2. TEORI PENUNJANG

Dalam mendesain dan membuat *security surveillance system* ini diperlukan beberapa teori dasar antara lain :

2.1. *WebCam*

Webcam adalah suatu kamera digital yang didesain khusus untuk untuk mengambil gambar sacara digital dan dapat mengirimkannya melalui *internet*. Ada dua cara untuk menghubungkan *webcam* dengan PC. Cara pertama adalah menghubungkan *webcam* dengan PC melalui USB. Solusi ini cukup murah dan paling mudah digunakan karena hanya membutuhkan sebuah *driver* yang di*install* di *Windows* OS yang otomatis akan menangani komunikasi USB dari *webcam* ke PC. Alternatif lain adalah menghubungkan sebuah *Video Camera* dengan *analog output* (*Composite video*) ke sebuah *Video Digitizing Card* di PC. Solusi ini jauh lebih mahal dibandingkan dengan solusi pertama tetapi akan menghasilkan kualitas yang lebih baik.

Pada saat *Windows98* masih populer API yang biasa digunakan untuk *webcam* dan peralatan *video source* yang lain adalah *Video for Windows* (VfW). *Program* yang menggunakan *Video for Windows* (VfW) salah satunya adalah NetMeeting. Akan tetapi sejak *Windows2000* dan *WindowsXP* populer maka *Video for Windows* (VfW) sudah jarang digunakan. Sebagai gantinya maka API yang digunakan pada *Windows2000* dan *WindowsXP* adalah *Windows Driver Model* (WDM). Beberapa *webcam* menyertakan *driver* yang mendukung *Video for Windows* (VfW) dan *Windows Driver Model* (WDM) akan tetapi banyak *webcam* terutama *webcam* baru hanya menyertakan *driver* untuk *Windows Driver Model* (WDM). API lainnya yang dapat digunakan ialah TWAIN API yang umumnya digunakan pada *scanner*, tetapi ada beberapa *webcam* yang mendukung API ini.

Ada tiga macam cara untuk meng*import* sebuah gambar ke dalam program yaitu :

1. Meng-*import* gambar secara langsung dengan menggunakan API *Video for Windows* (VfW) atau *Windows Driver Model* (WDM). Akan tetapi cara ini

agak rumit karena membutuhkan pengertian yang sangat mendalam tentang *Windows system programming*.

2. Menggunakan komponen OCX atau VCL untuk melakukan bagian yang sulit dikerjakan pada API *Video for Windows* (VfW) atau *Windows Driver Model* (WDM). Akan lebih mudah bila membiarkan *detail programming* yang sulit dikontrol oleh komponen VCL pada *delphi*. Contoh : *user* dapat melakukan *capture image* sekaligus merubahnya menjadi obyek T*image*.
3. Menggunakan *program* terpisah untuk meng-*capture* gambar dan menyimpannya pada *harddisk*. Lalu untuk meng*import* gambar ke *dalam program* cukup dengan membuka *file* gambar tersebut.

2.2. *Wireless Application Protocol* (WAP)

Hampir semua fasilitas *Web Services* tidak bisa digunakan secara langsung pada alat-alat komunikasi *mobile*. Hal ini disebabkan karena *web services* tersebut didesain untuk digunakan pada alat dengan layar yang lebih besar pada peralatan yang statis (tidak bergerak). Jadi *Wireless Application Protocol* (WAP) dibuat supaya peralatan komunikasi *mobile* dapat menggunakan fasilitas *Web Service.* WAP dikembangkan oleh Eropa dan merupakan saingan I-mode yang dibuat oleh Jepang. Dahulu WAP dan I-mode tidak *compatible* karena WAP dibuat berdasarkan format XML yang baru sedangkan I-mode dibuat berdasarkan format HTML yang lama. Akan tetapi kini telah diluncurkan WAP 2.X yang dapat *compatible* dengan *I-mode*.

Berbeda dengan arsitektur WAP 1.X, arsitektur WAP 2.X terdiri dari empat komponen konseptual yaitu : *Application environment*, *protocol framework*, *security service*, dan *service discovery*. Arsitektur WAP 2.X tidak mempunyai pembagian yang jelas antara *server*, *gateway*, dan *user terminal environment*. Selain itu WAP 2.X juga secara jelas membedakan antara :

- *Bearer* (CSD, GPRS, IMT-2000)
- *Transport* (WDP, TCP)
- *Session Layer* (cookies,CC/PP)
- *Applications*

JWAP adalah adalah implementasi dari *wireless application protocol* (WAP) dalam java. JWAP terdiri dari *Wireless Session Protocol* (WSP) dan *Wireless Transaction Protocol* (WTP). Untuk mengetahui status dari pesan maka digunakan tipe primitif (*primitive type*). Seperti yang terlihat pada Tabel 2.1. ada empat macam tipe primitif yaitu :

Tabel 2.1. Tabel Tipe Primitif

(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

TIPE	**SINGKATAN**	**DESKRIPSI**
Request	Req	Primitif yang berfungsi untuk me-*request* suatu *service*
Indication	Ind	Primitif yang berfungsi untuk menunjukkan adanya suatu aktivitas yang terjadi pada *peer* atau *provider*
Response	Res	Primitif yang digunakan untuk merespon suatu *indication* yang diterima.
Confirm	Cnf	Primitif yang berfungsi untuk menunjukkan *service* yang di-*request* telah selesai dikerjakan.

WSP bertujuan untuk mengatur operasi yang terjadi antara *client* dan *server*. Ada beberapa *service* yang disediakan antara lain :

- S_connect

Digunakan untuk mengadakan koneksi dengan WAP *gateway*. Parameter yang ada dalam s_connect dapat dilihat pada Tabel 2.2.

Tabel 2.2. Parameter s_connect

(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

	Primitive			
Parameter	**Req**	**Ind**	**Res**	**Cnf**
Server Address	M	M (=)	-	-
Client Address	M	M (=)	-	-
Client Headers	O	C (=)	-	-
Requested capabilities	O	M (=)	-	-
Server headers	-	-	O	C (=)
Negotiated capabilities	-	-	O	M (=)

Keterangan :

M : keberadaan parameter harus ada.

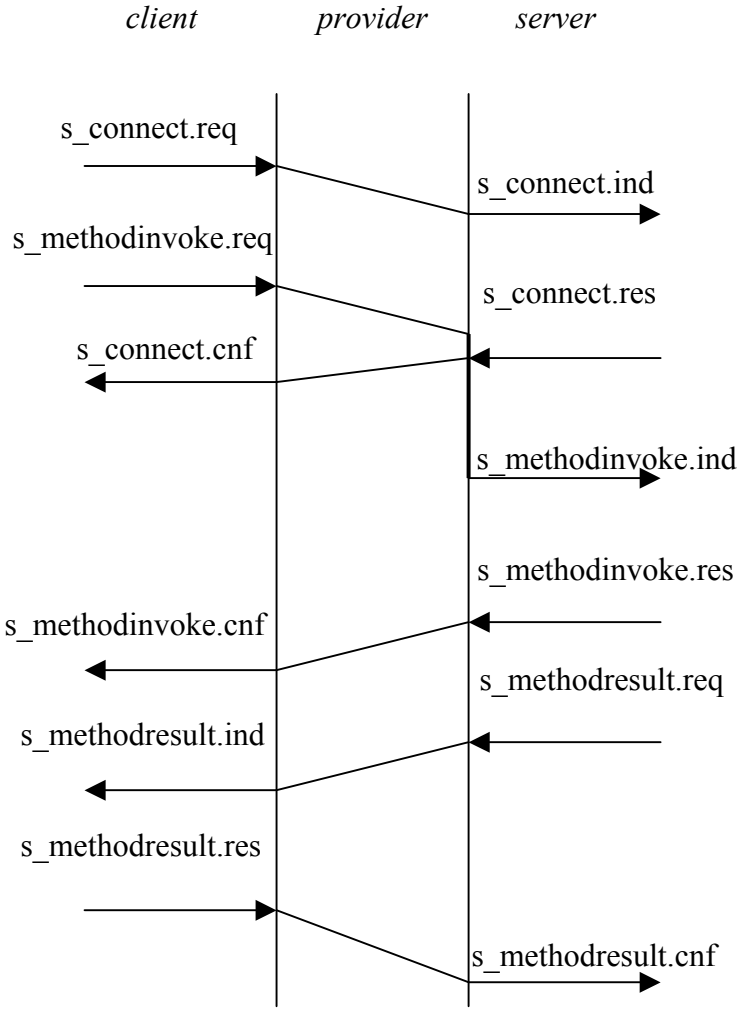
C : keberadaan parameter tergantung pada nilai dari parameter yang lain.

