk melakukan autentikasi.
- Memusatkan autentikasi utama hanya pada satu aplikasi *web* sederhana.

CAS didesain dan diimplementasikan sebagai sebuah *standalone web application*. Saat ini CAS baru dikembangkan sebagai *Java servlet* dan berjalan pada server HTTPS. Berikut skema penggunaan CAS sebagai autentikator untuk layanan web :



Gambar 2-9 Skema Penggunaan CAS

BAB 3 ANALISIS KEBUTUHAN DAN DISAIN SOLUSI

Pada bagian ini dijelaskan tentang analisis kebutuhan perangkat lunak uji performansi dan kapasitas sistem berbasis web. Pada bagian ini juga dijelaskan bagaimana disain dan implementasi dari perangkat lunak tersebut pada lingkungan Java.

3.1 Analisis Kebutuhan

Agar SI berbasis web yang dimiliki Unpar juga makin berkembang, perlu untuk dikembangkan API agar layanannya dapat digunakan oleh aplikasi lainnya yang berjalan pada platform yang berbeda. Ekspektasi yang diharapkan di kemudian hari adalah munculnya berbagai aplikasi perangkat lunak SI Unpar yang dapat berjalan di berbagai perangkat *mobile* yang berbasis integrasi, yang berarti pengembang perangkat lunak yang baru tidak perlu membuat ulang lojik dari program seperti halnya dengan versi web, melainkan hanya memanfaatkan layanan/fitur yang telah ada pada versi web.

Kebutuhan lainnya yang terkait dengan penggunaan SI di Unpar adalah transfer informasi secara *machine-to-machine* kepada pihak luar yang membutuhkan, misalnya kepada organisasi pemerintah pengelola pendidikan seperti DIKNAS, DIKTI, BAN-PT. Selama ini jika ada permintaan informasi dari pihak-pihak luar tersebut, proses transfer informasi dilakukan secara manual atau melibatkan entri data ulang di sistem informasi miliki pihak tersebut yang dilakukan oleh manusia. Diharapkan di kemudian hari SI pihak lain dapat berintegrasi dengan SI di Unpar agar menghilangkan jalur data entri sehingga meminimalisir kesalahan yang biasa terjadi pada cara manual.

Oleh karena itu, pada penelitian ini akan dirancang suatu model antarmuka *Web Services* yang dapat merealisasikan komunikasi *machine-to-machine* antara SI di Unpar dengan aplikasi lainnya yang tetap menjaga independensi dan sekuritas. Model tersebut diharapkan pula dapat memudahkan pengembang selanjutnya untuk melakukan revisi atau penambahan modul API yang baru.

Seperti yang telah tertera pada batasan masalah dan tujuan penelitian, akan dikembangkan pula prototipe hasil implementasi dari model yang telah dirancang. Prototipe tersebut mensimulasikan mekanisme aliran data yang terjadi pada model. Karena di Unpar terdapat 4 buah sistem informasi perangkat lunak dengan fitur-fitur atau layanan yang sangat banyak, pada prototipe ini hanya akan mengkaji 2 buah layanan pada *Student Portal*, yaitu sebagai berikut :

1. Mengambil informasi jadwal kuliah

2. Mengubah nomor telepon mahasiswa

Alasan mengapa dipilihnya kedua layanan tersebut adalah karena masing-masing mewakili operasi *read* (GET) dan operasi *write* (PUT, POST). Layanan-layanan berikutnya yang akan dibuka pada jalur *Web Services* dapat dipastikan berkarakteristik antara operasi *read* atau *write*.

3.2 Analisis Perbandingan Teknologi *Web Services*

Pada studi pustaka dideskripsikan 2 buah jenis Web Services yaitu *Big Web Services* dan *REST Web Services*. Saat ini telah banyak peneliti dan pengembang perangkat lunak yang mencoba membandingkan kekuatan dan kelemahan antara "Big" dan "REST". Menurut C. Pautasso, O. Zimmermann, dan F. Leymann [8], pemilihan antara kedua teknologi tersebut untuk diimplementasikan pada sebuah proyek integrasi perangkat lunak merupakan pemilihan keputusan arsitektural yang penting karena akan mempengaruhi proses pengembangan selanjutnya.

Berikut ini rangkuman perbandingan antara teknologi Big dan REST yang dikompilasi dari berbagai sumber:

Kriteria	BIG (SOAP)	REST
Orientasi	pembungkus lojik bisnis	akses resources/data
Sudut Pandang Developer	object-oriented	resource Oriented
Kebergantungan dengan bahasa pemrograman	tidak bergantung	tidak bergantung
Kebergantungan dengan platform	Tidak bergantung	Tidak bergantung
Kompleksitas	kompleks	sederhana
Standard-based	ya	tidak
Sekuritas	SSL, WS-Security	SSL
Transaksi	WS-Atomic Transaction	tidak terdefinisi
Reliabilitas	WS-ReliableMessaging	application-specific
Performa	relatif lambat	cepat
Caching	tidak ada	operasi GET dapat di-*cache*-kan
Dukungan protokol transport	HTTP, SMTP, JMS	HTTP
Ukuran pesan	cukup besar, terdiri atas bahasa mark-up SOAP dan WS-*	cukup kecil, tidak perlu bahasa markup XML tambahan
Protokol pesan	XML	XML, JSON, dan tipe MIME

		yang valid lainnya
Encoding pesan	ada (mendukung encoding ke *text* dan *binary*)	tidak ada
Service Description	WSDL	tidak terdefinisi standar formal tertentu
Penggunaan Kakas Perangkat Lunak / Framework	Dibutuhkan	Minimal atau tidak dibutuhkan sama sekali
Dukungan *attachment*	melalui spesifikasi SAAJ	melalui *RESTful WS implementation specific support*

Gambar 3-1 Perbandingan antara Big (SOAP) Web Services dengan REST Web Services

Pertanyaan selanjutnya adalah kondisi lingkungan perangkat lunak dan jaringan yang bagaimanakah yang sesuai untuk masing-masing jenis *web services* tersebut? Berikut ini kondisi-kondisi yang dimaksud yang dibahas pada salah satu sumber [14]:

Kondisi yang seusai untuk REST Web Services:

- *bandwidth* dan *resources* terbatas
- operasi *stateless*
- kebutuhan akan caching

Kondisi yang seusai untuk Big Web Services :

- proses dan invokasi asinkron
- kebutuhan akan spesifikasi formal
- operasi *stateful*

Meninjau model *web services* untuk SI di Unpar yang akan diimplementasikan, muncullah pertanyaan selanjutnya yaitu teknologi manakah yang paling sesuai untuk diimplementasikan pada *web services* Unpar? Pada subbab 3.3 akan dibahas detail mengenai model usulan termasuk penggunaan teknologi *web services* yang bersesuaian. Pada model tersebut diusulkan bahwa akan digunakan kedua teknologi tersebut pada jalur dan porsi yang berbeda.

