LAPORAN AKHIR
PENELITIAN

PENGEMBANGAN *SINGLE-ACCOUNT* UNTUK APLIKASI BERBASIS *WEB* DAN *DESKTOP*

PENELITIAN LABORATORIUM/LAPANGAN

Oleh:
Kusworo Anindito, S.T., M.T.

Program Studi Teknik Informatika
Fakultas Teknologi Industri
Universitas Atma Jaya Yogyakarta
2011

LEMBAR PENGESAHAN

1.	a. Judul Penelitian	:	PENGEMBANGAN *SINGLE-ACCOUNT* UNTUK APLIKASI BERBASIS *WEB* DAN DESKTOP
	b. Macam Penelitian	:	Laboratorium
2.	Peneliti		
	a. Nama	:	Kusworo Anindito, S.T., M.T.
	b. Jenis Kelamin	:	Laki-laki
	c. Usia saat pengajuan proposal	:	37 tahun 8 bulan
	d. Jabatan Akademik/Gol	:	Lektor / IIIc
	e. Fakultas/Program Studi	:	Teknologi Industri / Teknik Informatika
3.	Jumlah Peneliti	:	1 (satu) orang
4.	Lokasi Penelitian	:	Yogyakarta
5.	Jangka Waktu Penelitian	:	6 (enam) bulan
6.	Biaya yang disetujui	:	2.850.000,- (Dua juta delapan ratus lima puluh ribu rupiah)

Yogyakarta, 28 Februari 2011
Ketua Peneliti,

Kusworo Anindito, S.T., M.T.

Mengetahui,

Konsultan

Prof. Ir. Suyoto, M.Sc., Ph.D.

Kepala Lab Jaringan Komputer

Th. Devi Indriasari, ST, M.Sc.

Dekan FTI UAJY,

Ir. B. Kristyanto M.Eng., Ph.D.

Ketua LPPM UAJY,

Dr. MF. Shellyana Junaedi, S.E.

DAFTAR ISI

INTISARI

Seiring dengan bertambahnya aplikasi layanan teknologi informasi yang disediakan Universitas Atma Jaya Yogyakarta bagi mahasiswa, maka seorang mahasiswa harus mengingat berbagai *account* yang dimiliki untuk mengakses berbagai aplikasi tersebut. Mahasiswa tersebut memiliki *account* untuk mengakses email, situs kuliah, pengisian rencana studi, dan sebagainya. Hal ini sering membuat mahasiswa tersebut lupa *username* atau *password* salah satu aplikasi yang ada, sehingga tidak bisa mengakses sistem. Akibatnya, administrator/operator mendapatkan permohonan untuk mereset *password* para mahasiswa yang lupa tersebut. Penelitian ini dilakukan sebagai usaha untuk meminimalkan kemungkinan mahasiswa tidak bisa mengakses sistem informasi karena tidak bisa melakukan *login*. Mahasiswa cukup mengingat sebuah *account* untuk bisa mengakses berbagai layanan sistem informasi, baik *desktop* maupun *web*, yang disediakan kampus.

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Pengembangan sistem informasi untuk melayani mahasiswa sering kali dilakukan secara bertahap. Hal ini dilakukan suatu kampus karena keterbatasan dana dan sumber daya, ataupun munculnya layanan baru. Akibatnya tiap aplikasi membuat sistem *account* sendiri. Hal ini membuat seorang mahasiswa yang memanfaatkan berbagai layanan memiliki banyak *account* dengan *username* dan *password* yang berbeda.

Universitas Atma Jaya Yogyakarta telah mengembangkan sistem informasi akademik yang digunakan untuk mengelola informasi pribadi mahasiswa, pengambilan kuliah, nilai kuliah tiap semester, transkrip nilai, dan sebagainya untuk mahasiswa. Aplikasi ini berbasis *desktop*. Pihak universitas kemudian juga mengembangkan situs kuliah untuk mengelola informasi mengenai perkuliahan dan materi perkuliahan. Mahasiswa bisa mengunduh materi kuliah dan tugas dari aplikasi berbasis *web*. Saat ini universitas bekerjasama dengan Microsoft untuk memberikan layanan email, blog, dan media penyimpan *virtual*, dalam paket live@edu, yang ditujukan bagi seluruh sivitas akademik. Penggunaan *single-account* tentu saja akan membantu mahasiswa dalam menggunakan berbagai aplikasi ini.

1.2 Perumusan Masalah:

Rumusan masalah yang akan dijawab melalui penelitian yang akan dilakukan adalah sebagai berikut:

1. Bagaimana cara membuat *single-account* untuk aplikasi berbasis *web* dan desktop?
2. Bagaimana mengembangkan aplikasi yang ada agar bisa menggunakan *single-account* yang telah ditetapkan?

1.3 Tujuan Penelitian:

Penelitian yang akan dilakukan bertujuan untuk:

1. Mengembangkan aplikasi sehingga pengguna hanya memiliki satu *account*.
2. Melakukan tahap awal dari implementasi *single sign-on* pada sistem informasi yang ada di Universitas Atma Jaya Yogyakarta, yaitu dengan menerapkan *single-account*.

1.4 Metodologi:

Penelitian ini dilakukan dengan melakukan sejumlah aktivitas yang berkaitan, antara lain:

1. Studi Pustaka

   Studi pustaka dilakukan dengan mengumpulkan informasi dari buku-buku, artikel, maupun jurnal ilmiah yang membahas mengenai hal-hal yang terkait dengan *single-account* dan cara implementasiannya.

2. Analisis Kebutuhan Sistem.

   Analisis kebutuhan sistem dilakukan dengan menggali kebutuhan fungsional dari layanan aplikasi yang akan dikembangkan dan menentukan sejauh mana kebutuhan-kebutuhan tersebut akan diakomodasi dalam layanan aplikasi yang akan dibangun.

3. Perancangan Sistem.

   Perancangan sistem dilakukan untuk mendapatkan deskripsi mengenai arsitektural aplikasi dan deskripsi data.

4. Implementasi Sistem.

   Implementasi sistem dilakukan dengan menterjemahkan deskripsi perancangan yang telah dibuat ke dalam kode-kode program sesuai dengan tools yang digunakan untuk membangun aplikasi.

5. Pengujian Sistem

   Pengujian sistem dilakukan untuk menguji fungsionalitas aplikasi yang akan dibangun apakah sudah sesuai dengan spesifikasi yang telah ditentukan(menangani *single-account malware* untuk aplikasi *desktop* dan *web*).

6. Penulisan Laporan dan Dokumentasi

   Tahap ini dilakukan dengan membuat dokumentasi terhadap seluruh aktivitas penelitian dengan harapan dapat dipergunakan untuk penelitian lainnya.

1.5 Manfaat Hasil Penelitian:

Manfaat yang didapat dari penelitian yang akan dilakukan adalah sebagai berikut:

1. Mahasiswa, dosen, dan karyawan hanya perlu mengingat sebuah *account* untuk berbagai layanan yang dibutuhkannya.
2. Memudahkan dalam mengintegrasikan berbagai aplikasi sistem informasi di Universitas Atma Jaya Yogyakarta.

BAB 2. LANDASAN TEORI

2.1 Tinjauan Pustaka Mengenai *Single-Account*

Kendali akses adalah cara bagaimana administrator mengendalikan siapa saja yang boleh mengakses server dan layanan apa yang diperbolehkan bagi mereka (CISCO, 2010). Keamanan jaringan dengan *authentication* (otentikasi), *authorization* (otorisasi), dan *accounting* (AAA) menyediakan *framework* utama bagaimana administrator mengendalikan pengaksesan terhadap router atau server. Otentikasi menyediakan metode pengidentifikasian pengguna, termasuk dialog login dan *password*, *challenge* dan *response*, dukungan komunikasi pesan (*messaging*) dan enkripsi (tergantung protokol keamanan yang dipilih).

Otentikasi pada organisasi dengan berbagai aplikasi dilakukan dengan mekanisme *single sign-on* (SSO). SSO merupakan komponen yang sangat penting pada arsitektur keamanan sebuah organisasi (Ponnapalli, 2005). Dalam teknologi informasi diyakini bahwa implementasi SSO pada perusahaan mahal dan mungkin terus melebar. Bagaimanapun, perhatian perusahaan terhadap keuntungan pengimplementasian SSO semakin baik. Sebagian besar produk SSO yang ada di pasar, berdasarkan arsitekturnya, dapat dikategorikan menjadi dua tipe:

a. *Web-based* (juga dikenal sebagai *enterprise* SSO atau ESSO)
b. *Non web-based* (juga dikenal *legacy* SSO)

Berikut beberapa alasan penggunaan SSO:

a. Mengurangi kebutuhan bagi pengguna untuk mengingat banyak login dan *password*, yang secara tidak langsung mengurangi kemungkinan pengguna melakukan sesuatu yang tidak aman, seperti menuliskan *password* dan menempelkannya di tempat kerja.
b. Pemusatan pengendalian kebijakan keamanan pada aplikasi-aplikasi yang ada.
c. Memaksa batasan *password*, seperti panjang minimum, minimal memiliki satu karakter khusus, huruf besar, angka, dan sebagainya.
d. Pemaksaan penggantian *password* saat pertama kali login.
e. Memelihara histori *password* dan memastikan *password* baru tidak sama dengan *password* di histori
f. Mengakhiri masa berlaku *password* sepanjang waktu tertentu.