O : keberadaan parameter tergantung dari *client*.

P : keberadaan parameter tergantung dari *provider*.

(=) : nilai parameter identik dengan nilai parameter koresponden.

Seperti yang terlihat pada Gambar 2.1. untuk mengadakan koneksi dengan WAP gateway maka *client* akan mengirimkan primitif s_connect.req. Kemudian *client* juga mengirimkan primitif s_methodinvoke.req. Akan tetapi sebelum itu *server* akan mengirimkan respon berupa s_connect.res yang menunjukkan bahwa s_connect diterima. *Server* lalu mengirimkan s_method-invoke.res yng diterima oleh *client* sebagai s_methodinvoke.cnf. Setelah itu *server* baru mengirimkan s_methodresult.req yang direspon *client* dengan mengirimkan s_methodresult.res.



Gambar 2.1. urutan primitif koneksi ke WAP gateway
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

- S_disconnect

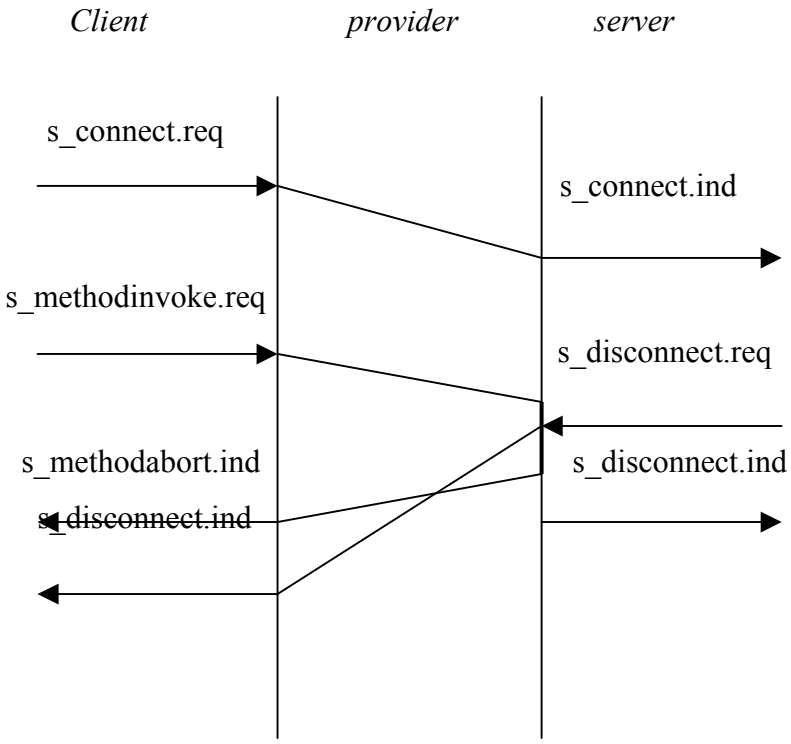
Digunakan untuk memutuskan koneksi dengan WAP *gateway*. Parameter yang ada dalam s_disconnect dapat dilihat pada Tabel 2.3.

Tabel 2.3. Parameter S_disconnect

(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

Parameter	**Primitive**	
	Req	**Ind**
Reason Code	M	M
Redirect Security	C	C (=)
Redirect Address	C	C (=)
Error Handler	O	P (=)
Error Body	O	P (=)

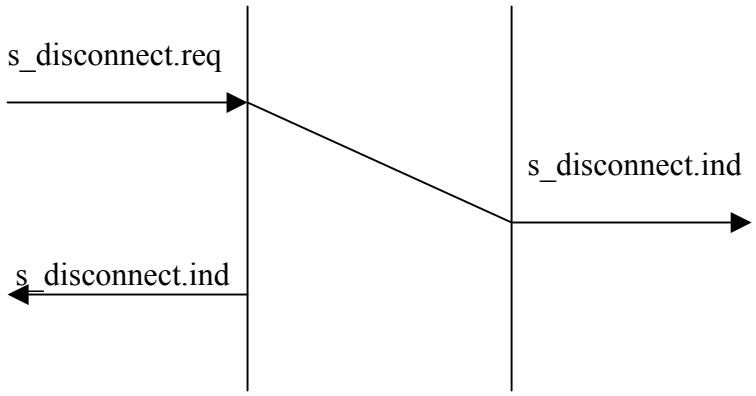
Seperti yang terlihat pada Gambar 2.2. jika koneksi ke WAP *gateway* ditolak maka setelah *client* mengirimkan s_connect.req dan s_methodinvoke.req maka *server* akan mengirimkan s_disconnect.req. *Client* lalu akan menerima s_methodabort.ind dan s_disconnect.ind yang menandakan bahwa hubungan ke WAP *gateway* ditolak.



Gambar 2.2. urutan primitif penolakan koneksi ke WAP gateway oleh *server*

(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

Selain ditolak oleh *server* proses *disconnect* bisa terjadi karena adanya proses *disconnect* setelah terjadi koneksi ke WAP *gateway*.proses *disconnect* bisa dilakukan oleh *client* maupun *server*. Seperti yang terlihat pada Gambar 2.3. jika *client* yang ingin melakukan proses *disconnect* maka *client* tinggal mengirimkan s_disconnect.req.



Gambar 2.3. urutan primitif pemutusan koneksi ke WAP gateway
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

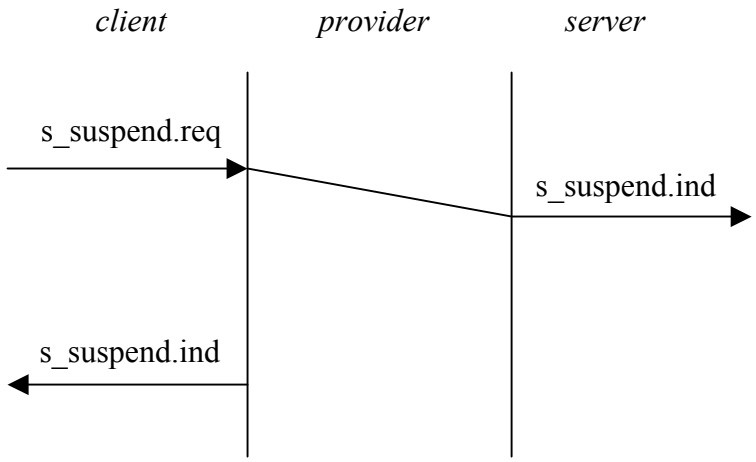
- S_suspend

Digunakan untuk menangguhkan *session* yang sedang dilakukan. Parameter yang ada dalam s_suspend dapat dilihat pada Tabel 2.4. berikut :

Tabel 2.4. Parameter S_suspend
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

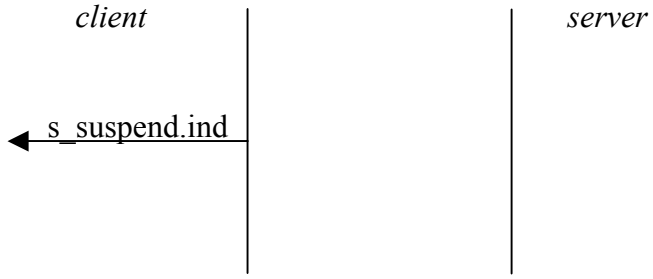
Parameter	**Primitive**	
	Req	**Ind**
Reason	-	M

Seperti yang terlihat pada Gambar 2.4. jika *client* ingin menangguhkan *session* maka *client* bisa mengirimkan s_suspend.req.