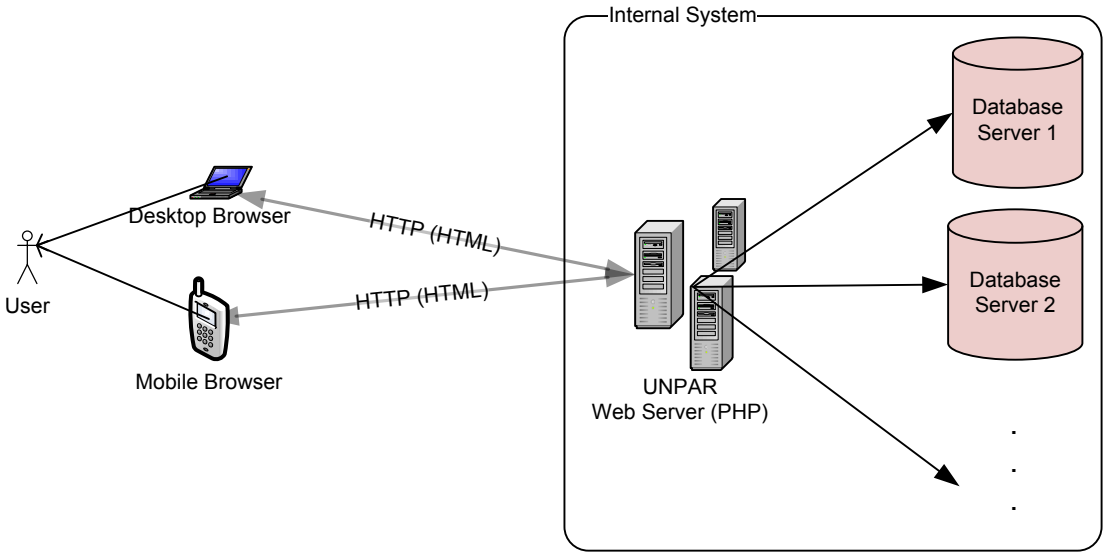
Selain jenis teknologi *web services*, hal lain yang penting untuk dipertimbangkan adalah format pesan yang akan dipertukarkan, yaitu antara XML dengan JSON. Perbandingan secara kuantitatif dan detail antara XML dengan JSON telah dilakukan oleh Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, Clemente Izurieta [19] melalui sebuah percoaan. Kesimpulan yang didapatkan dari percobaan tersebut adalah penggunaan format data JSON akan memberikan performa yang lebih cepat dan lebih sedikit memakan *resource* dibandingkan dengan penggunaan XML, yang lebih cocok apabila diimplementasikan pada lingkungan *web*. Kekuatan XML sendiri berada pada kemudahan *mapping* dengan

representasi objek pada aplikasi berorientasi objek (*object binding*) yang selama ini cukup lazim digunakan pada lingkungan SOA *middleware*. XML cocok digunakan apabila data yang transfer dalam bentuk dokumen yang berukuran besar yang membutuhkan pengelolaan yang mudah. Dikarenakan sistem informasi di UNPAR berbasis *web* dan akan lebih banyak digunakan layanannya pada aplikasi *mobile* dengan *resources* & *bandwidth* terbatas serta data yang ditransfer bukanlah berbasis dokumen, maka format pesan pertukaran yang akan digunakan pada model usulan adalah JSON.

3.3 Model *Web Services* Usulan

Seperti yang dijelaskan pada studi pustaka bahwa sistem informasi yang ada di Unpar terdiri dari berbagai perangkat lunak berbasis web dimana secara fisik diletakkan di beberapa *web server* dan terhubung dengan berbagai database server sebagai tempat penyimpanan data. Perangkat lunak tersebut hanya dapat diakses oleh *web browser* (komunikasi *machine-to-human*) melalui jaringan internet. Sebagian kecil aplikasi dibuka untuk umum dan sebagian besar lainnya hanya dapat diakses dari jaringan lokal.

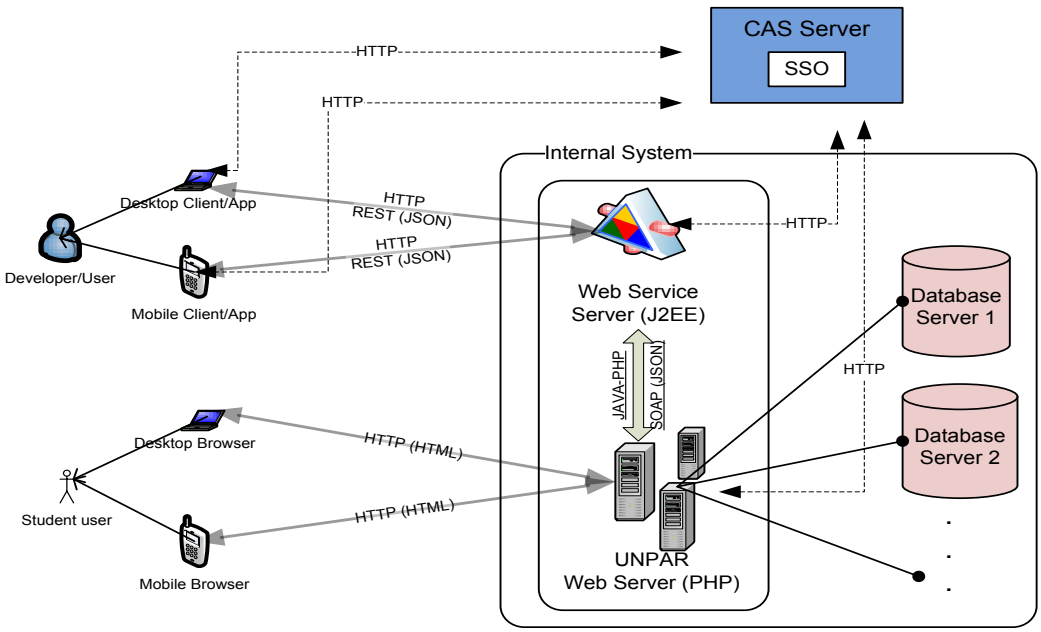
Jika disederhanakan dalam sebuah diagram, model keadaan sistem informasi kini yang ada di UNPAR dapat digambarkan sebagai berikut:



Gambar 3-2 Model Sistem Kini

Pada sistem kini, platform yang digunakan pada setiap *web server* adalah *Apache* yang dapat menjalankan bahasa pemrograman *server side* dalam PHP. Masing-masing aplikasi web terhubung langsung dengan *database server* yang juga dibangun dalam platform yang berbeda-beda. Untuk meminta dan menyimpan data ke database, digunakan berbagai jenis *query* dengan sintaks terkait berdasarkan *database server* yang mana yang diacu.

Dengan kondisi sistem kini tersebut, berikut ini diusulkan sebuah model baru untuk antarmuka *web services* pada sistem informasi yang ada di Unpar:



Gambar 3-3 Model Sistem Usulan untuk Antarmuka *Web Services* di Unpar

Model usulan di atas akan memungkinkan komunikasi *machine-to-machine* antara aplikasi-aplikasi SI di Unpar dengan perangkat lunak lainnya, baik yang berbasis *desktop* maupun yang berbasis *mobile*. Elemen utama yang menjadi pintu gerbang komunikasi *machine-to-machine* adalah *Web Service Server* yang akan dibangun dengan platform J2EE serta diimplementasi terpisah dari Unpar *Web Server* saat ini.

3.3.1 Pemisahan antara Web Service Server dengan UNPAR Web Server

Pada dasarnya penyediaan antarmuka *web services* dapat saja diimplementasi langsung pada UNPAR Web Server dengan mengembangkan sebuah modul berbasis PHP. Namun, penyediaan antarmuka *web services* dibuat terpisah dari *web server* utama, serta penggunaan platform J2EE untuk *WS Server* dikarenakan mempertimbangkan hal-hal sebagai berikut:

- ✓ **Jaminan keamanan pada level *database***
  Aplikasi *Web Service Server* dibuat terpisah dengan UNPAR Web Server agar tidak berkomunikasi secara langsung dengan *database* sistem. Dengan demikian, beberapa ancaman keamanan terkait *database* (misalnya, SQL injection) dapat diminimalisir.
- ✓ **Skalabilitas**

Selain masalah keamanan, Web Service Server yang tidak berkomunikasi langsung dengan *database* menyebabkan aplikasi tersebut tidak perlu menangani interaksi dengan *database* dalam bentuk *query*. Interaksi dengan menggunakan *query* hanya ditangani oleh UNPAR Web Server. Web Service Server hanya meminta informasi dari UNPAR Web Server dengan cara RPC (Remote Procedure Call) tanpa perlu memikirkan kerumitan *query* pada *database* UNPAR yang telah diimplementasi dengan platform yang berbeda-beda. Hal lainnya yang menjamin peningkatan skalabilitas adalah penggunaan platform J2EE yang dapat di-*cluster*. Apabila dikemudian hari perlu ditambahkan *resource* fisik untuk Web Service Server, maka yang dilakukan hanya menambahkan mesin saja.

- ✓ **Reliabilitas**
  Apabila terjadi *crash* pada modul *web services*, karena dibuat terpisah baik secara *hardware* maupun *software*, maka tidak akan mengganggu keberlangsungan jalannya *server* utama.

3.3.2 Penggunaan Teknologi *Web Services*

Terkait penggunaan teknologi *web services* serta mempertimbangkan hasil analisis perbandingan antar 2 teknologi web services pada subbab 3.2 , jalur komunikasi antara Web Service Server dengan Client Application akan digunakan arsitektur REST dengan format pesan JSON agar performa komunikasi berlangsung lebih cepat. Sedangkan, jalur komunikasi antara Web Service Server dengan UNPAR Web Server akan digunakan Semi-SOAP/Medium Web Service. Protokol yang digunakan pada jalur ini adalah SOAP, namun pesan yang dipertukarkan tetap dalam format JSON. Penggunaan SOAP pada jalur ini diperuntukkan untuk penanganan isu realibilitas dan sekuritas yang pada SOAP telah didefinisikan dan ditangani. Sedangkan arsitektur REST tidak memiliki mekanisme khusus untuk penanganan reliabilitas dan sekuritas.

3.3.3 Penggunaan CAS dan Keamanan Jaringan Komunikasi

Pada model usulan, mekanisme autentikasi dibuat dengan Single Sign-On baik untuk autentikasi login biasa melalui masukan pengguna maupun autentikasi penggunaan layanan *web services*. Khusus untuk kebutuhan *web services*, ketika ada permintaan layanan dari aplikasi *client*, apabila permintaan tersebut belum terautentikasi maka akan diarahkan ke aplikasi CAS untuk diautentikasi terlebih dahulu. Proses autentikasi *web services* pada CAS akan tetap memanfaatkan informasi *user login* dan *password* yang dilakukan pada aplikasi *client*. Apabila terautentikasi, maka CAS akan memberikan *ticket* kepada aplikasi *client* dan juga *ticket* yang sama kepada Web Service Server dan UNPAR Web Server. *Ticket* tersebut berfungsi sebagai *by-pass* bagi aplikasi *client* untuk mengakses *resource* apapun pada Web Service Server selama *ticket* tersebut valid.

Untuk masalah keamanan jaringan HTTP, akan digunakan protokol standar keamanan SSL (Secure Socket Layer) pada setiap jalur HTTP sehingga menjadi HTTPS. Hampir semua *web browser* yang tersedia saat ini telah mendefinisikan komunikasi secara HTTPS agar data yang terenkripsi selama berada di jaringan dapat dibaca pada *browser*. Karena komunikasi *web services* ini tidak menggunakan *web browser*, maka perlu didefinisikan sendiri fitur SSL pada Web Service Server yang berbasis J2EE. Teknologi SSL yang akan digunakan adalah JSSE (Java Secure Socket Extension).