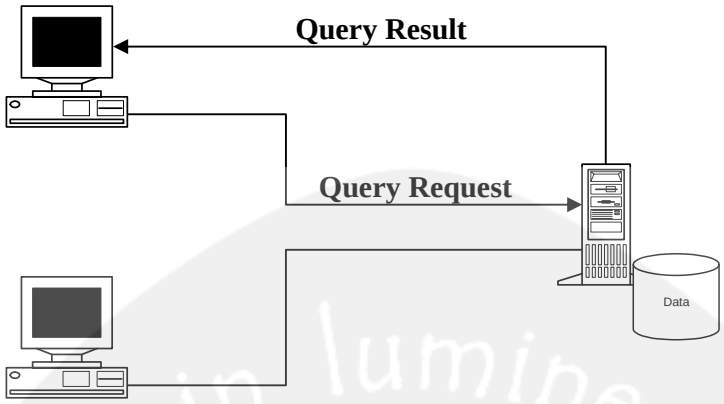
g. Mengakhiri sesi jika pengguna tidak aktif selama waktu tertentu atau batas waktu maksimal pengaksesan habis.
h. Kemampuan pengawasan dan pelaporan tingkat lanjut.

Seiring dengan berkembangnya banyak aplikasi *web*, dimana setiap aplikasi menyediakan fasilitas otentikasi bagi anggotanya, maka setiap pengguna cenderung memiliki banyak *account*. Otentikasi dengan *Lightweight Directory Access Protocol* (LDAP) memungkinkan setiap aplikasi berbasis *web* dapat secara terpadu menggunakan satu informasi identifikasi pengguna yang tersimpan di direktori server LDAP. Metode ini telah berhasil diimplementasikan pada Sistem Informasi Kepegawaian Universitas Indonesia (Sari, 2006).

SSO juga telah berhasil diimplementasikan si Universitas Bina Nusantara untuk menggabungkan aplikasi yang sudah ada di binus-access ke dalam sebuah *web* portal (Rudy, 2009). Dengan adanya *web portal* yang menggunakan SSO ini berarti pengguna hanya memiliki sebuah *username* dan *password*. Pengguna hanya butuh login sekali untuk mendapatkan semua layanan di *web* portal. Hal ini mempermudah mempermudah pengguna dalam menggunakan aplikasi dan tidak perlu mengingat banyak *account*.

2.2 *Client-Server Architecture*

Client-server merupakan sebuah model komputasi dimana sebuah aplikasi mengakses informasi dari aplikasi lainnya, bisa pada komputer yang sama atau pada komputer lain. Bagian *client* dari aplikasi tersebut dibangun untuk interaksi pengguna, sementara bagian *server* dibangun untuk menangani beberapa permintaan *client*. Model ini dapat diimplementasikan dalam lingkungan perangkat keras dan perangkat lunak yang berbeda-beda. Proses *client* dan *server* dapat dijalankan pada komputer yang sama ataupun berbeda, bahkan sebuah proses server dapat meminta layanan ke *server* lain.



Gambar 2.1 Arsitektur two-tier

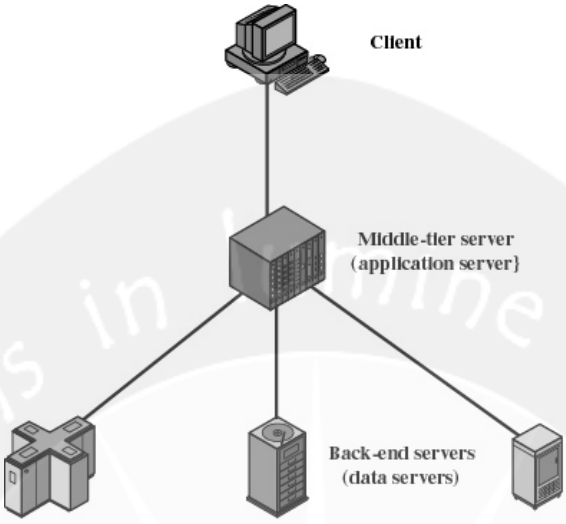
Dalam *two-tier system* terdapat program *client* dan program *server*. Perbedaan utama dari keduanya adalah bahwa *server* merespon permintaan dari banyak *client* yang berbeda, sementara *client* biasanya berinisiatif melakukan permintaan informasi dari *server*. Dalam arsitektur ini terdapat dua pilihan pembangunan aplikasinya, *fat client* dan *thin client* (*fat server*), tergantung fungsionalitas yang berada di *client*. Model *fat client-thin server* merupakan sebuah model dimana hampir semua fungsionalitas dan *business logic* berada pada *client*. Sebaliknya model *thin client-fat server* menunjukkan hampir semua fungsionalitas dan *business logic* berada di *server*.

Arsitektur *three-tier* merupakan sebuah perbaikan dari *two-tier architecture*. Aliran informasi pada dasarnya tetap linear: sebuah permintaan (*request*) datang dari *client* ke *server*; *server* tersebut mengambil atau menyimpan data dari/ke basisdata; basisdata tersebut mengembalikan informasi ke server; server tersebut mengembalikan informasi ke *client*. Ada tiga bagian penting yang paling mendasar pada aristektur ini, seperti terlihat pada Gambar 2.2 dan 2.3, adanya lapisan-lapisan yang menunjukkan bahwa arsitektur tersebut 3-*tier*, yaitu :

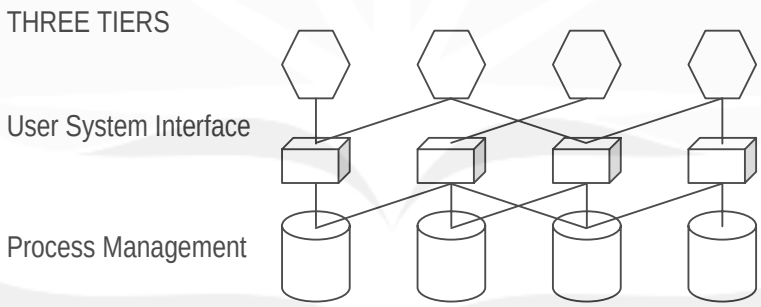
a. *User tier (presentation layer, user services).*
b. *Business tier (business logic layer, business services).*
c. *Data tier (data layer, data services).*

Perbedaan yang paling menonjol dari *2-tier* dengan *3-tier* yaitu adanya *middle tier* pada arsitektur *3-tier* yaitu *business logic layer*. Arsitektur *3-tier* muncul karena adanya keterbatasan pada arsitektur *2-tier*, dimana jika ada perubahan fungsi suatu komponen pada satu sisi mempengaruhi kedua sisi yaitu *client* dan *server*. Hal ini tidak terjadi pada sistem *3*-tier. Arsitektur *3-tier* digunakan ketika diperlukan suatu rancangan *client-server* yang

efektif, dimana meningkatkan kinerja, fleksibilitas, kemudahan perawatan, kemampuan untuk dapat digunakan ulang, dan skalabilitas.



Gambar 2.2 *Three-tier client-server architecture.*



Gambar 2.3 Konsep *three-tier*.

Arsitektur *3-tier* memiliki komponen terdepan yang bertanggung jawab dalam penyediaan antarmuka ke pengguna, dan komponen paling belakang adalah *server* basisdata itu sendiri. Komponen *middle tier* memperbolehkan pengguna untuk berbagi dan mengontrol *business logic layer* dengan mengisolasi komponen-komponen yang ada didalam *middle tier* tersebut dari aplikasi yang sesungguhnya dihadapi pengguna. Dengan *middle tier* tersebut maka *business logic* dan *rules* dieksekusi dan dapat mencakup hingga banyak *user*, lebih banyak daripada menggunakan *2-tier*, dengan menyediakan fungsi-fungsi untuk antrian, eksekusi aplikasi, maupun akses basisdata. Sistem pada *client* berinteraksi dengan *middle tier* melalui sebuah protokol yang sudah menjadi standar

seperti misalnya HTTP, RPC, ataupun *middleware* seperti CORBA, DCOM, atau JavaBean. Dan *middle tier* tersebut berinteraksi dengan *database server* melalui protokol *database* standar seperti SQL, ODBC, dan JDBC. *Middle tier* merupakan lapisan yang paling penting dan menentukan adanya komunikasi antara *client* dengan *server*. Pada *middle tier* ini segala fungsi-fungsi yang dibutuhkan untuk komunikasi antara *client* dan *server* muncul. Adanya istilah *fat server* maupun *fat client* merupakan pengertian bahwa fungsi-fungsi tersebut cenderung lebih banyak menempel pada *client* atau pada *server*.

Keuntungan dari arsitektur *3-tier* yaitu :

(1) Perubahan pada antarmuka pengguna atau *application logic* tidak saling mempengaruhi satu sama lain, membuat aplikasi tersebut mudah berevolusi untuk memenuhi kebutuhan baru.

(2) *Network bottleneck* diminimalkan karena *application layer* tidak mentransmisikan data ekstra ke *client*, hanya data yang diperlukan untuk menangani suatu *task*.

(3) Ketika *business logic* perlu diubah, hanya *server* yang perlu diubah. Dalam *two-tier architecture* setiap *client* perlu dimodifikasi.

(4) *Client* terisolasi dari basisdata dan operasi jaringan. *Client* tersebut dapat mengakses data dengan mudah dan cepat tanpa harus mengetahui lokasi penyimpanan data atau jumlah *server* yang ada dalam sistem.

(5) Koneksi basisdata dapat di-*pool* dan kemudian dibagi-pakai oleh beberapa pengguna, yang akan sangat mengurangi biaya sehubungan dengan *per-user licensing*.

(6) Organisasi memiliki basisdata yang independen, karena *data layer* ditulis menggunakan SQL standar yang memang bebas *platform*. Perusahaan tersebut tidak terikat pada *vendor-specific stored procedures*.

(7) *Application layer* ditulis dalam bahasa generasi ketiga atau keempat standar, yang telah biasa digunakan para pemrogram dari perusahaan tersebut.

2.3 *Web Service*

Web service menyediakan standar komunikasi antara berbagai aplikasi perangkat lunak yang berbeda, yang dapat dijalankan di berbagai platform dan framework. Konsorsium W3C mendefinisikan: *"A web service is a software system designed to support interable machine-to-machine interaction over a network.* Dalam *web service* komunikasi

dilakukan antara *provider* dan *requester*. *Provider* adalah orang atau organisasi yang menyediakan fungsi layanan tertentu, sedangkan *requester* adalah orang atau organisasi yang menggunakan layanan yang disediakan *provider*.