Gambar 2.4. urutan primitif penangguhan session oleh *client*
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

Selain penangguhan *session* yang terjadi atas kehendak *client* maka *server* juga dapat melakukan penangguhan *session* sewaktu-waktu. Seperti yang terlihat pada Gambar 2.5. jika terjadi penangguhan *session* yang dilakukan oleh *server* maka *server* langsung mengirimkan primitif s_suspend.ind sehingga tidak memerlukan s_suspend.req.



Gambar 2.5. urutan primitif penangguhan session oleh *server*
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

- S_resume

Digunakan untuk melanjutkan *session* yang sedang ditangguhkan. Parameter yang ada dalam s_resume dapat dilihat pada Tabel 2.5. berikut :

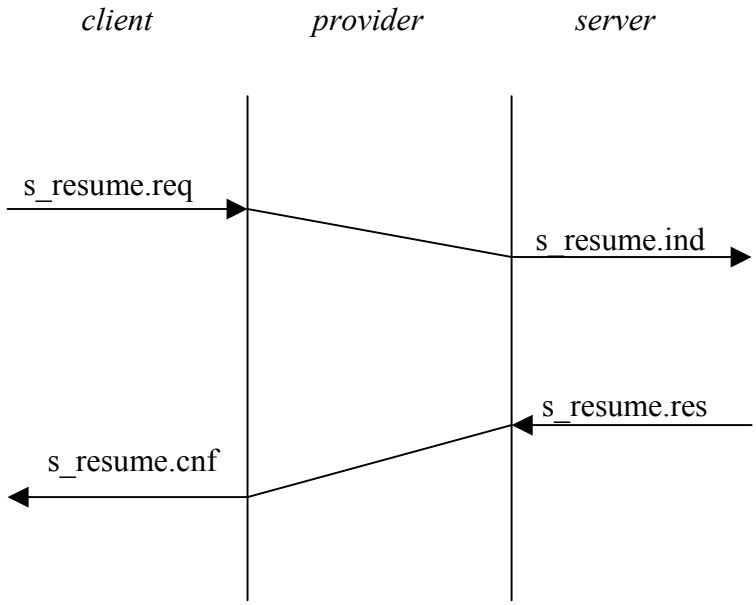
Tabel 2.5. Parameter S_resume
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

	Primitive			
Parameter	**Req**	**Ind**	**Res**	**Cnf**
Server Address	M	M (=)	-	-

Tabel 2.5. Parameter S_resume (lanjutan)

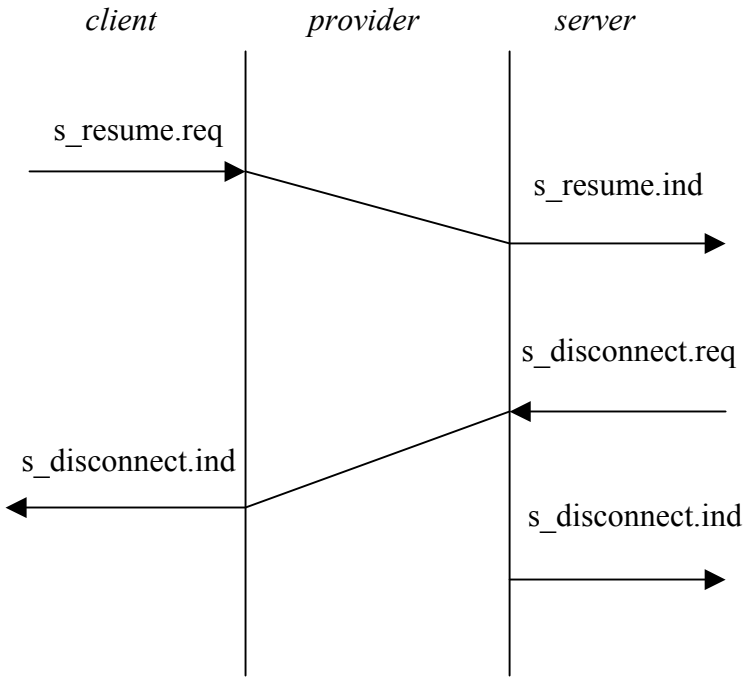
Client Address	M	M (=)	-	-
Client Headers	O	C (=)	-	-
Server Headers	-	-	O	C (=)

Seperti yang terlihat pada Gambar 2.6. jika *client* ingin melanjutkan session yang ditangguhkan maka *client* akan mengirimkan s_resume.req. dan dan jika *server* menyetujuinya maka *server* akan merespon dengan mengirimkan s_resume.res.



Gambar 2.6. urutan primitif proses resume yang diterima oleh *server* (http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

Seperti yang terlihat pada Gambar 2.7. jika *server* menolak proses resume yang dilakukan oleh *client* maka setelah *client* mengirimkan s_resume.req maka *server* akan langsung meresponnya dengan mengirimkan s_disconnect.ind.



Gambar 2.7. urutan primitif proses resume yang ditolak oleh *server*
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

- S_methodinvoke

Digunakan untuk membuat *request post* atau *get*. *Post request* berguna untuk mengirimkan MMS ke MMSC (MMS centre). Parameter yang ada dalam s_methodinvoke dapat dilihat pada Tabel 2.6. berikut :

Tabel 2.6. Parameter S_Methodinvoke
(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

	Primitive			
Parameter	**Req**	**Ind**	**Res**	**Cnf**
Client Transaction ID	M	-	-	M (=)
Server Transaction ID	-	M	M (=)	-
Method	M	M (=)	-	-
Request URI	M	M (=)	-	-
Request Header	O	C (=)	-	-
Request Body	C	C (=)	-	-
More data	M	M (=)	-	-

- S_methodresult

Digunakan untuk menyatakan respon terhadap *request post* atau *get* yang dibuat oleh s_methodinvoke. Parameter yang ada dalam s_methodresult dapat dilihat pada Tabel 2.7. berikut

Tabel 2.7. Parameter S_Methodresult

(http://www.wmlclub.com/docs/especwap2.0/WAP-230-WSP-20010705-a.pdf)

Parameter	**Primitive**			
	Req	**Ind**	**Res**	**Cnf**
Client Transaction ID	M	-	-	M (=)
Server Transaction ID	-	M	M (=)	-
Status	M	M (=)	-	-
Response Header	O	C (=)	-	-
Response Body	C	C (=)	-	-
Acknowledgement Headers	-	-	O	P (=)
More data	M	M (=)	-	-

2.3. *Multimedia Messaging Service* (MMS)

Multimedia Messaging Service adalah suatu metode pengiriman yang mampu mengirimkan data berupa gambar, suara dan teks sekaligus. Layanan MMS hanya dimungkinkan beroperasi dalam jaringan GPRS. Saat ini layanan MMS memungkinkan untuk pengiriman antar operator tetapi masih belum mendukung layanan *international roaming. Content* yang diperbolehkan pada saat pengiriman dan penerimaan MMS di *handphone* yaitu : teks, gambar, klip suara atau kombinasi dari ketiganya. Dasar dari implementasi MMS adalah kombinasi dari penggunaan teknologi *Wireless Application Protocol* (WAP) dan teknologi *Short Message Service* (SMS).