3.3.4 Mekanisme Aliran Data *Web Services*

Secara keseluruhan, mekanisme aliran data operasi *read* dari aplikasi *client* hingga ke *database* melalui jalur *Web Services* adalah sebagai berikut:

1. Aplikasi *client* meminta layanan dalam bentuk *request packet* yang dilakukan secara RESTful melalui pengaksesan URI pada Web Service Server
    a. Apabila pada *request packet* tersebut telah terdefinisi CAS *ticket* yang telah terautentikasi, maka *request* akan diizinkan.
    b. Apabila belum terdapat *ticket* yang terautentikasi, maka akan diarahkan terlebih dahulu ke CAS untuk meminta *ticket*.
2. *Request* yang telah terautentikasi akan menyebabkan *Web Service Server* (J2EE) melakukan Remote Procedure Call dengan menggunakan protokol SOAP terhadap fungsi/prosedur pada UNPAR Web Server yang bersesuaian dengan *resource* yang diminta. RPC dilakukan dengan cara mengirimkan pesan *SOAP request* ke UNPAR Web Server. Pada pesan SOAP request tersebut juga akan diikutsertakan *ticket* terautentikasi untuk diverifikasi kembali di UNPAR Web Server.
3. Setelah menerima pesan pemanggilan RPC dari Web Service Server, UNPAR Web Server akan melakukan permintaan data ke database melalui *query* yang bersesuaian dengan resource yang diminta. Sebelum melakukan permintaan, UNPAR Web Server akan memastikan autentikasi terlebih dahulu melalui verifikasi *ticket*.
4. Data yang telah didapatkan dari *database* akan dikodifikasi oleh UNPAR Web Server ke dalam bentuk JSON lalu dikirimkan kembali ke Web Service Server.
5. Web Service Server mengirimkan kembali data dalam bentuk JSON tersebut ke aplikasi Client.

3.4 Rancangan URI dan Skema *Messaging*

Karena jalur *web services* antara aplikasi *client* dengan Web Service Server dilakukan secara RESTful, maka perlu dibuat rancangan URI yang sesuai dengan *resource* yang diminta untuk kemudahan pengaksesan layanan (*addressability*). Karena pada penelitian ini hanya akan diimplementasi 2 layanan yaitu fitur “Melihat Jadwal Kuliah” dan “Mengubah Nomor

Telepon Mahasiswa", oleh karena itu, rancangan URI yang akan diimplementasikan adalah sebagai berikut :

- `/btiservices/rest/studentportal/courseschedule/{NPM}/{tahun}/{semester}`
- `/btiservices/rest/studentportal/phonenumber/{NPM}/{no_telp}`

URI yang pertama digunakan untuk mengakses jadwal perkuliahan seorang mahasiswa pada tahun dan semester tertentu, sedangkan URI yang kedua digunakan untuk mengakses nomor telepon seorang mahasiswa. Perihal mengenai operasi apakah yang akan dikerjakan terhadap *resource* tersebut bergantung pada informasi operasi (GET atau PUT/POST) yang akan disisipkan pada *request header*.

Contoh skema isi pesan dalam bentuk JSON setelah operasi GET untuk layanan "Melihat Jadwal Kuliah" adalah sebagai berikut:

```
[{
    "namamk": "Arsitektur dan Organisasi Komputer",
    "kodemk":"AKT211",
    "hari":"Rabu",
    "dosen":"M. Ghifary",
    "kelas":"C",
    "jam":"08:00",
    "ruang":"9121"
},
{
    "namamk":"Dasar-Dasar Pemrograman",
    "kodemk":"AKT121",
    "hari":"Senin",
    "dosen":"Lionov",
    "kelas":"B",
    "jam":"10:00",
    "ruang":"10114"
}]
```

Gambar 3-4 Contoh Skema JSON untuk layanan "Melihat Jadwal Kuliah"

Sedangkan, skema JSON untuk layanan "Mengubah Nomor Telepon Mahasiswa" adalah sebagai berikut :

```
{
    "npm": "2011730041",
    "no_telp":"0222222222",
}
```

Gambar 3-5 Skema JSON untuk layanan "Mengubah Nomor Telepon Mahasiswa"

BAB 4 Hasil Implementasi dan Pengujian

Berikut adalah hasil implementasi perangkat lunak dari beberapa elemen yang telah dikaji pada model yang telah dirancang. Terdapat 4 jenis prototipe perangkat lunak yang diimplementasikan yaitu BTI Services Server, CAS Server, dan Aplikasi Klien Uji, yang mensimulasikan mekanisme aliran data dari model usulan.

Pada kondisi nyata, seharusnya masing-masing perangkat lunak berjalan pada perangkat *server* yang berbeda secara fisik. Pada penelitian kali ini, implementasi dan pengujian semua perangkat lunak akan dilakukan pada sebuah perangkat komputer yang sama.

4.1 Lingkungan Implementasi

Pembuatan dan pengeksekusian prototipe dijalankan pada lingkungan perangkat keras dan perangkat lunak yang sama dengan rincian sebagai berikut:

- Manufacture : AXIOO
- Processor : Intel® Core™ i7-2670QM CPU @ 2.20 GHz
- RAM : 4.00 GB (3.10 GB Usable)
- Harddisk : TOSHIBA MK7559GSXP 750 GB
- Operating System : Windows 7 32-bit

4.2 Implementasi BTI Services Server

Perangkat lunak prototipe BTI Services Server merupakan aplikasi Web Services Server yang menyediakan layanan *web services* untuk UNPAR Web Server. Aplikasi ini diimplementasi dengan kakas-kakas sebagai berikut:

- IDE : Eclipse Java EE Indigo SR1
- Web Server : Apache Tomcat 7.0.25 32-bit
- Additional Library : JAX-RS (Jersey 1.3), cas-client-core-3.2.1

4.2.1 JAX-RS (Jersey 1.3)

Library utama yang digunakan pada aplikasi ini adalah Jersey 1.3 yang memungkinkan pendefinisian URI untuk suatu method tertentu pada kelas Java. Selain URI *mapping*, pada Jersey juga sudah tercakup fitur-fitur pengolahan atau konversi berbagai format data bertipe MIME, termasuk JSON dan XML. Berikut daftar kumpulan file jar yang berada di dalam library Jersey:

- asm-3.1.jar
- jackson-all-1.9.3.jar
- jackson-core-asl-1.1.1.jar
- jersey-client-1.3.jar
- jersey-core-1.3.jar
- jersey-json-1.3.jar
- jersey-server-1.3.jar
- jettison-1.1.jar
- jsr311-api-1.1.1.jar

4.2.2 Kelas-Kelas Utama

Saat ini hanya terdapat 2 kelas utama yang dibuat pada BTI Services Server, yaitu kelas StudentPortal dan kelas CourseSchedule, yang berfungsi untuk membuka 2 layanan yang telah dibahas pada bab 3 yaitu "Melihat Jadwal Kuliah" dan "Mengubah Nomor Telepon Mahasiswa". Berikut *source code* dari kedua kelas tersebut:

```
package rest.studentportal;

import java.io.IOException;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.List;
import java.util.LinkedList;
import javax.ws.rs.PathParam;
import javax.ws.rs.Consumes;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import org.codehaus.jackson.annotate.JsonValue;
import org.codehaus.jackson.map.ObjectMapper;
import rest.studentportal.entities.CourseSchedule;

/**
 * REST Web Service
 *
 * @author Ghifar
 */
@Path("studentportal")
public class StudentPortal {

    //@Context
    //private UriInfo context;

    /** Creates a new instance of StudentPortal */
    public StudentPortal() {
    }

    /**
```

```
     * Retrieves representation of an instance of
studentportal.StudentPortal
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("application/json")
    public String getJson() {
        //TODO return proper representation object

        return "json string";
    }

    @GET
    @Produces("application/json")
    @Path("/courseschedule/{NPM}/{tahun}/{semester}")
    @JsonValue
    public String getCourseSchedule(@PathParam("NPM") String NPM,
                                    @PathParam("semester") String semester,
                                    @PathParam("tahun") String tahun){

        ObjectMapper mapper = new ObjectMapper();

        @SuppressWarnings("rawtypes")
                            List<Map> courseList = new LinkedList<Map>();

        CourseSchedule cs = new CourseSchedule();
        cs.setKode("AKT211");
        cs.setNama("Arsitektur dan Organisasi Komputer");
        cs.setDosen("M. Ghifary");
        cs.setHari("Rabu");
        cs.setJam("08:00");
        cs.setKelas("C");
        cs.setRuang("9121");
        courseList.add(cs.getHashMap());

        cs = new CourseSchedule();
        cs.setKode("AKT121");
        cs.setNama("Dasar-Dasar Pemrograman");
        cs.setDosen("Lionov");
        cs.setHari("Senin");
        cs.setJam("10:00");
        cs.setKelas("B");
        cs.setRuang("10114");
        courseList.add(cs.getHashMap());

        String jsonOutput = "";
        try {
            jsonOutput = mapper.writeValueAsString(courseList);
        } catch (IOException ex) {

Logger.getLogger(StudentPortal.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return jsonOutput;
    }

    /**
     * PUT method for updating or creating an instance of StudentPortal
     * @param content representation for the resource
```

```
     * @return an HTTP response with content of the updated or created
resource.
     */
    @PUT
    @Consumes("application/json")
    public void putJson(String content) {
    }

    @PUT
    @Path("/updatephonenum/{NPM}/{newphonenum}")
    @Consumes("application/json")
    public void updatePhoneNumber(String content) {
      SOAP.sendToUNPARWebServer(content);
    }
}
```

Gambar 4-1 Source Code kelas StudentPortal.java

```
package rest.studentportal.entities;

import java.util.HashMap;
import java.util.Map;

/**
 *
 * @author ghifar
 */
public class CourseSchedule {
    /**
     * ATTRIBUTES
     */
    private String kode;
    private String nama;
    private String dosen;
    private String hari;
    private String jam;
    private String kelas;
    private String ruang;
    /**
     * METHODS
     */
    public CourseSchedule(){
        this.kode = "[kode]";
        this.nama = "[nama]";
        this.dosen = "[dosen]";
        this.hari = "[hari]";
        this.jam = "[jam]";
        this.kelas = "[kelas]";
        this.ruang = "[ruang]";
    }

    //Getter

    public String getDosen() {
        return dosen;
    }

    public String getHari() {
```

```
        return hari;
    }

    public String getJam() {
        return jam;
    }

    public String getKelas() {
        return kelas;
    }

    public String getKode() {
        return kode;
    }

    public String getNama() {
        return nama;
    }

    public String getRuang() {
        return ruang;
    }

    //Setter

    public void setDosen(String dosen) {
        this.dosen = dosen;
    }

    public void setHari(String hari) {
        this.hari = hari;
    }

    public void setJam(String jam) {
        this.jam = jam;
    }

    public void setKelas(String kelas) {
        this.kelas = kelas;
    }

    public void setKode(String kode) {
        this.kode = kode;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }

    public void setRuang(String ruang) {
        this.ruang = ruang;
    }

    //Others
    public Map<String, Object> getHashMap(){
        Map<String, Object> courseMap = new HashMap<String, Object>();
        courseMap.put("kodemk",this.kode);
        courseMap.put("namamk",this.nama);
        courseMap.put("dosen",this.dosen);
```

```
        courseMap.put("hari",this.hari);
        courseMap.put("jam",this.jam);
        courseMap.put("kelas",this.kelas);
        courseMap.put("ruang",this.ruang);
        return courseMap;
    }
}
```

Gambar 4-2 Source Code kelas CourseSchedule.java

Kelas StudentPortal menggunakan *library* JAX-RS untuk mengelola URI yang dipasangkan dengan method tertentu. Contohnya adalah pada method **getCourseSchedule()**. Dengan menggunakan anotasi @Path, itu berarti method getCourseSchedule() akan dipanggil apa bila ada aplikasi *client* yang mengakses URI yang didefinisikan pada @Path, yaitu `/courseschedule/{NPM}/{tahun}/{semester}`. Yang diberi tanda kurung kurawal menandakan parameter masukan. Salah satu contoh URI lengkap yang akan memanggil fungsi getCourseSchedule() ketika URI tersebut diakses adalah seperti ini: *https://localhost:8443/btiservices/rest/courseschedule/2011730074/2012/ganjil*.