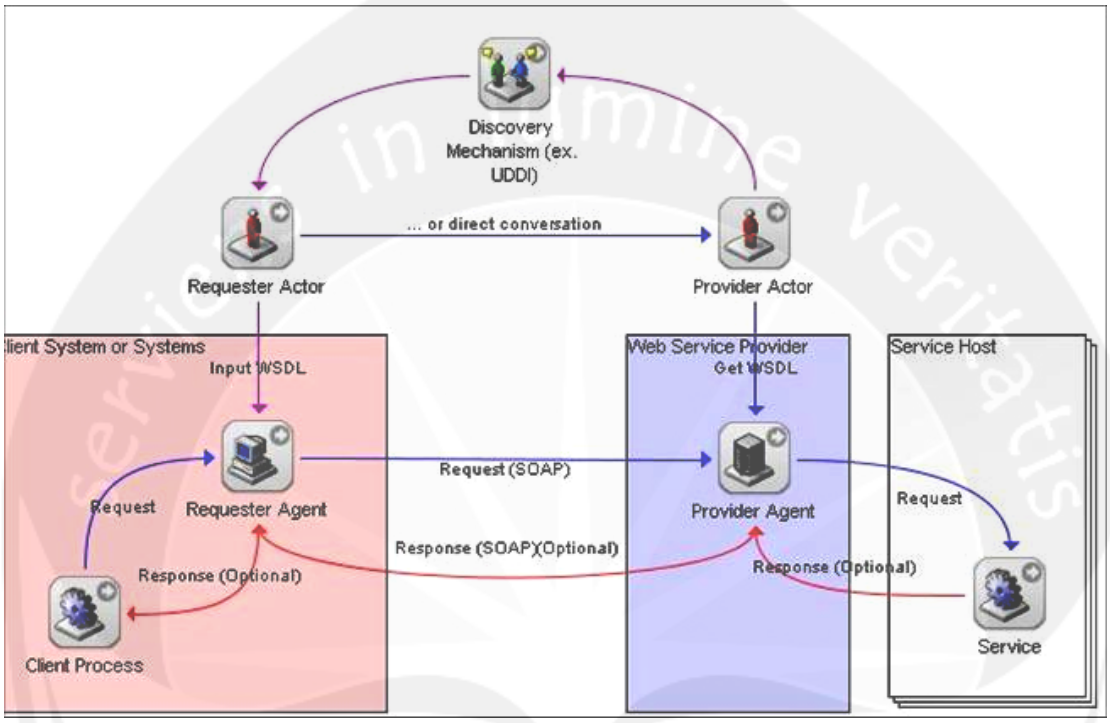
Web Services merupakan salah satu bentuk implementasi dari arsitektur model aplikasi *N-Tier* yang berorientasi layanan. Perbedaan *Web Services* dengan pendekatan *N-Tier* lainnya adalah dari segi infrastruktur dan dokumen yang digunakan sebagai format pertukaran data. Dalam implementasinya, *Web Services* tidak mempunyai tampilan, karena *Web* Services termasuk dalam Business- *Service* tier. Artinya didalam *Web Services* hanya tersedia fungsi-fungsi yang nantinya dapat digunakan oleh aplikasi lainnya*Web Services* menggunakan XML sebagai format dokumen dalam melakukan pertukaran datanya. Karena XML merupakan suatu format dokumen yang berbasis teks, maka *Web Services* memungkinkan berlangsungnya komunikasi antar aplikasi yang berbeda dengan platform yang berbeda pula. *Web Services* dapat diimplementasikan dalam berbagai jenis platform dengan menggunakan bahasa pemrograman apa pun, dan bisa digunakan oleh berbagai aplikasi yang menggunakan bahasa pemrograman apapun dengan platform apapun juga. Selama aplikasi tersebut dapat berkomunikasi dengan *Web Services* menggunakan protokol-protokol komunikasi. Termasuk HTTP, XML, SOAP, UDDI (Universal Description Discovery and Integration), dan WSDL (*Web* Services Description Language).

XML *Web Services* dapat digunakan secara internal oleh suatu aplikasi atau secara external. Digunakan secara external di sini maksudnya yaitu XML *Web Services* terdapat pada internet dan digunakan oleh berbagai macam aplikasi. XML *Web Services* mudah diakses melalui interface standar dan XML *Web Services* juga mengijinkan bermacam-macam sistem untuk bekerja bersama-sama dalam sebuah pekerjaan.*Web Services* memiliki blok bangunan sebagai berikut :

1. Menggunakan UDDI dan DISCO sebagai sarana pencarian *service*
2. Menggunakan WSDL dan XML Schema sebagai sarana untuk menjelaskan informasi yang memadai kepada *client* tentang spesifikasi *Web Services* itu sendiri. Biasanya disertai dengan contoh penggunaannya bila parameter yang diperlukan hanya berupa tipe data primitif.
3. Format pesan yang digunakan sebagai sarana komunikasi *Web Services* dengan *client* maupun *service* lainnya didefinisikan dengan SOAP
4. Pengkodean pesan yang digunakan *Web Services* adalah dalam format XML

5. Transportasi pesan menggunakan protokol standar, yaitu HTTP, SMTP, dan lain-lainnya.

Berikut gambaran sederhana konsep *Web Services* serta keterhubungan antara *Web Services* dengan aplikasi.

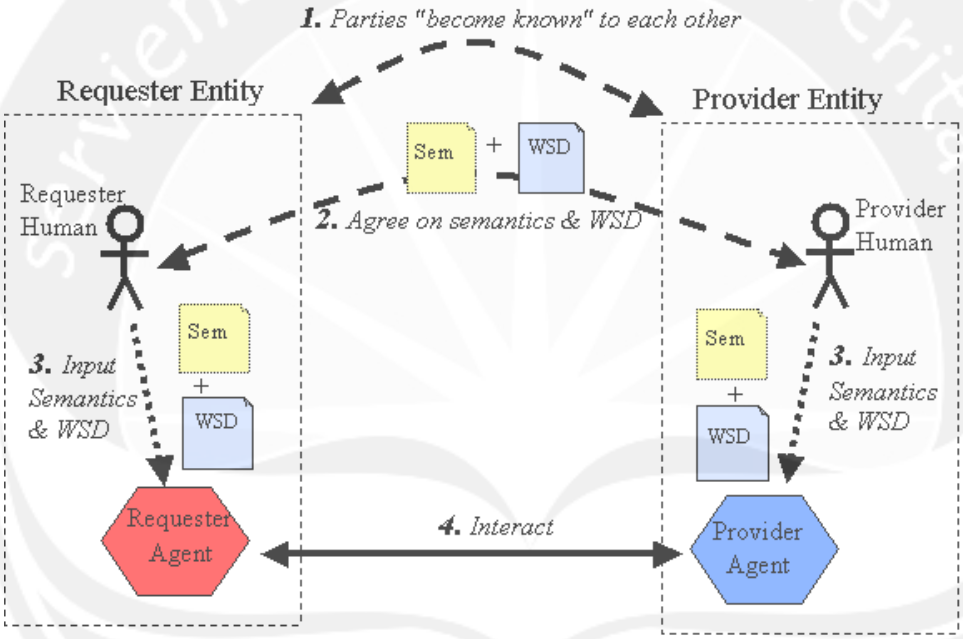


Gambar 2.4 Konsep *Web Service* [inter-locale.com].

WSDL merupakan bahasa standard yang menyediakan mekanisme untuk mendeskripsikan *Service* yang disediakan oleh sistem (*Web Service*), lokasi keberadaan *service* tersebut dan bagaimana cara memperolehnya, secara terstruktur dalam format XML. WSDL dapat dianalogikan sebagai IDL (interface definition language) dalam CORBA dan COM. *Service* dideskripsikan sebagai koleksi dari entry-point atau port komunikasi. WSDL mendeskripsikan *service* dengan menggunakan elemen sebagai berikut:

1. Type, yaitu tipe data yang digunakan sebagai argumen dan return type
2. Message, digunakan untuk merepresentasikan definisi data yang ditransmisikan.
3. Port type, merupakan Sekumpulan operasi yang didukung oleh satu atau lebih endpoint.

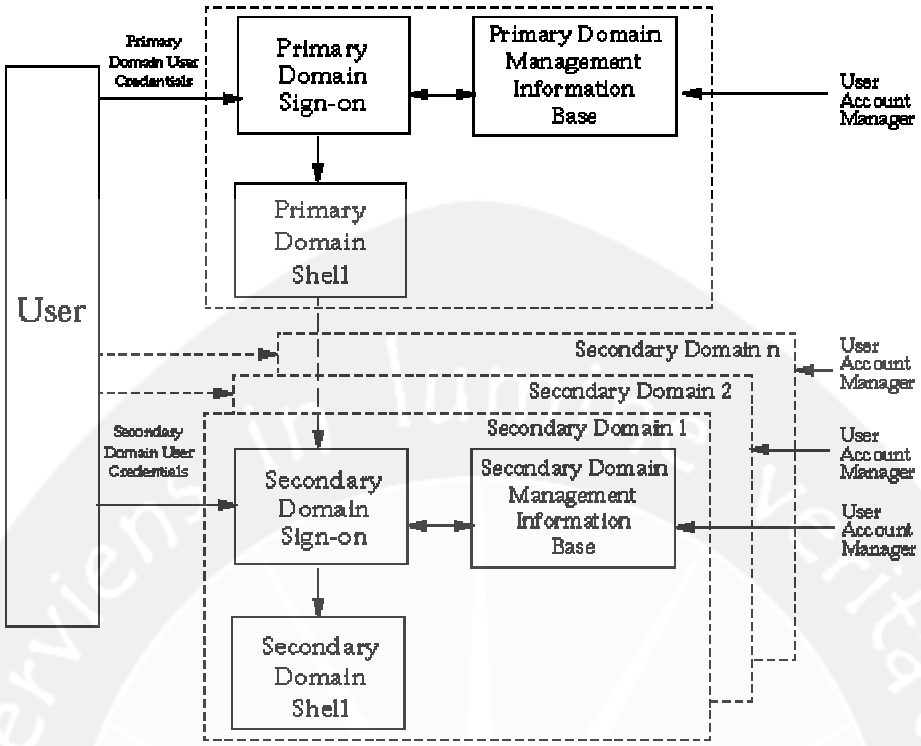
4. Binding, digunakan untuk mendefinisikan protokol dan format pertukaran data untuk operasi yang didefinisikan oleh Port type.
5. Port, digunakan untuk menspesifikasikan end-point yang digunakan untuk binding.
6. *Service*, merupakan koleksi endpoint yang berkaitan yang disediakan oleh *Web Service*.
7. Operation, digunakan untuk mendefinisikan kemampuan yang didukung oleh servis tertentu.