2.3.1. Proses Sederhana Pengiriman MMS

Berikut adalah proses kerja pengiriman MMS kepada *handphone* :

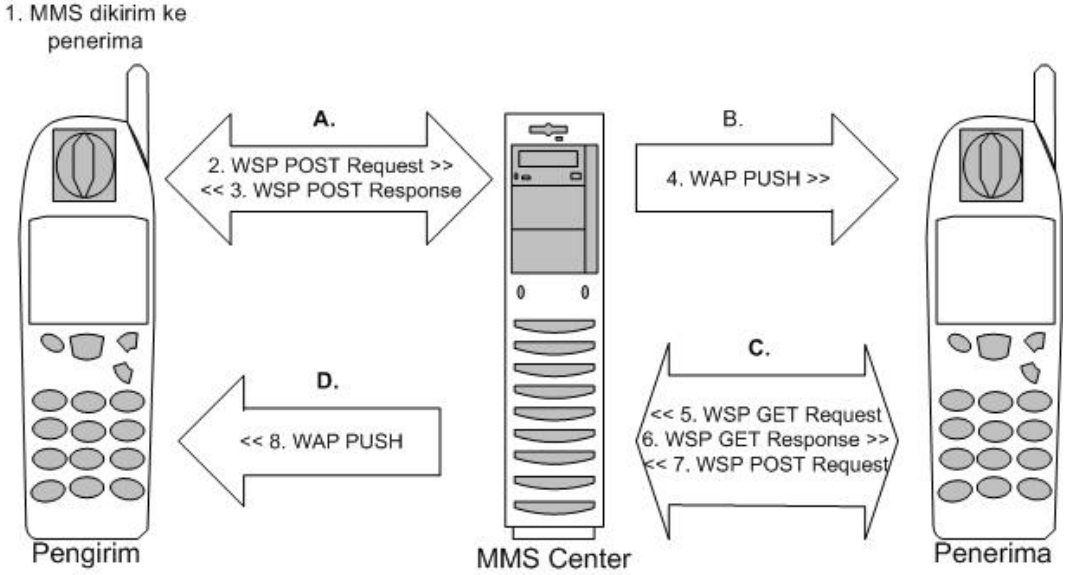
1. Pertama-tama dimulai dari *content file* yang memuat isi dari suatu pesan yang bertipe MMS. Pesan ini dapat dibuat melalui suatu program aplikasi atau dibuat oleh *user* yang mengirimkan *message* melalui *handphone*. (ketika message dikirim dari *handphone* maka *handphone* tersebut terhubung pada

MMSC melalui WAP dan mengirimkan isi dari *message* tersebut kepada MMSC untuk selanjutnya MMSC akan bertanggungjawab atas pengiriman selanjutnya).

2. Isi dari *message* MMS tersebut di-*publish* sehingga dapat diakses melalui URL.
3. Sebuah MMS *notification message* (*special binary message* yang dikirimkan melalui SMS) dikirimkan kepada *handphone* penerima. MMS *notification message* yang terdiri dari *header information* tentang MMS *message* dan *pointer* URL sehingga penerima dapat membuka isi MMS *message* yang dikirimkan.
4. Penerima membuka WAP *session* untuk dapat membuka isi MMS *message* yang dikirimkan.

Jadi penerima yang tidak mempunyai *handphone* yang mendukung MMS tetap dapat membaca pesan yang dikirimkan melalui URL yang tersedia karena sebelumnya menerima SMS yang berisi *header information* dan URL tempat message di-*publish*.

2.3.2. Detail Proses Pengiriman MMS



Gambar 2.8. Proses pengiriman MMS
(http://24.234.57.173/p17/MMS/HowToCreateMMSServices.pdf)

Seperti yang terlihat pada Gambar 2.8. detail proses pengiriman MMS adalah :

A. Pengirim mengirim MMS

1. Pengirim MMS memasukkan tujuan MMS kepada nomor penerima
2. Terminal (dalam hal ini HP) yang memiliki informasi tentang MMSC, menginisialisasi sebuah koneksi WAP, yang bisa melalui CSD (*Circuit Switched Data*) atau GPRS (*General Packet Radio Service*). Kemudian pesan MMS dikirim sebagai *content* dari sebuah WSP (*Wireless Session Protocol*) POST.
3. MMSC menerima pesan MMS tersebut dan merespon kepada pengirim MMS melalui koneksi WAP yang sama. Pada terminal (HP) pengirim akan mengindikasikan "*Message Sent*".

B. MMSC menginformasikan Penerima

4. MMSC akan menggunakan WAP PUSH untuk mengirimkan pesan ke penerima bahwa ada pesan MMS baru.

C. Penerima menerima MMS

5. Dengan mengasumsikan bahwa terminal penerima sudah disetting untuk menerima pesan MMS, maka terminal itu akan menginisiasikan sebuah koneksi WAP dan menggunakan fasilitas WSP GET untuk men*download* pesan MMS tersebut dari MMSC.
6. Pesan MMS dikirim kepada penerima sebagai *content* dari WSP GET RESPONSE melalui koneksi WAP yang sama. Lalu terminal penerima akan mengindikasikan "*Message Received*".
7. Terminal penerima akan mengkonfirmasikan penerimaan pesan dengan WSP POST, masih melalui koneksi WAP yang sama.

D. MMSC memberitahukan pengirim tentang status pengiriman.

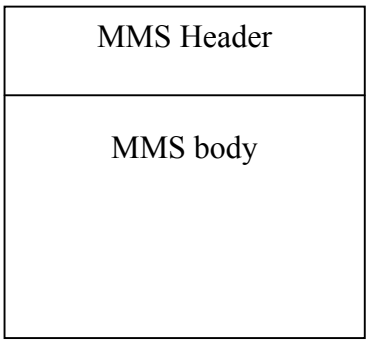
8. MMSC menggunakan WAP PUSH untuk memberitahukan kepada pengirim MMS bahwa pesan tersebut sudah terkirim. Pada terminal pengirim akan tertulis "*Message Delivered*".

Jadi peran WAP dalam proses pengiriman MMS adalah sebagai media penghantar antara pengirim atau penerima MMS dengan MMSC. Untuk dapat mengirimkan

pesan MMS pengirim harus mengadakan koneksi ke WAP *gateway*, jika koneksi koneksi ke WAP *gateway* sudah terhubung maka pengirim dapat mengirimkan pesan MMS melalui metode WSP *post* ke WAP *gateway*, dari WAP *gateway* pesan yang dikirim akan diteruskan ke MMSC. Jika penerima ingin men*download* pesan yang dikirim maka penerima mengirimkan sinyal kepada WAP *gateway*. WAP *gateway* lalu men*download* pesan dari MMSC dan mengirimkannya ke user.

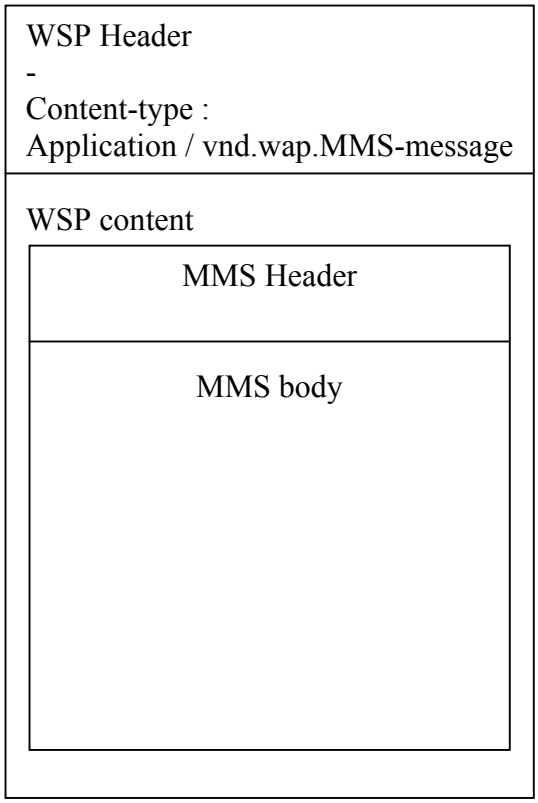
2.3.3. Format dari MMS

Dalam komunikasi seperti pada Gambar 2.8. diatas, yang sebenarnya dikirim adalah MMS PDUs (*Protocol Data Units*). Sebuah MMS PDU terdiri dari *header* dan *body* MMS. Tapi kadang-kadang MMS PDU dapat dikirim tanpa harus memiliki *body*. Hanya pada langkah 2 dan 6 saja, MMS PDU tersebut memiliki *body*. Sedangkan pada langkah lainnya PDU hanya terdiri dari bagian *header* saja.