Sedangkan, kelas CourseSchedule merupakan kelas yang merepresentasikan struktur data dari jadwal kuliah. Kelas ini dibuat agar memudahkan pengelolaan data jadwal kuliah dan juga memudahkan untuk kebutuhan konversi format data dari representasi objek Java menjadi string JSON ataupun sebaliknya. Library yang bekerja untuk melakukan konversi tersebut adalah library *Jackson* yang merupakan bagian dari JAX-RS (Jersey).

4.2.3 Konfigurasi SSL

Karena BTI Services Center akan mengunakan jalur HTTPS yang menggunakan teknologi SSL Java Secure Socket Extension (JSSE), maka mula-mula harus didefinisikan Konfigurasi server.xml pada Web Server aplikasi BTI Services terlebih dahulu (lihat subbab 2.3.1). Pada percobaan ini, konfigurasi *server cerfiticate keystore* yang digunakan adalah sebagai berikut:

- [server-alias] : bti.gif.services
- [key-pass] : changeit
- [store-pass] : changeit
- [keystore-file] : keystore.jks

Perintah diatas memberikan output sebuah file *keystore.jks* yang berisi kunci enkripsi dan dekripsi yang dibutuhkan pada protokol HTTPS. Setelah itu, file *keystore.jks* diletakkan folder tempat instalasi Apache Tomcat 7.0.25. Kemudian dilakukan sedikit konfigurasi pada file *$TOMCAT-HOME/conf/server.xml* untuk menghidupkan HTTPS pada *web server*. Berikut isi konfigurasi dari *server.xml*.

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8666" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.security.SecurityListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="off" />
  <Listener className="org.apache.catalina.core.JasperListener" />
```

```
 <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
 <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
 <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

 <GlobalNamingResources>
  <Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />
 </GlobalNamingResources>
 <Service name="Catalina">
  <Connector port="8080" protocol="HTTP/1.1"
         connectionTimeout="20000"
         redirectPort="8443" />

  <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
SSLEnabled="true" maxThreads="150" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" keyAlias="gif.bti.services" keystoreFile="keystore.jks"
keystorePass="changeit"/>

  <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
  <Engine name="Catalina" defaultHost="localhost">
   <Realm className="org.apache.catalina.realm.LockOutRealm">
    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
         resourceName="UserDatabase"/>
   </Realm>
   <Host name="localhost"  appBase="webapps"
       unpackWARs="true" autoDeploy="true">

    <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
         prefix="localhost_access_log." suffix=".txt"
         pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
   </Host>
  </Engine>
 </Service>
</Server>
```

Gambar 4-3 Konfigurasi server.xml pada Web Server aplikasi BTI Services

Pada kode di atas, yang diberi warna merah merupakan konfigurasi Connector untuk jalur SSL sehingga memungkinkan aplikasi BTI Services Server untuk diakses dengan HTTPS pada URI-nya. Port yang dipilih sebagai tempat berjalannya HTTPS adalah port 8443.

4.3 Implementasi CAS Server

Untuk memungkinkan autentikasi Single Sign-On pada jalur web services, diimplementasikan sebuah aplikasi CAS yang berupa aplikasi *web*. Aplikasi CAS ini juga akan dijalankan pada *web server* Apache Tomcat 7.0.25 32-bit. *Library* yang perlu ditambahkan pada aplikasi CAS adalah *cas-server-core-3.4.11.jar* yang melakukan pekerjaan inti dari Single Sign-On.

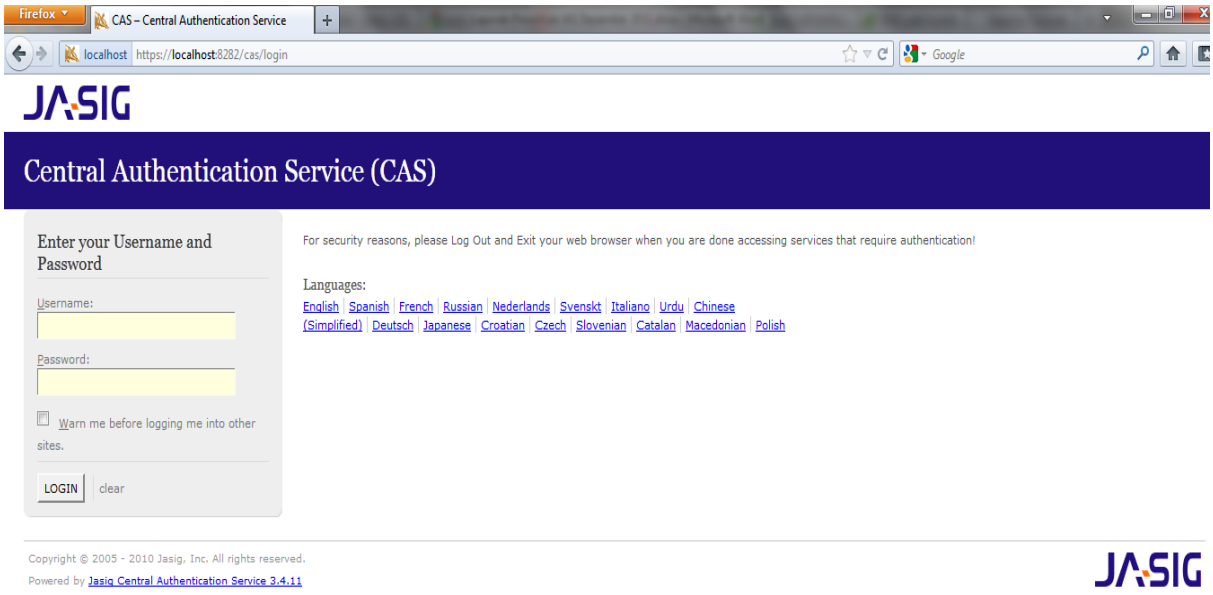
Karena CAS Server menggunakan *web server* yang sejenis dengan BTI Services Server, agar keduanya dapat berjalan berbarengan pada satu komputer, maka port yang akan dipakai sebagai tempat berjalannya CAS Server harus berbeda dengan port yang dipakai oleh BTI Services Server. Pada BTI Services Server, port yang dipakai adalah 8443. Pada percobaan ini, CAS Server akan dijalankan pada jalur HTTPS dengan menggunakan port 8282. Berikut konfigurasi pada file server.xml yang harus dilakukan :

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8111" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.security.SecurityListener" />
  <Listener className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="off" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener
className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener
className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <Service name="Catalina">
    <Connector port="8282"
protocol="org.apache.coyote.http11.Http11NioProtocol" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS"
                     keyAlias="gif.bti.services"
                     keystoreFile="keystore.jks"
                     keystorePass="changeit"
                     />

    <Engine name="Catalina" defaultHost="localhost">
      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
               resourceName="UserDatabase"/>
      </Realm>
      <Host name="localhost"  appBase="webapps"
```

```
            unpackWARs="true" autoDeploy="true">
        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
      </Host>
    </Engine>
  </Service>
</Server>
```