Gambar 2.4 WSDL Menyediakan Mekanisme *Web Service* [W3C].

2.4 *Single Sign-On*

Seiring berkembangnya tekologi informasi yang mendukung proses bisnis, pengguna dan administrator dihadapkan pada semakin banyaknya aplikasi yang digunakan dalam pekerjaan mereka. Pengguna biasanya harus *sign-on* beberapa kali, sebanyak aplikasi yang dibutuhkan, masing-masing mungkin dengan data login (*account*) yang berbeda. Administrator juga harus mengelola banyak *account* pada masing-masing aplikasi. Gambar 2.5 menunjukkan sistem lama dengan banyak *account* (Open Group, 2011).

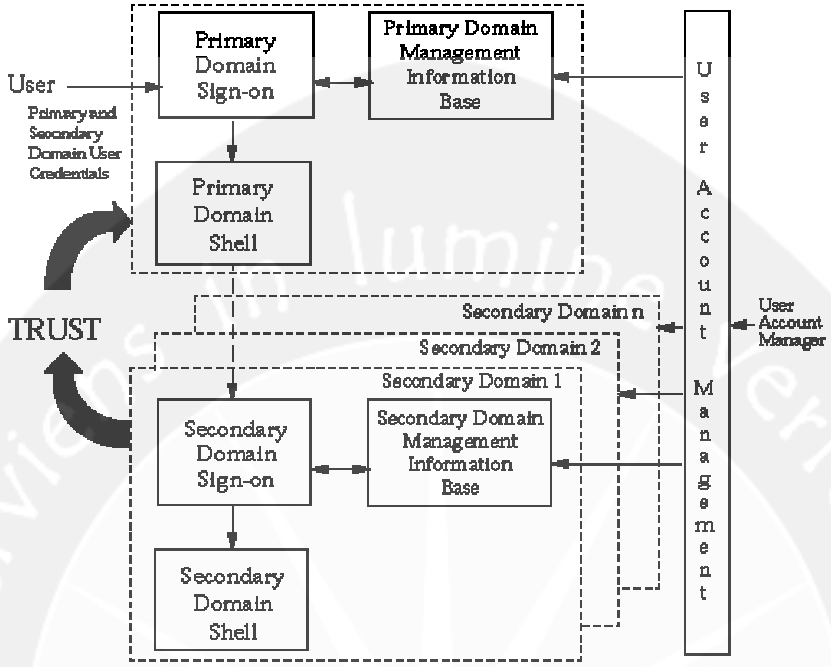


Gambar 2.5 Pendekatan Lama Sing-On pada Banyak Sistem

Dari perspektif manajemen, pendekatan lama ini mengharuskan manajemen independen untuk setiap domain dan penggunaan beberapa antarmuka manajemen *account*. Pertimbangan kegunaan dan keamanan menimbulkan kebutuhan untuk mengkoordinasikan, dan jika memungkinkan, mengintegrasikan fungsi *sign-on* pengguna dan fungsi manajemen *account* pengguna karena banyaknya domain yang berbeda dalam perusahaan. A *service* that provides such co-ordination and integration can provide real cost benefits to an enterprise through: Sebuah layanan yang menyediakan koordinasi dan integrasi seperti itu dapat memberikan manfaat biaya bagi perusahaan melalui:

- pengurangan waktu yang diambil oleh pengguna dalam operasi *sign-on* untuk masing-masing domain, termasuk mengurangi kemungkinan gagalnya operasi *sign-on*.
- keamanan ditingkatkan melalui pengurangan kebutuhan pengguna untuk menangani dan mengingat beberapa set informasi otentikasi.
- pengurangan waktu yang diambil, dan peningkatan respon, oleh administrator sistem dalam menambahkan/menghapus pengguna atau memodifikasi hak akses mereka.

- meningkatkan keamanan melalui peningkatan kemampuan administrator sistem untuk menjaga integritas konfigurasi *user ccount*



2.6 *Single Sign-On* Untuk Banyak Layanan

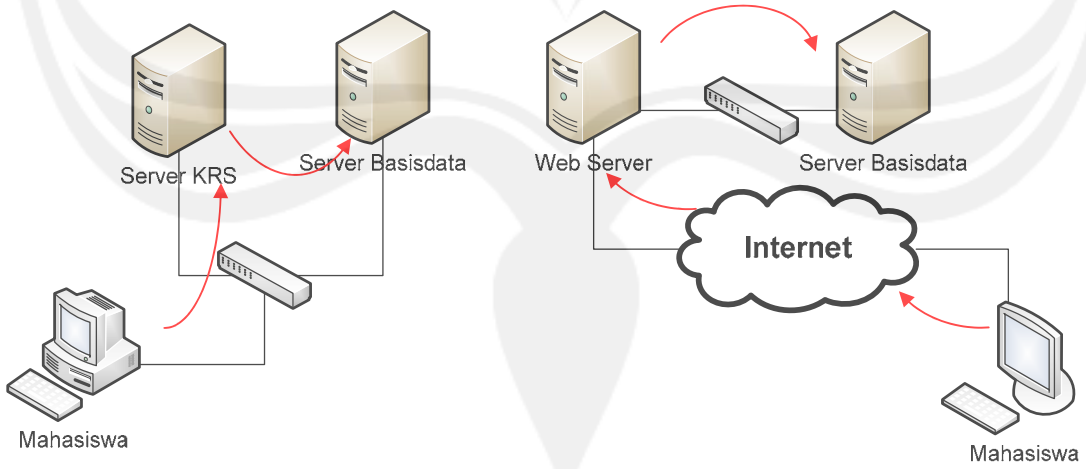
Dalam single sign-on, pendekatan sistem diperlukan untuk mengumpulkan informasi dan identifikasi yang diperlukan untuk mendukung otentikasi dari pengguna untuk setiap domain sekunder. Informasi yang diberikan oleh pengguna ini kemudian digunakan oleh layanan *Single Sign-On* dalam domain utama untuk mendukung otentikasi dari pengguna untuk setiap domain sekunder dimana pengguna berinteraksi.

BAB 3. ANALISA DAN PERANCANGAN SISTEM

Bab ini berisi analisis dan perancangan kebutuhan *single-account* serta spesifikasi kebutuhan dan perancangan perangkat lunak untuk mengembangkan sistem informasi yang ada di Universitas Atma Jaya Yogyakarta, yaitu SIATMA (khususnya aplikasi pengisian KRS secara *online*, sebagai contoh aplikasi *desktop*) dan SIAMA (Sistem Informasi Akademik Mahasiswa, sebagai contoh aplikasi *web*).

3.1 Analisis Kebutuhan *Single-Account*

SIATMA merupakan perangkat lunak *desktop* yang dikembangkan untuk menangani proses layanan akademik yang dilakukan Universitas Atma Jaya Yogyakarta. Sistem informasi ini menangani proses penawaran kelas kuliah, pengisian KRS, presensi, kelola nilai, dan lain-lain. SIAMA merupakan perangkat lunak berbasis *web* yang dikembangkan untuk membantu proses perolehan informasi akademik mahasiswa selama berkuliah di Universitas Atma Jaya Yogyakarta. Sistem ini dapat menampilkan jadwal pribadi mahasiswa, menampilkan Kartu Hasil Studi mahasiswa, menampilkan transkrip mahasiswa, menampilkan persensi mahasiswa untuk matakuliah yang diambil, serta dapat menampilkan batasan-batasan dalam mengambil matakuliah tertentu seperti Tugas Akhir, Kuliah Kerja Nyata, maupun kerja praktek.



Gambar 3.1 SIATMA dan SIAMA mengakses basisdata yang berbeda

Gambar 3.1 menunjukkan bahwa kedua sistem tidak saling berhubungan, dengan kata lain, data *account* yang dimiliki juga berbeda. Pemisahan basis data ini sengaja

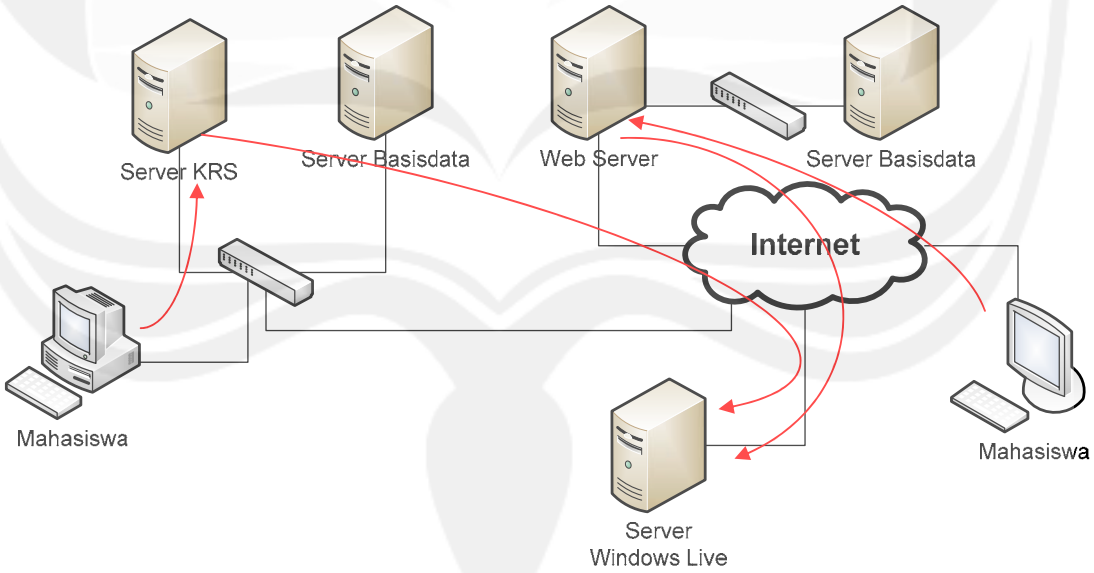
dilakukan karena besarnya kemungkinan terjadinya serangan terhadap data mahasiswa melalui Internet. Pemisahan kedua basisdata itu membuat pengembang aplikasi membuat *user account* yang berbeda, sehingga mahasiswa harus mengingat banyak *account*. Hal ini yang menyulitan mahasiswa, sehingga muncul kebutuhan untuk hanya mengingat satu *account* saja untuk login ke berbagai aplikasi.