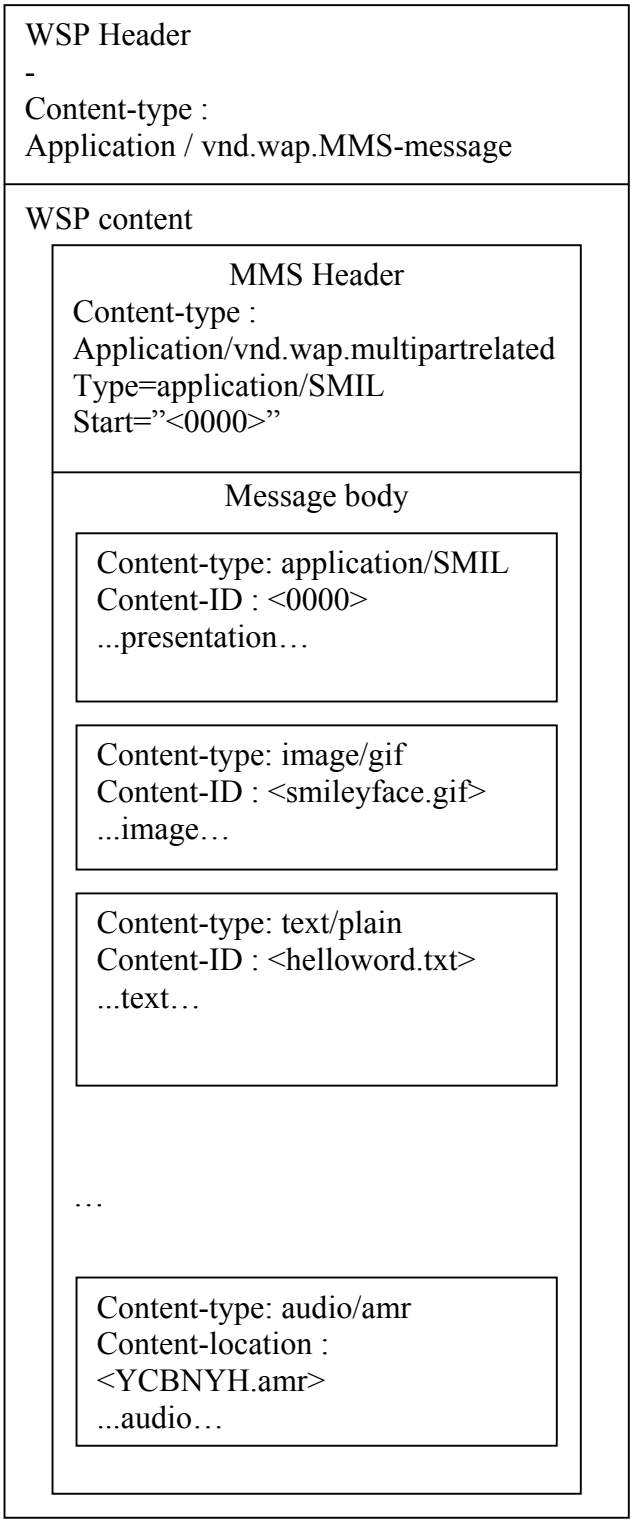
Gambar 2.9. MMS Protocol Data Unit
(http://24.234.57.173/p17/MMS/HowToCreateMMSServices.pdf)

Seperti yang terlihat pada Gambar 2.9. MMS PDUs ini dimasukkan dalam bagian *content* dari WSP atau HTTP, tergantung dari *transport protoco*l yang dipakai, dan *content-type* dari PDUs ini di-*set* ke "application/vnd.wap.mms-message". Berikut ini adalah contoh dari WSP :

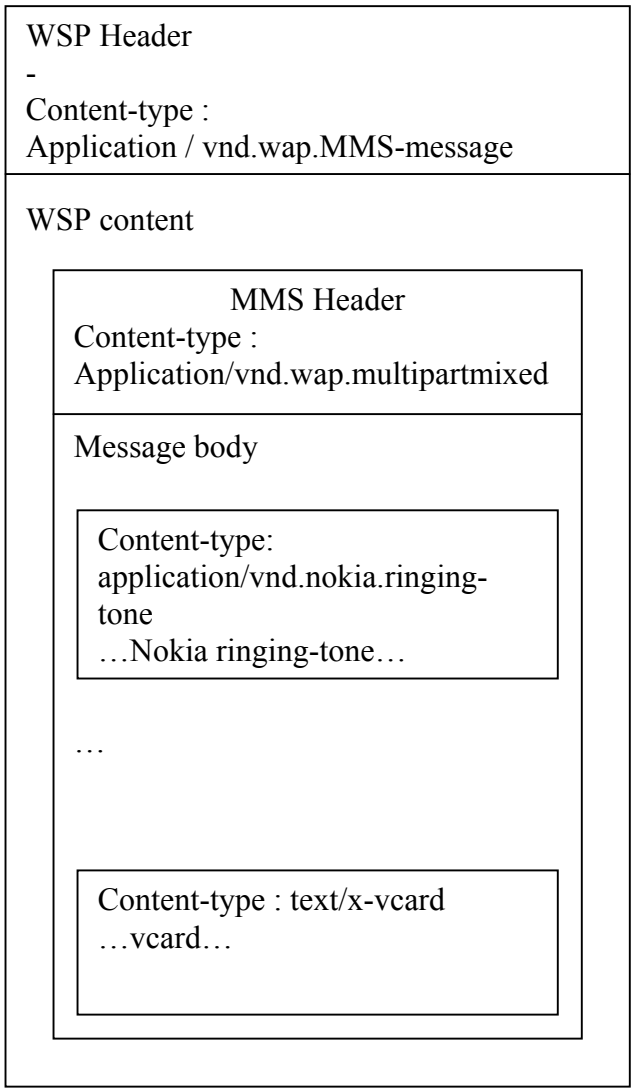


Gambar 2.10. WSP *message* dengan MMS PDUs
(http://24.234.57.173/p17/MMS/HowToCreateMMSServices.pdf)

Dari dalam isi dari MMS PDU, maka dapat dilihat bahwa bagian *body* dari PDU terdiri dari set-set *content* yang lain, seperti terlihat pada Gambar 2.10.