Gambar 4-4 Konfigurasi server.xml pada *web server* aplikasi CAS Server

Apabila semua konfigurasi di atas berhasil dilakukan, setelah *server* Apache Tomcat dihidupkan dan menjalankan *web browser* dengan URI terkait, maka akan muncul tampilan aplikasi CAS Server seperti di bawah ini.



Gambar 4-5 Tampilan Antarmuka Aplikasi CAS Server

4.4 Implementasi Aplikasi Klien Uji

Prototipe perangkat lunak terakhir yang diimplementasikan adalah aplikasi klien yang mengakses *resources* dari BTI Services Server. Aplikasi ini berfungsi untuk memastikan apakah layanan-layanan yang telah ada di BTI Services Server dapat berjalan atau tidak. Aplikasi tersebut berbentuk *Java Desktop Application* yang dibangun dengan kakas-kakas sebagai berikut:

- IDE : NetBeans IDE 7.0
- Additional Library : JAX-RS (Jersey 1.3), cas-client-core-3.2.1, javax.net.ssl

Berikut *source code* dari aplikasi klien yang dimaksud:

```
package rest.client.test;

import java.net.URI;
import java.security.NoSuchAlgorithmException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.ClientResponse;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import com.sun.jersey.client.urlconnection.HTTPSProperties;
import java.io.IOException;
import java.security.KeyManagementException;
import java.util.HashMap;
import java.util.Map;
import javax.net.ssl.KeyManager;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import org.codehaus.jackson.map.ObjectMapper;
import rest.client.ssl.MyX509KeyManager;
import rest.client.ssl.MyX509TrustManager;

public class BTITestWSSSL {

    public static void main(String[] args) {
        System.out.println("=== Aplikasi Client untuk Menguji BTI Services
Server ===");
        //Enabling SSL protocol
        TrustManager mytm[] = null;
        KeyManager mykm[] = null;

        try {
            mytm = new TrustManager[]{
                new MyX509TrustManager("keystore.jks",
"changeit".toCharArray())};
            mykm = new KeyManager[]{new MyX509KeyManager("keystore.jks",
"changeit".toCharArray())};
        } catch (Exception ex) {
        }

        SSLContext context = null;
        try {
            context = SSLContext.getInstance("SSL");
            context.init(mykm, mytm, null);
        } catch (NoSuchAlgorithmException ex) {

Logger.getLogger(BTITestWSSSL.class.getName()).log(Level.SEVERE, null, ex);
        } catch (KeyManagementException ex) {

Logger.getLogger(BTITestClientCAS.class.getName()).log(Level.SEVERE, null,
ex);
        }
```

```
        HTTPSProperties prop = new HTTPSProperties(null, context);

        ClientConfig config = new DefaultClientConfig();

config.getProperties().put(HTTPSProperties.PROPERTY_HTTPS_PROPERTIES,
prop);

        Client client = Client.create(config);
        WebResource serviceGet = client.resource(getBaseURI());


        String response =
serviceGet.path("rest").path("studentportal").path("courseschedule").
                path("2011730074").path("2011").path("ganjil").

accept(MediaType.APPLICATION_JSON).get(ClientResponse.class).toString();

        String content =
serviceGet.path("rest").path("studentportal").path("courseschedule").
                path("2011730074").path("2011").path("ganjil").
                accept(MediaType.APPLICATION_JSON).get(String.class);

        System.out.println("Melihat jadwal kuliah mahasiswa ber-NPM
2011730074 semester ganjil tahun 2011...");
        System.out.println(" >> Server Response : "+response);
        System.out.println(" >> Content (JSON) : "+content);


        WebResource servicePost =
client.resource(getBaseURI()+"/rest/studentportal/updatephonenum");
        Map<String, String> updatePhoneMap = new HashMap<String,String>();
        updatePhoneMap.put("2011730074","0227331517");

        ObjectMapper mapper = new ObjectMapper();

        String updatePhoneJSON = "";
        try{
            updatePhoneJSON = mapper.writeValueAsString(updatePhoneMap);
        }catch (IOException ex) {

Logger.getLogger(BTITestWSSSL.class.getName()).log(Level.SEVERE, null, ex);
        }


        System.out.println("UpdatePhonseJSON : "+updatePhoneJSON);
        ClientResponse clResponse =
servicePost.type("application/json").post(ClientResponse.class,updatePhoneJ
SON);
        System.out.println("Mengubah nomor telepon mahasiswa ber-NPM
2011730074 ...");
        System.out.println(" >> Server Response : "+clResponse);
        System.out.println(" >> Content (JSON) :
"+clResponse.getEntity(String.class));


    }

    private static URI getBaseURI() {
```

```
        return
UriBuilder.fromUri("https://localhost:8443/btiservices").build();
    }
}
```

Gambar 4-6 *Source Code* Aplikasi Klien Uji

4.5 Pengujian

Pada penelitian ini, tidak dilakukan pengujian untuk tujuan khusus. Yang dilakukan hanyalah menjalankan aplikasi klien uji untuk memastikan apakah model yang telah dirancang memang benar-benar dapat diimplementasikan atau tidak.