3.2 Perancangan Kebutuhan *Single-Account*

Analisis kebutuhan di atas menunjukkan bahwa terdapat kebutuhan untuk membuat *single-account* tetapi tidak melalui integrasi basisdata (basisdata tetap terpisah). Ada dua pilihan *account* utama, yaitu:

1. Memilih *account* yang tersimpan di salah satu basisdata, lalu dibuatkan layanan untuk melakukan validasi login.
2. Menggunakan *account* lain di luar kedua sistem informasi (basisdata) tersebut.

Seperti telah diungkapkan sebelumnya, bahwa saat ini Universitas Atma Jaya Yogyakarta bekerjasama dengan Microsoft untuk memberikan layanan email, blog, dan media penyimpan virtual, dalam paket live@edu, yang ditujukan bagi seluruh sivitas akademik.



Gambar 3.2 SIATMA dan SIAMA mengakses basisdata yang sama

Ini berarti, setiap sivitas akademik (termasuk mahasiswa) memiliki *account* di Windows Live yang digunakan untuk mengakses layanan dari Microsoft tersebut. Windows Live sendiri telah menyediakan layanan berupa *web service* untuk melakukan validasi login.

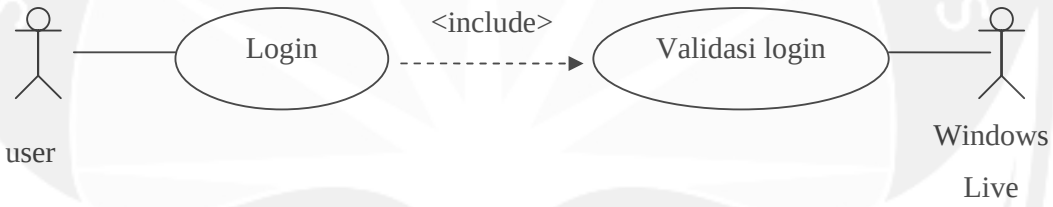
Oleh karena itu, *account* Windows Live ini yang dipilih untuk dijadikan *account* utama, sehingga aplikasi-aplikasi tersebut akan mengakses ke server Windoes Live,seperti terlihat pada Gambar 3.2.

3.3 Spesifikasi Kebutuhan Fungsional

3.3.1 *Use Case Diagram*

Berdasarkan analisa, kebutuhan fungsionalitas dari *single-account login* ini ditunjukkan dengan diagram *use case* pada gambar 3.3 yang rinciannya akan dijelaskan berikut ini:

Use case ini sama seperti *use case* login aplikasi biasa, hanya saja ditambahkan *use case* untuk megakses layanan *web service*.



Gambar 3.3. Diagram *Use Case Single-Account Login*

3.3.2 Spesifikasi *Use Case*

Berikut akan diuraikan lebih detil mengenai spesifikasi dari *use case* yang ada:

a. Use Case Spesification : Login**.**

1. Brief Description

Use Case ini digunakan oleh aktor untuk memperoleh akses ke dalam sistem. Login didasarkan pada sebuah id unik yaitu NIM mahasiswa dan password yang berupa rangkaian karakter atau gabungan tempat dengan tanggal lahir mahasiswa.

2. Primary Aktor

Mahasiswa.

3. Supporting Aktor

None

4. Basic Flow

6. Use case ini dimulai ketika aktor akan masuk ke dalam sistem dan harus login terlebih dahulu.
7. Aktor memasukkan username dan password.
8. Sistem akan memeriksa konelsi ke layanan Windows Live.

   E-1 Koneksi atau layanan tidak ada.
9. Sistem akan mengakses layanan *authentication* dari Windows Live.

   E-2 Username atau password yang diinputkan oleh aktor salah.
10. Sistem akan memberikan konfirmasi login.
11. Use case ini selesai dilakukan.

5. Alternative Flow

Aktor membatalkan login

6. Error Flow

E-1 Koneksi atau layanan tidak ada.

E-2 *Password* atau *username* tidak sesuai.

b. Use Case Spesification : Validasi Login**.**

1. Brief Description

Use Case ini digunakan untuk melakukan validasi login melalui *web service*.

2. Primary Aktor

Windows Live

3. Supporting Aktor

Mahasiswa

4. Basic Flow

1. Use case ini dimulai ketika menerima *request* untuk melakukan validasi serta menerima data berupa *username* dan *password*.
2. Memeriksa validasi login dari data *account* yang diterima.

   E-1 *Username* atau *password* yang diterima tidak valid.
3. Sistem akan memberikan hasil validasi ke *client* yang meminta.
4. *Use case* ini selesai dilakukan.

5. Alternative Flow

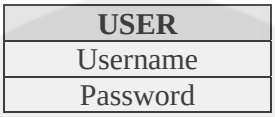
Mengirim konfirmasi bahwa *web service* aktif

6. Error Flow

E-1 *Password* atau *username* tidak sesuai.

3.4 Spesifikasi kebutuhan Data

Penelitian ini hanya fokus pada proses login, oleh karena itu kebutuhan data yang diperlukan hanyalah data *account* sederhana, yaitu *username* dan *password*.



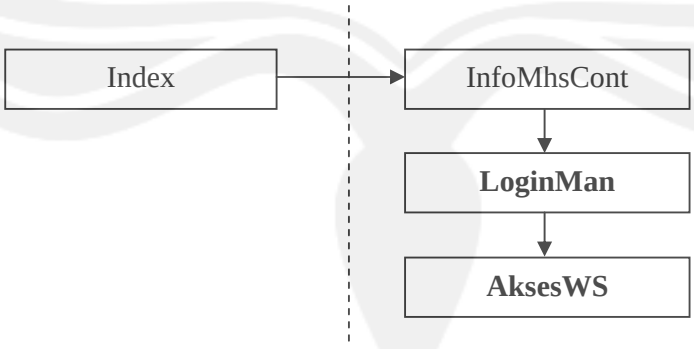
Gambar 3.4. *Entity Relationship Diagram (ERD)*

3.5 Perancangan Fungsional

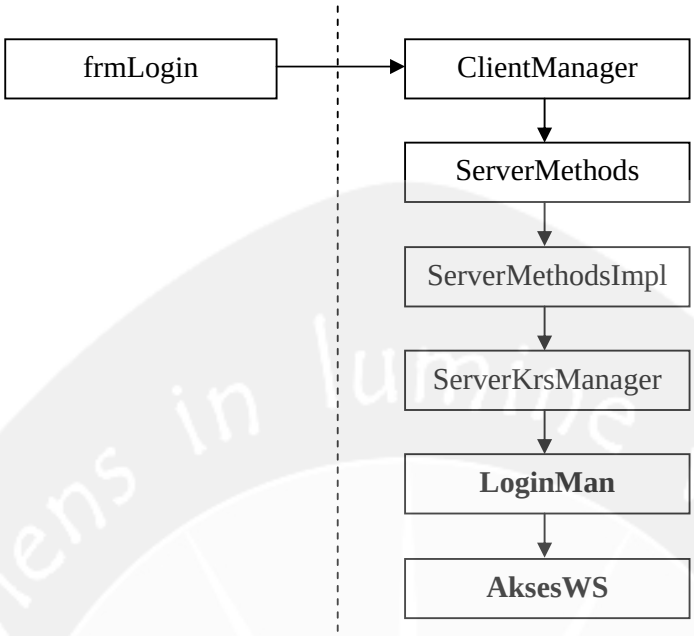
Fungsionalitas-fungsionalitas yang dideskripsikan dalam bentuk use case pada bab sebelumnya selanjutnya dijadikan dasar dalam perancangan sistem.

3.5.1 Perancangan Arsitektur

Perancangan arsitektur perangkat lunak ini melibatkan beberapa kelas yang ada dalam aplikasi yang akan menggunakan *single-account* ini, yaitu SIATMA, seperti terlihat di Gambar 3.5, dan SIAMA, terlihat di Gambar 3.6. Kelas LoginMan dan AksesWS merupakan kelas baru yang dibuat dalam penelitian ini.