Gambar 2.11. WSP *message* dengan MMS PDU - *multipartrelated*
(http://24.234.57.173/p17/MMS/HowToCreateMMSServices.pdf)



Gambar 2.12. WSP *message* dengan MMS PDU – *multipartmixed*
(http://24.234.57.173/p17/MMS/HowToCreateMMSServices.pdf)

Seperti yang terlihat pada Gambar 2.11. dan Gambar 2.12. *content-type* pada MMS header terdiri dari dua macam yaitu application/vnd.wap.multipartrelated dan application/vnd.wap.multipartmixed. Application/vnd.wap.multipartrelated digunakan bila MMS message yang dibuat mempunyai *presentation part.* Salah satu bahasa yang dapat membuat *presentation part* adalah *Synchronized Multmedia Integration Language* (SMIL) sedangkan application/vnd.wap.multipartmixed digunakan bila MMS *message* tidak mempunyai *presentation part.*

Synchronized Multmedia Integration Language (SMIL) adalah suatu bahasa yang mirip dengan HTML yang dapat membuat presentasi multimedia. Jika suatu pesan MMS tidak menggunakan SMIL maka obyek yang berupa gambar, teks, dan suara akan ditampilkan satu persatu tetapi jika memakai SMIL maka semua obyek akan dapat ditampilkan bersama-sama sesuai dengan keinginan.

2.3.4. Operator MMS

Ada beberapa operator GSM yang menyediakan layanan pengiriman MMS antara lain adalah Telkomsel. Jenis layanan MMS Telkomsel yang tersedia saat ini :

- Pengiriman pesan dari ponsel MMS ke ponsel MMS.
- Pengiriman pesan dari ponsel MMS ke ponsel non MMS.
- Pengiriman pesan dari ponsel MMS ke alamat *e-mail*.

2.3.5. Nokia MMS java library (Nokia MMS Java Library v1.1.pdf)

Nokia MMS java library menyediakan MMS library yang berguna untuk pembuatan dan pengiriman pesan MMS. Selain itu pada Nokia MMS java library juga menyediakan contoh-contoh bagaimana membuat, melakukan *encode*, mengirim dan melakukan *decode* pada multimedia *message*.

Nokia MMS java library menyediakan MMS library tentang :

- *Message creation and encoding*

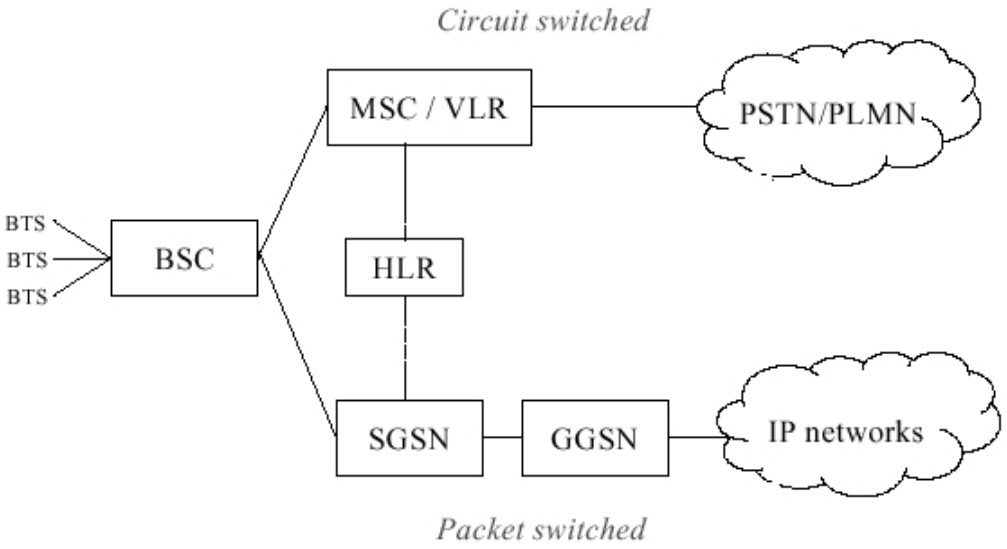
  Nokia MMS *library* menyediakan *method-method* untuk melakukan *encode* pada *multimedia message,* membuat pesan baru dengan *content type* yang berbeda, dan juga untuk mengisi pesan MMS dengan *content* yang diinginkan.
- *Message decoding*

  Nokia MMS library menyediakan *method-method* untuk melakukan *decode* pada *multimedia message.*

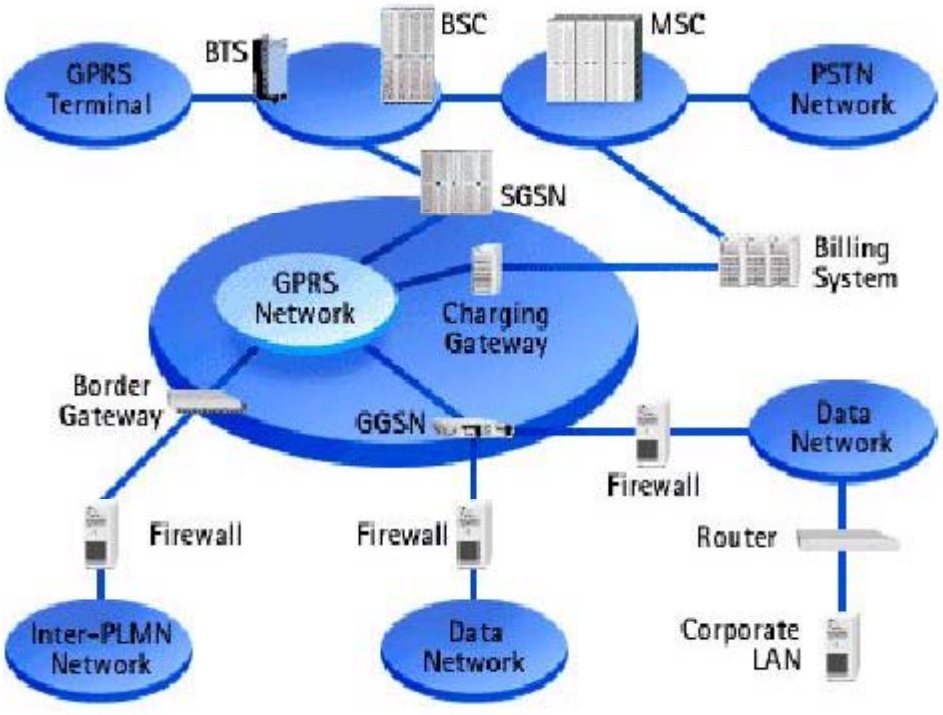
2.4. *General Packet Radio Service* (GPRS)

Jaringan GSM mempunyai dua macam *network* yaitu *circuit switched network* yang berguna untuk pengiriman *voice* serta *packed switched network* yang berguna untuk pengiriman data. Fasilitas GPRS diimplementasikan melalui jaringan GSM yang telah ada. Untuk itu ditambahkan dua elemen *network* baru ke dalam jaringan GSM, yaitu *Gateway GPRS Support Node*(GGSN) dan *Serving GPRS Support Node*(SGSN).

Seperti yang terlihat pada Gambar 2.13. *Base Transceiver Station*(BTS) berguna untuk menyediakan *service* bagi pengguna *handphone*. BTS dikontrol oleh *Basestation controler*(BSC). BSC berfungsi untuk mengatur dan menjembatani antara BTS dengan *Mobile Switching Center*(MSC) atau dengan GGSN. Secara garis besar fungsi dari SGSN dan MSC adalah sama, yang membedakan adalah jika MSC mengatur lalu lintas percakapan suara dan mengatur sistem pembayaran untuk percakapan, sedangkan SGSN mengatur lalu lintas pengiriman data dan mengatur sistem pembayaran untuk penggunaan GPRS. Sedangakan GGSN berguna untuk menjembatani antara SGSN dengan *external network*. Biasanya GGSN difungsikan menjadi WAP *gateway*. Sedangkan *Home Location Register*(HLR) adalah *database* yang berisi data pelanggan. Sedangkan *Visitor Location Register*(VLR) adalah *database* sementara yang berisi data yang dibutuhkan oleh MSC.