Setelah aplikasi klien uji dijalankan, hasil keluaran yang didapatkan adalah sebagai berikut:

```
=== Aplikasi Client untuk Menguji BTI Services Server ===

Melihat jadwal kuliah mahasiswa ber-NPM 2011730074 semester ganjil tahun
2011...

 >> Server Response : GET
https://localhost:8443/btiservices/rest/studentportal/courseschedule/201173
0074/2011/ganjil returned a response status of 200

 >> Content (JSON) : [{"namamk":"Arsitektur dan Organisasi
Komputer","kodemk":"AKT211","hari":"Rabu","dosen":"M.
Ghifary","kelas":"C","jam":"08:00","ruang":"9121"},{"namamk":"Dasar-Dasar
Pemrograman","kodemk":"AKT121","hari":"Senin","dosen":"Lionov","kelas":"B",
"jam":"10:00","ruang":"10114"}]

UpdatePhonseJSON : {"2011730074":"0227331517"}

Mengubah nomor telepon mahasiswa ber-NPM 2011730074 ...

 >> Server Response : POST
https://localhost:8443/btiservices/rest/studentportal/updatephonenum
returned a response status of 201

 >> Content (JSON) : {"2011730074":"0227331517"}
```

Gambar 4-7 Output Aplikasi Klien Uji

Dari hasil keluaran dapat disimpulkan bahwa baik layanan *web services* operasi GET (“Melihat Jadwal Kuliah”) maupun operasi POST (“Mengubah Nomor Telepon Mahasiswa”) yang diminta dari BTI Services Server keduanya-duanya berfungsi. Ini terlihat dari *response*

code yang didapatkan dari BTI Services Server yaitu masing-masing 200 (GET) dan 201 (POST) yang berarti berhasil menjalankan *method* tersebut.

BAB 5 KESIMPULAN DAN POTENSI PENGEMBANGAN

Pada bagian ini dijelaskan tentang kesimpulan yang diambil berdasarkan metodologi, kajian pustaka, dan hasil-hasil penelitian. Bagian ini diakhiri dengan potensi pengembangan yang dapat dilakukan untuk penelitian selanjutnya.

5.1 Kesimpulan

Berdasarkan hasil penelitian di atas, dapat disimpulkan:

1. Dihasilkan sebuah model hipotesis antarmuka *web services* berbasis REST untuk kebutuhan sistem informasi di UNPAR yang mencakup penggunaan teknologi yang sesuai, isu keamanan, reliabilitas, independensi, dan skalabilitas sistem.
2. Model antarmuka *web services* yang dirancang dapat diimplementasikan dengan menggunakan sejumlah prototipe perangkat lunak yaitu BTI Services Server, CAS Server, dan Aplikasi Klien Uji.

5.2 Potensi Pengembangan

Dari hasil penelitian yang telah dilakukan, beberapa hal yang dapat dikembangkan antara lain yaitu:

1. Pengimplementasian prototipe perangkat lunak menjadi perangkat lunak yang siap pakai yang kemudian akan di-*deploy* pada lingkungan nyata.
2. Pembangunan beberapa perangkat lunak klien pada platform lainnya (selain Java) untuk memastikan bahwa layanan *web services* benar-benar dapat dikonsumsi oleh perangkat yang berbeda platform.
3. Pengukuran performa dari antarmuka *web services* secara kuantitatif. Klaim bahwa penggunaan REST Web Services akan memberikan performa yang lebih baik dibandingkan Big Web Services belum terbukti sepenuhnya untuk kasus model antarmuka *web services* SI Unpar yang telah dirancang.
4. Kajian keamanan terhadap jaringan komunikasi yang diusung pada model. Pada model ini, unsur keamanan ditangani oleh protokol SSL dan autentikasi Single Sign-On pada CAS, yang mana belum dilakukan pengkajian secara komprehensif apakah sistem keamanan tersebut sudah benar-benar aman atau tidak.

BAB 6 DAFTAR PUSTAKA

1. Alonso, G., Casati F., Kuno H., Machiraju V., *Web Services: Concepts, Architectures, and Applications*, Springer, 2003.
2. Oracle Sun Developer Network (Oracle SDN), *The Java™ Web Services Tutorial for Java Web Services Developer's Pack v1.6*, http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html, Juni 2005.
3. Newcomer, E. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Independent Technology Guide, 2002.
4. Adams P., Easton P., Mehta B., Merrick R. *SOAP over Java Message Service 1.0*, http://www.w3.org/TR/2009/CR-soapjms-20090604/ , Juni 2009.
5. Chappel D. A., Jewell T. *Java Web Services*, O'Reilly Media, 2002.
6. Cerami E., *Web Services Essentials*, O'Reilly Media, 2002.
7. R.T. FIelding, R. N. Taylor. *Principled Design of the Modern Web Architecture*, in the Proceedings of the 22nd International Conference on Software Engineering, Ireland, June 2000 (ICSE 2000), 407 - 416.
8. C. Pautasso, O. Zimmermann, F. Leymann. *RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision.* in the Proceedings of the 17th International WWW Conference, April 2008, 805-814
9. W3C. Web Services Architecture. http://www.w3.org/TR/ws-arch/ . Diakses: Desember 2011
10. RAD Studio for Microsoft .NET. *Web Services Protocol*. http://docs.embarcadero.com/products/rad_studio/radstudio2007/RS2007_helpupdates/HUpdate4/EN/html/devnet/webservicesprotocol_xml.html. Diakses: Januari 2012
11. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctor of Philosophy Dissetation in Information and Computer Science, University of California, Irvine, 2000.
12. Jasig Community. *Central Authentication Service WIKI*. https://wiki.jasig.org/display/CAS/Home. Diakses : Desember 2011
13. Jasig Community. CAS 1 Architecture. http://www.jasig.org/cas/cas1-architecture. Diakses: Desember 2011
14. Technical Mumbo Jumbo.*Interview Question: Compare two web services type SOAP and RESTful (SOAP Vs RESTful)*. http://technicalmumbojumbo.wordpress.com/2011/01/15/interview-question-soap-restful-webservices-comparison-soap-vs-restful/ . Diakses: Januari 2012
15. J. Bosak, *"Xml, java, and the future of the web,"* World Wide Web Journal, 2(4):219-227, 1997.

16. Extensible markup language (xml) 1.0 (fourth edition). W3C, 2006.
17. JSON, json.org, http://www.json.org. Diakses: Januari 2012
18. W3C Document Object Model. W3C, 2005. http://www.w3.org/DOM. Diakses: Januari 2012
19. N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta. *Comparison of JSON and XML Data Interchange Formats: A Case Study*. in Proc. CAINE, 2009.