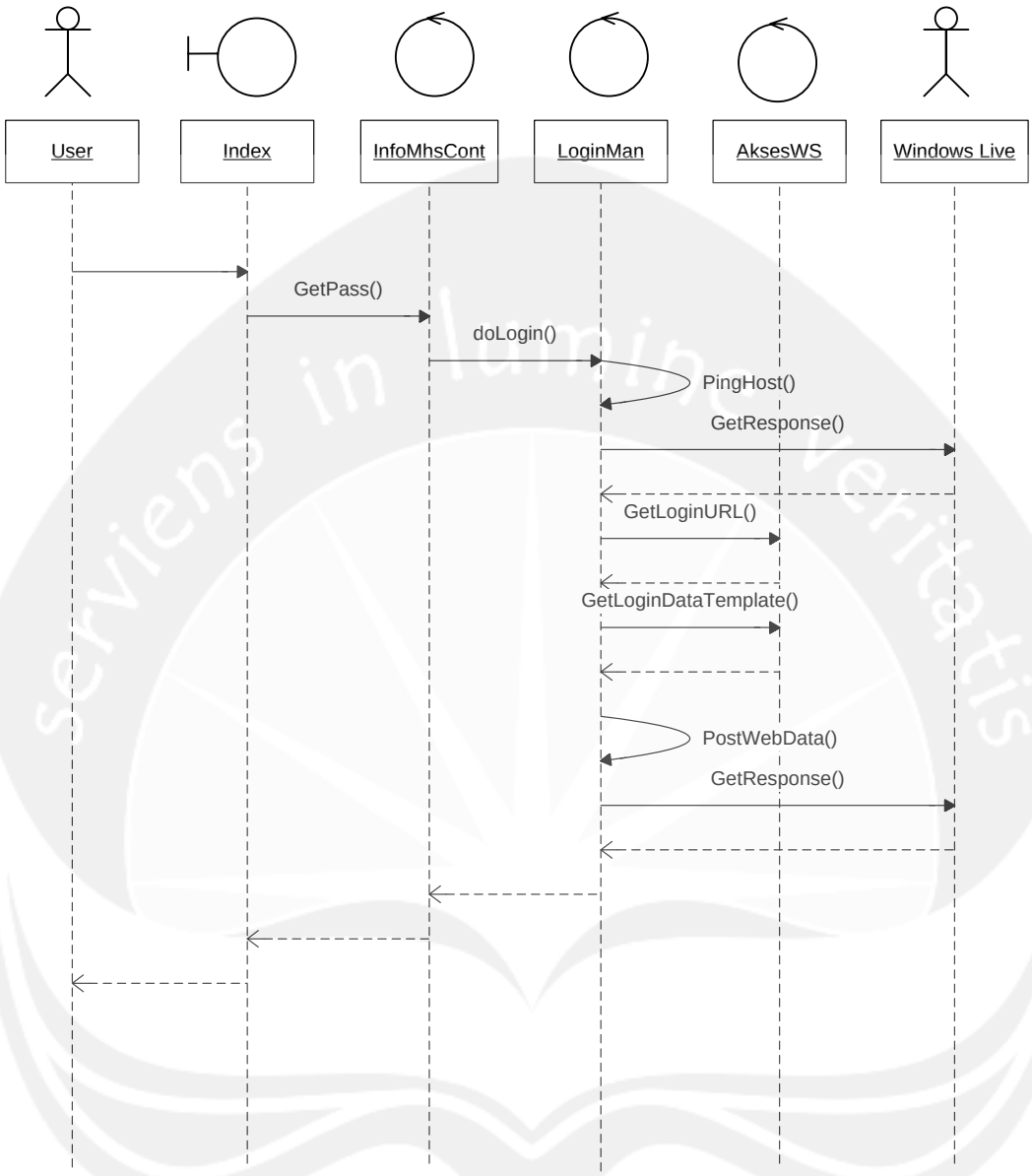
Gambar 3.5. Arsitektur Perangkat Lunak Login SIAMA (*web*)



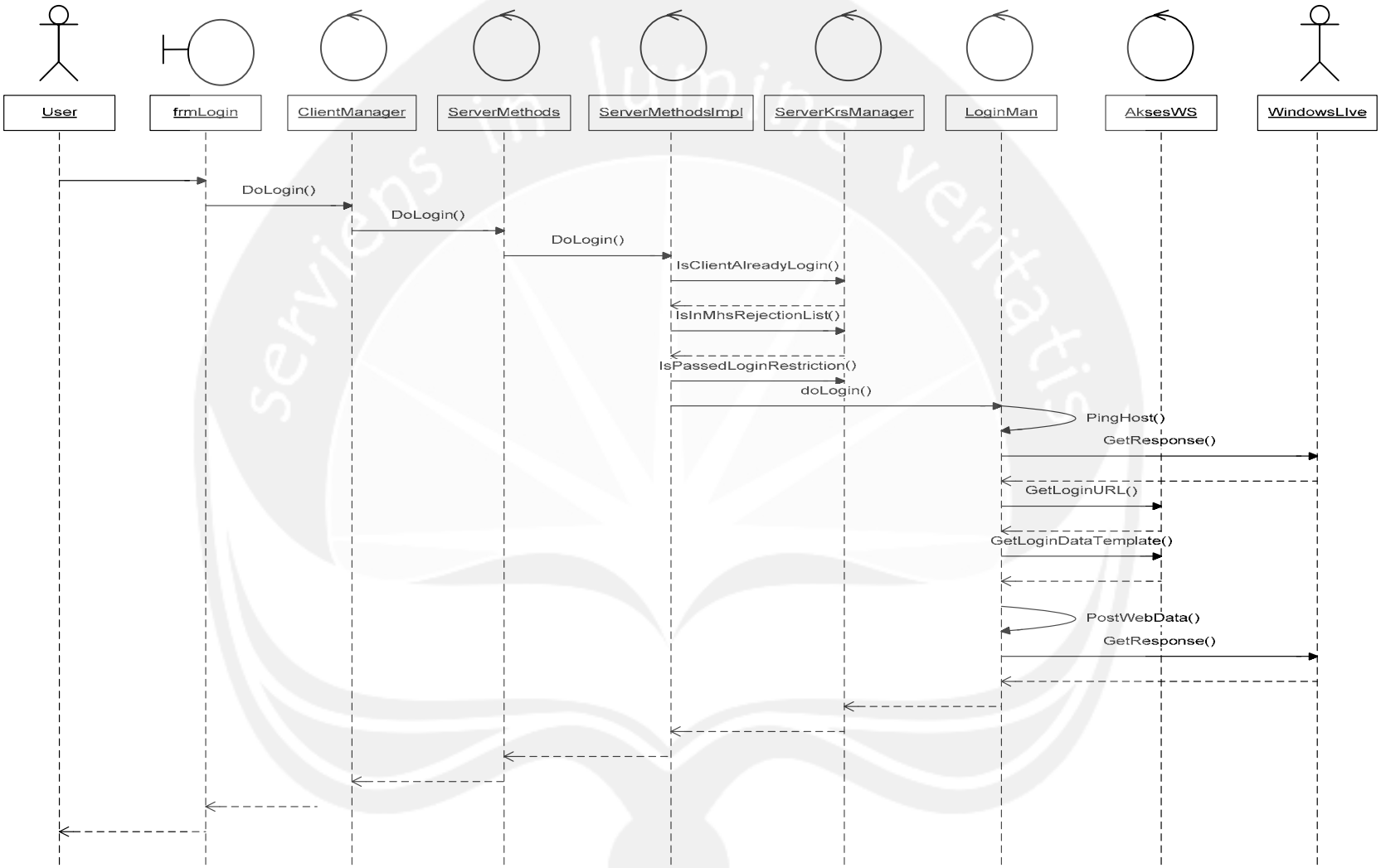
Gambar 3.6. Arsitektur Perangkat Lunak Login SIATMA (*desktop*)

3.5.2 Perancangan Rinci

Relasi antar kelas tersebut kemudian dapat didetilkan dengan memperlihatkan pemanggilan *method* kelas berdasarkan urutan waktu dalam bentuk *sequence diagram*. Gambar 3.7 menunjukkan urutan eksekusi *method* pada masing-masing kelas pada proses login SIAMA (*web*) dan Gambar 3.8 untuk SIATMA(*desktop*).

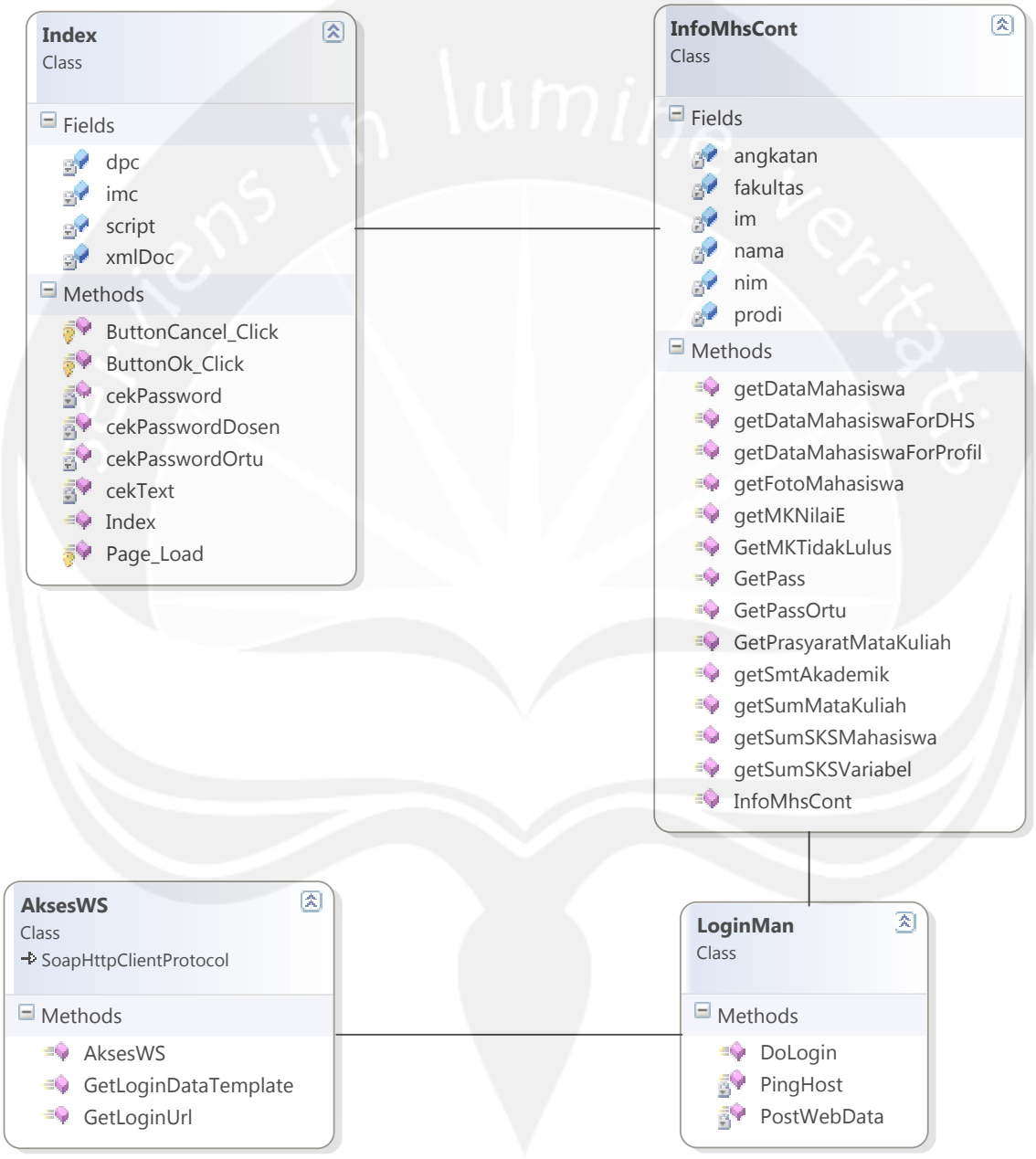


Gambar 3.7. *Sequence Diagram* Proses Login SIAMA (*web*)