Gambar 2.13 Struktur GPRS network
(http://www.ics.lu.se/publikationer/inf01/inf01-009.pdf)



Gambar 2.14 Detail jaringan GPRS
(http://whitepapers.zdnet.co.uk/0,39025945,60054584p-39000517q,00.htm)

Pada Gambar 2.14. menunjukkan detail dari GPRS *network*. Pada GPRS *network* dapat terdiri dari beberapa BSC, SGSN, dan MSC. Banyaknya BSC, SGSN, ataupun MSC disesuaikan dengan kebutuhan. Selain itu pada Gambar 2.14. juga menunjukkan adanya *firewall* yang berfungsi untuk mencegah adanya gangguan yang dapat merusak sistem GPRS serta adanya *Billing system* untuk menghitung biaya yang digunakan pelanggan pada saat menggunakan fasilitas GPRS.

2.5. *Motion Detection*

Untuk mendeteksi adanya gerakan dapat dilakukan dengan 2 macam cara yaitu dengan menggunakan program eksternal atau dengan menggunakan algoritma sederhana yang dapat dibuat sendiri. Jika digunakan program *external* yang kadang2 dilengkapi dengan *built-in motion detection*, mis : WebCam32, dalam hal ini, sebuah *image* baru akan di*upload*/di*save* hanya bila ada gerakan terdeteksi. Dalam hal ini *program* tinggal mengecek apakah ada *image* baru yang baru di-*upload* atau di-*save* ke *disk*. Jika ada, maka sebuah pesan MMS akan dikirimkan ke *user*.

Bila menggunakan algoritma sederhana, *Motion detection* dapat dicapai dengan membandingkan suatu *image* dengan *image* yang lain yang diambil (di*capture*) secara berurutan.. Misalkan sebuah *image* dengan 160x120 *pixel* diambil (*capture*). *Image* ini dapat direpresentasikan sebagai *array* 160x120, dgn *entry* tiap *array* mewakili satu *pixel*. Tiap *pixel* dapat dikode dengan 3 *bytes* (Red, Green, blue). *Coding* lain juga dapat digunakan seperti *coding* 16 bit YUV, 8 bit *colour pallet number*, dsb.

Selain itu juga dapat dilakukan dengan mengkonversi *image* ke *Black and White*, 1 *byte* per *pixel*. Dengan menjumlahkan nilai absolut dari perbedaan tiap *pixel* baru dan lama, akan didapatkan sebuah ukuran jumlah perbedaan antara 2 *image*.

```
Contoh :
  Image1: array[1..160,1..120];
  Image2: array[1..160,1..120];
```

```
GetImageAndConvert(Image1)
Sleep(1)
GetImageAndConvert(Image2)
sum : = 0;
for I : =1 to 160 do
    for  j : = 1 to 120 do
      sum : = sum + abs ( Image2 [i,j]  - Image1 [i,j] );
if sum  >  'Nilai Tertentu' Then 'Ada gerakan terdeteksi'
```

2.6. Java

Java merupakan teknologi di mana teknologi tersebut mencakup java sebagai bahasa pemrograman yang memiliki sintaks dan aturan pemrograman sendiri juga sebagai *platform* dimana teknologi ini memiliki *virtual machine* dan *library* yang diperlukan untuk menulis dan menjalankan program yang ditulis dengan bahasa pemrograman java. (Rickyanto, Isak 2003)

Bahasa pemrograman java menjadi bahasa pemrograman yang sangat popoler sebab dapat dijalankan di berbagai *operating system* (OS). Misalnya windows, linux. Dengan demikian bahasa pemrogrman java telah mengatasi masalah portabilitas yang sering menjadi kendala dan hambatan dalam pembuatan suatu *software* karena biasanya sebuah *software developer* harus mengeluarkan banyak tenaga, pikiran dan waktu untuk menghasilkan aplikasi yang dapat berjalan di *operating system* yang lain. (Rickyanto, Isak 2003)

Java juga mempunyai beberapa karakteristik yang membuat java semakin unggul antara lain :

- Sederhana (*simple*)

  Java tidak memiliki sintaks yang aneh tetapi banyak menggunakan sintaks bahasa pemrograman C++ yang sudah banyak dikenal sehingga tidak menyulitkan para *programmer*. Selain itu java juga memberikan banyak keuntungan dan kemudahan dibandingkan dengan C++. (Flanagan, David 1997)

- Berorientasi obyek (*object-oriented*)

  Java merupakan pemrograman yang berorientasi obyek yang murni. Dalam pemrograman java semua adalah obyek kecuali tipe data primitif. (Flanagan, David 1997)

- Terdistribusi (*distributed*)

  Java menyediakan banyak kemudahan dalam hal *networking*. Salah satu contohnya ialah RMI (*Remote Method Invocation*) API (*Application Programming Interface*) yang membuat program java mampu menjalankan metode dari *remote java object* seakan-akan menjalankan metode dari *local java object*. (Flanagan, David 1997)

- Aman (*secure*)

  Aman karena program java memiliki *library security* serta *policy* yang membatasi akses *applet* di komputer *client*. (Flanagan, David 1997)

- Diinterpretasi (*interpreted*)

  Java memerlukan *virtual machine* yang bertindak sebagai *interpreter* yang menterjemahkan *bytecode* menjadi bahasa mesin yang dimengerti oleh komputer *host*. (Flanagan, David 1997)

- Portabel (*portable*)

  Portabel karena java dapat dijalankan di berbagai *platform* tanpa perubahan kode sama sekali. (Flanagan, David 1997)

- *Multithreading*

  Java memiliki kemampuan untuk menangani dan menjalankan banyak *thread* sekaligus. (Flanagan, David 1997)

- Dinamis (*dynamic*)

  Java merupakan teknologi yang terus berkembang. Hal ini dapat dilihat dari *library* yang semakin lengkap. Selain itu program java telah dapat diimplementasikan untuk aplikasi mobile dengan adanya Java 2 Micro Edition (J2ME). (Flanagan, David 1997)

- Netral terhadap arsitektur *hardware* (*architecture neutral*)

  Java dapat dijalankan dengan baik di semua arsitektur komputer yang berbeda. (Flanagan, David 1997)

- *Robust*

  Java mampu menolong para programmer untuk menghasilkan program secara cepat dan handal karena java mencegah adanya *memory leaking*, meniadakan *pointer* serta mencegah berbagai macam *error* yang mungkin terjadi dengan adanya berbagai pengecekan awal pada kompilasi. (Flanagan, David 1997)

2.7. Tutorial Nokia MMS Java Library

Untuk dapat membuat dan mengirimkan pesan MMS melalui komputer dibutuhkan Nokia MMS java library. Nokia java library diperlukan untuk membuat kerangka MMS, melakukan proses *encode*, *decode*, dan pengisian *content* pada kerangka MMS. Nokia MMS java library dapat di-*download* dari (http://www.forum.nokia.com/main/0%2C6566%2C034-21%2C00.html)

MMS java library hanya memiliki satu buah *package* yaitu com.nokia.mms. Pada package com.nokia.mms terdapat sebuah *interface* yaitu IMMConstants. com.nokia.mms mempunyai tujuh buah *class* yaitu :

- MMAddress : Bertujuan untuk merepresentasikan *generic address* dari pengirim dan penerima MMS.
- MMContent : Bertujuan untuk merepresentasikan *generic entry* dari suatu pesan MMS
- MMDecoder : Bertujuan untuk melakukan proses *decode* pada pesan MMS yang dibuat.
- MMEncoder : Bertujuan untuk melakukan proses *encode* pada pesan MMS yang dibuat.
- MMMessage : Bertujuan untuk merepresentasikan suatu MMS.
- MMResponse : Berisi *method* untuk mendapatkan respon dari MMSC.
- MMSender : Berisi *method* untuk mengirimkan MMS kepada MMSC.

Selain mempunyai *interface* dan *class*, com.nokia.mms juga mempunyai *exception* yaitu :

- MMDecoderException : Bertujuan untuk mengetahui error yang terjadi pada waktu proses *decode* suatu pesan MMS.
- MMEncoderException : Bertujuan untuk mengetahui error yang terjadi pada waktu proses *encode* suatu pesan MMS.
- MMSenderException : Bertujuan untuk mengetahui error yang terjadi pada waktu proses pengiriman suatu pesan MMS.

2.7.1. Pembuatan Kerangka MMS dan Pengisian *content* MMS

Class yang digunakan untuk pembuatan kerangka MMS dan pengisian *content* MMS adalah MMMessage. Contoh pembuatan kerangka MMS dapat

dilihat pada Segmen program 2.1. dan Contoh pengisian *content* MMS dapat dilihat pada Segmen program 2.2.

Segmen program 2.1. Contoh Pembuatan Kerangka MMS

```
MMMessage mm=new MMMessage();

  mm.setVersion(IMMConstants.MMS_VERSION_10);
  mm.setMessageType(IMMConstants.MESSAGE_TYPE_M_SEND_REQ);
  mm.setTransactionId("000001");
  mm.setDate(new Date(System.currentTimeMillis()));
  mm.setFrom("+358990000003/TYPE=PLMN");
  mm.addToAddress("+358990000005/TYPE=PLMN");
  mm.addCcAddress("+358990000003/TYPE=PLMN");
  mm.setSubject("Hello guys !");
  mm.setMessageClass(IMMConstants.MESSAGE_CLASS_PERSONAL);
  mm.setPriority(IMMConstants.PRIORITY_HIGH);
  mm.setContentType(IMMConstants.CT_APPLICATION_MULTIPART_MIXED);

// jika sudah mendukung teknologi SMIL maka content type diubah
menjadi :
mm.setContentType(IMMConstants.CT_APPLICATION_MULTIPART_RELATED);
mm.setMultipartRelatedType(IMMConstants.CT_APPLICATION_SMIL);
mm.setPresentationId("<A0>"); // dimana <A0> adalah id dari
content yang berisi presentasi SMIL
```

Segmen program 2.2. Contoh Pengisian *Content* MMS

```
MMContent part1=new MMContent();

// membaca file dengan readFile(), yaitu function yang membaca
sebuah file dan mengembalikannya dalam bentuk array of bytes
  byte [] buf1 = readFile("text.txt");
  part1.setContent(buf1,0,buf1.length);
  part1.setContentId("text.txt");
  part1.setType(IMMConstants.CT_TEXT_PLAIN);
  mm.addContent(part1);

  MMContent part2=new MMContent();
  byte [] buf2 = readFile("img1.jpg");
  part2.setContent(buf2,0,buf2.length);
  part2.setContentId("img1.jpg");
  part2.setType(IMMConstants.CT_IMAGE_JPEG);
  mm.addContent(part2);
```

Pengisian *content* MMS terdiri dari beberapa bagian. Pada segmen program 2.2. pengisian *content* MMS terdiri dua bagian yaitu teks dan gambar. Akan tetapi *user* dapat menambahkan *content* yang lain pada part yang berbeda. Selain teks dan gambar content MMS juga dapat berupa *sound clip* dan *video clip*. Dalam sebuah pesan MMS tidak boleh berisi *content* yang sama.

2.7.2. Proses *Encode* Pesan MMS

Setelah pesan MMS selesai dibuat maka proses selanjutnya adalah melakukan *encode* pesan MMS. Pesan MMS perlu di-*encode* terlebih dahulu agar dapat diterima oleh MMSC. Contoh proses *encode* dapat dilihat pada segmen program 2.3.

Segmen program 2.3. Proses *Encode* MMS

```
// inisialisasi sebuah encoder object
   MMEncoder encoder=new MMEncoder();

// set pesan yang akan di-encode
   encoder.setMessage(mm);

// encode pesan MMS
   try {
     encoder.encodeMessage();
   } catch (MMEncoderException e) {
     System.err.printlm("An error occurred encoding the
message.");
   }

   byte [] buf=encoder.getMessage();
```

2.7.3. Proses *Decode*

Proses *decode* berfungsi untuk melakukan *decode* terhadap respon dari MMSC. Contoh proses *decode* dapat dilihat pada segmen program 2.4.

Segmen program 2.4. Proses *Decode* MMS

```
// kode untuk membaca array of bytes dari pesan yang akan di-
decode
   byte[] buf = ....

   // inisialisasi sebuah MMDecoder object
   MMDecoder mmDecoder = new MMDecoder();

   // set the buffer
   mmDecoder.setMessage(buf);

   // melakukan proses decode
   try {
    mmDecoder.decodeMessage();
   } catch (Exception e) {
    System.err.println(e.getMessage());
   }


   MMMessage mm=mmDecoder.getMessage();
```

2.8. Tutorial JWAP *Protocol Stack*

Untuk dapat melakukan pengiriman pesan MMS dibutuhkan JWAP *protocol stack*. JWAP *protocol stack* dapat di-*download* dari (http://sourceforge.net/project/showfiles.php?group_id=49484).

Sebelum melakukan pengiriman MMS, terlebih dahulu dilakukan koneksi ke WAP *gateway*. Contoh untuk melakukan koneksi ke WAP gateway dapat dilihat pada segmen program 2.5.

Untuk mengirimkan pesan MMS yang dibutuhkan adalah metode *post* pada JWAP *protocol stack*. Contoh penggunaan metode *post* pada JWAP *protocol stack* dapat dilihat pada segmen program 2.5.

Segmen program 2.5. Koneksi ke WAP *gateway*

```
public class CWapPostExample implements IWSPUpperLayer{
   // URL dan port dari WAP-Gateway
   private InetAddress wapgw;
   private int wapgwport;
   // URI dari server dimana pesan MMS akan di-post
   private String posturi;
   // besar yang akan di-post
   private byte[] bytes;
   private String contentType;
   // menjalankan WAP session untuk mengirimkan pesan (dari
connect sampai disconnect)
   private CWSPSession s;
   // menjalankan Method transaction untuk posting (dari
m_send_req sampai m_send_conf)
   private CWSPMethodManager m;


   private void init()
   {
     try
     {
        s = new CWSPSession(wapgw, wapgwport, this);
        System.out.println("connecting to WAP gateway " + wapgw);
        s.s_connect();
     }
     catch (Exception e)
     {
        e.printStackTrace();
     }
   }
```

Setelah terjadi koneksi ke WAP *gateway* maka selanjutnya adalah mengirimkan pesan MMS dengan menggunakan metode POST. Contoh penggunaan metode POST dapat dilihat pada segmen program 2.6.

Segmen program 2.6. Metode *Post* pada JWAP *protocol stack*

```
public  void s_connect_cnf()
{
   try
   {
      System.out.println("connected to WAP gateway " + wapgw);

      System.out.println("sending data (WSP POST)");
      m = s.s_post(bytes,
                   contentType,
                   posturi);

   }
   catch (Exception e )
   {
      e.printStackTrace();
   }
}
```

Setelah dilakukan proses POST maka untuk mengetahui apakah pesan sudah berhasil terkirim atau tidak diperlukan respon dari MMSC. Untuk mengetahui respon dari MMSC digunakan metode "methodresult". Contoh penggunaan metode "methodresult" dapat dilihat pada segmen program 2.7.

Segmen program 2.7. Penggunaan Metode Methodresult

```
public void s_methodResult_ind(byte[] payload,
                                String contentType,
                                boolean moreData)
   {  //  disini dilakukan proses decode respon dari MMSC

  String nl = System.getProperty("line.separator");


  System.out.println("Response Content Type: " +  contentType + nl
             + payload.toString()
             );

  System.out.println("disconnecting from WAP gateway " + wapgw);
      m.s_methodResult(null);
      s.s_disconnect();
   }
```