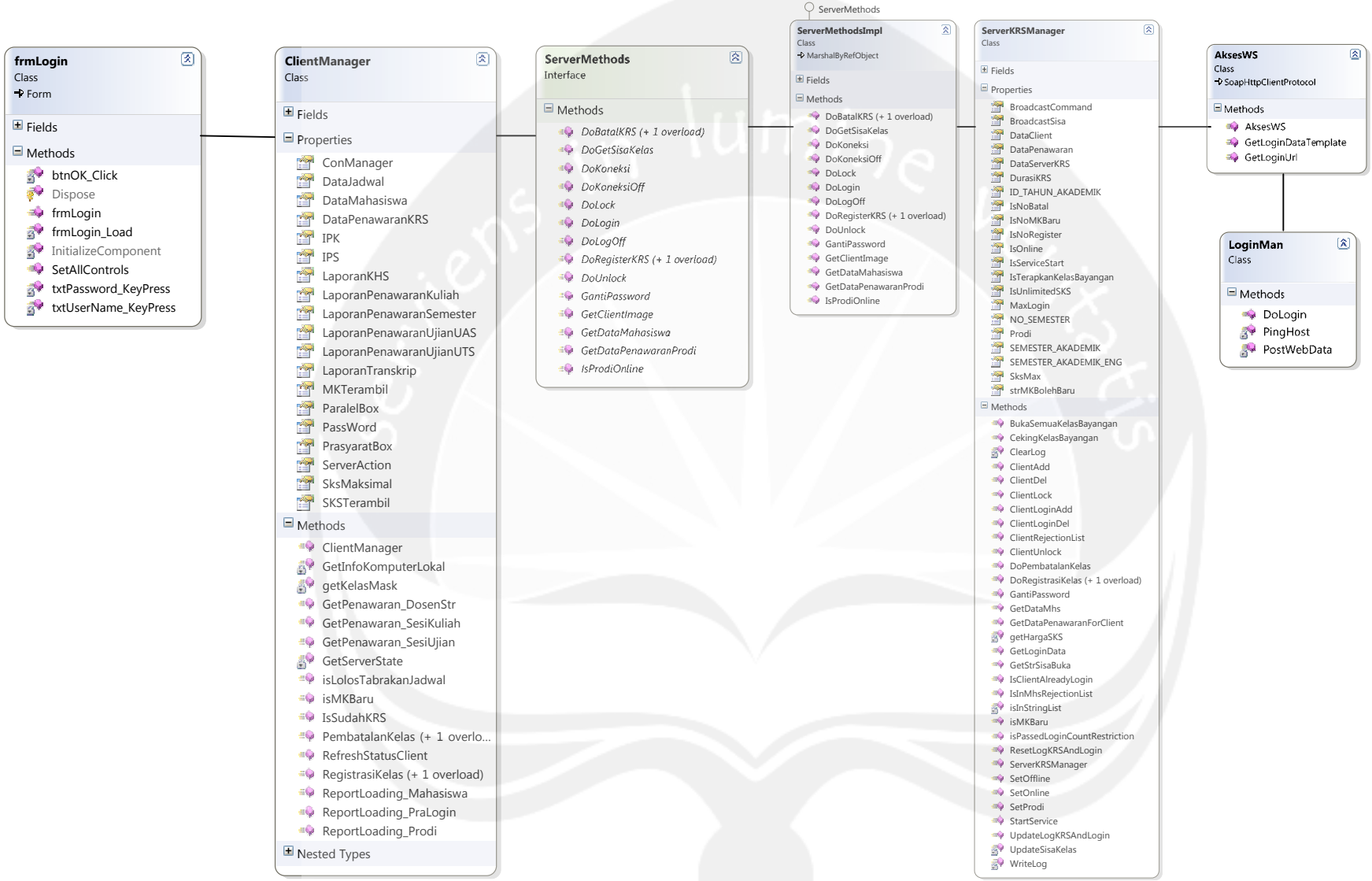


Gambar 3.8. *Sequence Diagram* Proses Login SIATMA (*desktop*)

Fungsionalitas-fungsionalitas yang dideskripsikan dalam bentuk *use case* pada bab sebelumnya, selanjutnya direalisasi dalam bentuk kelas-kelas yang mengimplementasikan fungsionalitas tersebut. Realisasi kelas-kelas dikelompokkan dalam dua bagian, yaitu untuk SIATMA dan SIAMA.



Gambar 3.9. *Class Diagram* Proses Login SIAMA (*web*)



Gambar 3.9. *Class Diagram* Proses Login SIATMA (*desktop*)

BAB 4 IMPLEMENTASI DAN PEMBAHASAN

4.1 Hasil Implementasi

Bab ini menjelaskan hasil implementasi dari penggunaan *single-account* pada aplikasi SIATMA, sebagai contoh aplikasi *desktop*, serta SIAMA, sebagai contoh aplikasi *web*. Berdasarkan analisis dan perancangan yang dilakukan, *account* Windows Live dari pengguna, dalam hal ini mahasiswa, digunakan sebagai *account* bagi kedua aplikasi tersebut.

Gambar 4.1 menampilkan kode program dari kelas AksesWS. Dalam kelas ini terdapat fungsi-fingsi untuk memberi Url *web service* untuk login dan template data login yang sesuai dengan format dari Windows live.

```
using System;

namespace LoginLiveKoe
{
    [System.CodeDom.Compiler.GeneratedCodeAttribute("wsdl", "2.0.50727.42")]
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Web.Services.WebServiceBindingAttribute(Name = "ManageDomain2Soap", Namespace =
"http://domains.live.com/Service/ManageDomain/V1.0")]

    class AksesWS : System.Web.Services.Protocols.SoapHttpClientProtocol
    {
        public AksesWS()
        {
            this.Url = "https://domains-dev.live-int.com/service/managedomain2.asmx";
        }

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://domains.live.com/Service/ManageDomain/V1.0/GetLoginUrl
", RequestNamespace = "http://domains.live.com/Service/ManageDomain/V1.0", ResponseNamespace =
"http://domains.live.com/Service/ManageDomain/V1.0", Use = System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle = System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        public string GetLoginUrl(string memberNameIn)
        {
            object[] results = this.Invoke("GetLoginUrl", new object[] {
                    memberNameIn});
            return ((string)(results[0]));
        }

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://domains.live.com/Service/ManageDomain/V1.0/GetLoginDa
taTemplate", RequestNamespace = "http://domains.live.com/Service/ManageDomain/V1.0", ResponseNamespace =
"http://domains.live.com/Service/ManageDomain/V1.0", Use = System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle = System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        public string GetLoginDataTemplate()
        {
            object[] results = this.Invoke("GetLoginDataTemplate", new object[0]);
            return ((string)(results[0]));
        }
    }
}
```

Gambar 4.1 Kode program dari Kelas AksesWS

Gambar 4.2 menampilkan potongan kode program dari kelas LoginMan yang juga harus ditambahkan pada kedua aplikasi. Kelas ini berisi fungsi-fungsi yang digunakan untuk melakukan pengecekan apakah *web service* dapat diakses, meminta *request* untuk melakukan pengecekan terhadap data login yang dikirimkan, serta mengembalikan hasilnya ke *client*.

```
public string DoLogin(string nama, string password)
{
    string server = "https://domains.live.com/service/managedomain2.asmx";
    string hasil = "";

   //Cek keberadaan Web Sercive
    if (PingHost(server) != "Service OK")
    {
        hasil = "Gagal terhubung ke Web Service!";
    }
    else
    {
        // buat SOAP proxy ke server
        AksesWS aksesWS = new AksesWS();
        aksesWS.Url = server;

        try
        {
            // Ambil login Url dan login data template
            string loginUrl = aksesWS.GetLoginUrl(nama);
            string loginDataTemplate = aksesWS.GetLoginDataTemplate();

            // Replace %NAME% dan %PASSWORD%
            string loginData = loginDataTemplate.Replace("%NAME%", nama);
            loginData = loginData.Replace("%PASSWORD%", password);

            // Post login data ke login Urluntuk dapat login ticket
            string loginTicket = PostWebData(loginUrl, loginData);

            if (loginTicket != "gagal")
                hasil = "Login Sukses";
            else
                hasil = "Gagal login!";
        }
        catch
        {
            hasil = "Domain tidak diketemukan!";
        }
    }
    return hasil;
}

private string PingHost(string args)
{
    HttpWebResponse res = null;

    try
    {
        //cinptakan request
        HttpWebRequest req = (HttpWebRequest)WebRequest.Create(args);
        req.Credentials = CredentialCache.DefaultNetworkCredentials;
        // Get response
        res = (HttpWebResponse)req.GetResponse();

        return "Service OK";
    }
    catch (Exception e)
    {
        return "Web Service tidak ketemu";
    }
}
```

Gambar 4.2.a Potongan Kode Program dari Kelas LoginMan

```
private static string PostWebData(string uri, string body)
{
    HttpWebRequest request = null;
    HttpWebResponse response = null;
    ASCIIEncoding encoding = new ASCIIEncoding();
    byte[] byte1 = encoding.GetBytes(body);
    string result;

    // Setup  request
    request = (HttpWebRequest)WebRequest.Create(uri);
    request.Method = "POST";
    request.ContentLength = body.Length;
    request.UserAgent = "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.1.4322)";

    Stream newStream = request.GetRequestStream();
    request.AllowAutoRedirect = true;
    newStream.Write(byte1, 0, byte1.Length);

    // Get response
    try
    {
        response = (HttpWebResponse)request.GetResponse();

        result = new StreamReader(response.GetResponseStream()).ReadToEnd();
    }
    catch
    {
        result = "gagal";
    }

    newStream.Close();

    return result;
}
```

Gambar 4.2.b Potongan Kode Program dari Kelas LoginMan

Setelah kedua kelas (AksesWS dan LoginMan) ditambahkan pada project masing-masing aplikasi, SIATMA dan SIAMA, kemudian ada kode program yang harus dimodifikasi pada kedua aplikasi tersebut. Dalam melakukan modifikasi program ini perlu dipertimbangkan lokasi modifikasi kode program untuk login.

Aplikasi *desktop* SIATMA terdiri dari aplikasi *client* dan *server*. Pemodifikasian kode program dilakukan di aplikasi *server*, tepat pada kode validasi login yang terletak di fungsi DoLogin() pada kelas ServerMethodsImpl. Aplikasi SIAMA, karena berbasis *web*, maka seluruh kode program terletak di *server*. Pemodifikasi kode program dilakukan pada fungsi GetPass() di kelas InfoMhsCont. Fungsi GetPass() hasil modifikasi pada aplikasi *web* SIAMA terlihat pada gambar 4.3. Sedangkan fungsi DoLogin(hasil modifikasi pada aplikasi *desktop* SIATMA terlihat pada gambar 4.4.

```
public string GetPass(string npm)
{
      LoginMan loginMan = new LoginMan();
      return loginMan.DoLogin(nama, password, server);
}
```

Gambar 4.3 Fungsi GetPass() Hasil Modifikasi di SIAMA

```
public ServerType.LoginResult DoLogin(string MAC, string username, string password, string namakomputer, string ipkomputer)
{
   try
   {
      if (dr != null)
      {
         if(Program.KrsManager.IsClientAlreadyLogin(username))
         {
            return ServerType.LoginResult.Error_TolakMhs;
         }

         if (Program.KrsManager.IsInMhsRejectionList(username))
         {
            return ServerType.LoginResult.Error_TolakMhs ;
         }

         if (!Program.KrsManager.isPassedLoginCountRestriction(username))
         {
            return ServerType.LoginResult.Error_LoginCountRestriction;
         }

         LoginMan loginMan = new LoginMan();
         string hasil = loginMan.DoLogin(nama, password, server);
         if (hasil == "Login Sukses")
         {
            Program.KrsManager.ClientLoginAdd(MAC, username, dr.NAMA_MHS, namakomputer, ipkomputer);
            return ServerType.LoginResult.Sukses;
         }
         else
            return ServerType.LoginResult.Error_Username;
      }
      else
      {
         return ServerType.LoginResult.Error_Username;
      }
   }
   catch(Exception ex)
   {
      return ServerType.LoginResult.Error_Unknown;
   }
}
```

Gambar 4.4 Fungsi DoLogin() Hasil Modifikasi di SIATMA

4.2 Pembahasan

Secara umum penerapan *single-account* dengan menggunakan *account* dari Windows Live ini berjalan dengan baik. Seorang mahasiswa bisa login ke aplikasi SIATMA maupun SIAMA dengan *login account* yang sama. Tentu saja *account* tersebut juga bisa digunakan mahasiswa untuk mengakses layanan Live@edu. Penggunaan *single-account* ini sangat membantu mahasiswa untuk melakukan login di berbagai layanan

aplikasi, karena hanya perlu mengingat satu *login account*. Pengguna, dalam hal ini mahasiswa, tidak perlu belajar menggunakan aplikasi lagi meskipun proses login sudah berubah, karena tidak ada perubahan antarmuka pengguna. Gambar 4.5 dan 4.6 memperlihatkan antarmuka pengguna untuk login dari kedua aplikasi.

Penggunaan *account* Windows Live sebagai *account* tunggal juga memiliki beberapa kekurangan. Karena layanan ini tersedia di Internet, maka penggunaan aplikasi SIATMA yang sebelumnya hanya berjalan di jaringan lokal, setelah penerapan *single-account* ini, kemudian menuntut adanya koneksi Internet. Selain itu,jika suatu saat Microsoft mengubah kebijakannya mengenai Windows Live ini, maka akan berpengaruh ke aplikasi tersebut.

Penelitian ini belum memecahkan masalah jika suatu saat koneksi Internet terputus atau server layanan bermasalah, padahal sedang dilakukan pengisian Kartu Rencana Studi secara *online*. Masalah tersebut bisa diatasi dengan membuat *account* sementara yang hanya berlaku pada rentang waktu tertentu. *Single-account* ini bisa dikembangkan lagi menjadi *single sign-on*, dimana pengguna hanya perlu login sekali untuk bisa mengakses berbagai layanan aplikasi.

Gambar 4.5 Antarmuka Login dari Aplikasi SIAMA

Gambar 4.6 Antarmuka Login dari Aplikasi SIATMA

BAB 5 KESIMPULAN

Penerapan sistem *single-account* untuk berbagai aplikasi, baik desktop maupun *web* ini telah berhasil dikembangkan dengan baik. Pemilihan *account* Windows Live lebih didasarkan pada fakta bahwa seluruh sivitas akademika Universitas Atma Jaya Yogyakarta memiliki Live Id. Otentikasi *account* Live Id dilakukan dengan mengakses layanan *web service* yang disediakan oleh pihak Microsoft. *Ticket login* yang diperoleh oleh aplikasi menunjukkan pengguna tersebut memang pemilik Live Id tersebut, tetapi aplikasi tetap harus mengatur *role* (hak akses) pengguna tersebut pada aplikasi yang diakses.

Pemodifikasian kode program dilakukan di aplikasi, tepat pada kode validasi login. Kode bisa disisipkan dalam bentuk kelas, yang kemudian diakses dari fungsi validasi login dari tiap aplikasi. Pengaksesan ke layanan otentikasi harus mengikuti aturan main dari pemilik layanan, misalnya urutan pengaksesan dan format datanya. Pada aplikasi *desktop* yang memiliki aplikasi *server* yang terpisah, pengaksesan layanan otentikasi dilakukan dari aplikasi *server* (bukan di *client*). Sedangkan pada aplikasi *web*, karena semua kode program ada di *server*, maka perubahan dilakukan pada fungsi validasi login.

Dengan penerapan *single-account* ini diharapkan, baik pengguna maupun administrator, semakin lebih mudah dalam menggunakan dan mengatur berbagai aplikasi yang harus diakses. Penggunaan *single-account* ini sangat membantu mahasiswa untuk melakukan login di berbagai layanan aplikasi, karena hanya perlu mengingat satu *login account*. Meskipun demikian, *single-acount* ini masih menuntut pengguna untuk login beberapa kali, di masing-masing aplikasi yang diakses. Karena itu, diharapkan integrasi *account* ini dikembangkan lagi menjadi *single sign-on*, dimana pengguna cukup login sekali untuk dapat mengakses berbagai aplikasi.

DAFTAR PUSTAKA

Chaffee, Alex , diakses 18/12/2010, "One, two, three, or n tiers? Should you hold back the tiers of your application?", http://www.javaworld.com/javaworld/jw-01-2000/jw-01-ssj-tiers.html.

CISCO, diakses 22/3/2010, "Cisco IOS Security Configuration Guide, Release 12.2: AAA Overview", http://www.cisco.com/en/US/docs/ios/12_2/security/configuration/guide/scfaaa.html#wp1000871.

Ittelkom, Admin, diakses 24/12/2010, "*Web Service*", http://www.ittelkom.ac.id/library/index.php?v%E2%80%A6%00%00, diakses 4/12/2010.

Open Group, The, diakses 4/12/2010, "Introduction to Single Sing-On", http://www.opengroup.org/security/sso/sso_intro.htm.

Ponnapalli, Ravikanth, 2005, "Secure implementation of Enterprise single sign-on product in an organization", SANS Institute InfoSec Reading Room.

Phillips, Addison P., diakses 4/12/2010, "Web Services and Internationalization", http://www.inter-locale.com/whitepaper/multilingual/ml73-ws-20050524.xml.

Rudy, diakses 24/3/2010, Riechie, Gunadi, O., "Integrasi Aplikasi Menggunakan Single Sign On Berbasiskan Lightweight Directory Access Protocol (LDAP) dalam Portal Binus@Access (Bee-Portal)", http://ict.binus.edu/file/research/jurnal-skripsi-odie-v-2.1-revis-renan-RECEIVED.pdf.

Sadoski, D., S. Comella-Dorda, diakses 18/12/2010, *Three Tier Software Architectures*, http://www.sei.cmu.edu.

Sari, R.F., Hidayat, S., 2006, "Integrasi Mekanisme Autentikasi Aplikasi *Web* Server dengan Metode LDAP: Studi Kasus Aplikasi SIPEG UI", Jurnal Teknologi Edisi Khusus No. 1.

Working Group, W3C, 2004, diakses 4/12/2010, "Web Services Architecture", http://www.w3.org/TR/ws-arch/.