BAB II

LANDASAN TEORI

1.1 Serialisasi Dokumen

Serialisasi adalah proses yang berjalan secara *runtime* untuk mengkonversi obyek ke dalam bentuk sekuensial byte secara linier. Kegunaan utama dari proses serialisasi adalah pemrosesan lebih lanjut hasil serialisasi ke sebuah bentuk blok memori untuk ditransfer melalui jaringan dengan protokol yang umum.

Sebuah proses serialisasi dapat menghasilkan tiga macam bentuk output antara lain :

1. Binary
2. *Simple Object Access Protocol*(SOAP)
3. XML

Khusus untuk proses serialisasi yang mampu menghasilkan dokumen XML, merupakan sebuah antarmuka yang tersedia secara publik agar proses serialisasi mampu mengkonversi sebuah obyek ataupun dokumen ke dalam bentuk XML. Skema dengan kemampuan yang mampu menangani referensi secara majemuk. Dalam kata lain, bahwa proses serialisasi yang menghasilkan dokumen XML mampu menangani beberapa obyek atau dokumen sekaligus untuk dapat dikonversikan menjadi sebuah dokumen XML secara langsung.

Proses serialisasi yang menghasilkan dokumen XML memiliki beberapa ciri khusus antara lain :

1. Properti yang konsisten

STIKOM SURABAYA

Karena dokumen XML mensyaratkan sebuah tag yang tidak terbatas, maka properti yang terjadi dalam sebuah dokumen XML hasil serialisasi akan selalu konsisten. Dokumen XML yang terbentuk juga tidak akan mengikutsertakan properti khusus dari dokumen asal, misalkan: properti *read only*, properti *private* dan lainnya.

2. Identitas obyek

Karena dokumen XML memiliki aturan tentang elemen dan atribut, maka identitas obyek berupa *namespace* dan lainnya akan hilang saat proses deserialisasi dilakukan. Tetapi hal ini akan sangat berguna saat terjadi proses serialisasi untuk beberapa obyek sekaligus, sehingga tiap obyek dapat dipisahkan dengan elemen XML secara tersendiri.

3. Kontrol terhadap output

Dengan adanya kontrol terhadap tiap elemen yang terbentuk, maka dokumen XML hasil serialisasi dapat dimodifikasi untuk keperluan yang lebih luas, misal dengan menerapkan terhadap class lain sehingga dapat diatur hanya obyek tertentu yang dapat di deserialisasi.

1.2 XML

Extensible Markup Language (XML) adalah sebuah subset dari SGML (*Standard Generalized Markup Language*), yang sasarannya memungkinkan SGML generik dapat diterima dan diproses web dengan cara yang sekarang dipakai untuk HTML. XML dirancang untuk kemudahan implementasi dan untuk interopabilitas dengan SGML dan HTML. Pada awalnya XML didefinisikan oleh XML *Working Group* W3C(*World Wide Web Consortium*) pada bulan Februari 1998.

STIKOM SURABAYA

Sebuah dokumen XML secara umum merupakan sebuah dokumen yang memenuhi protokol standart industri yang memiliki sifat *platform independent* dan mampu ditranspotasikan dengan menggunakan protokol internet umum seperti halnya HTTP. Sebuah dokumen XML bisa memiliki pengertian bias dan rancu tergantung dari konteks dan kasus yang dihadapi dalam pemanfaatan XML itu sendiri. Dokumen XML merupakan sebuah dokumen berbasis teks yang mampu menampung hasil konversi dari dokumen citra digital sekaligus melakukan kompresi sehingga dokumen menjadi berukuran lebih kecil

XML bukan suatu "bahasa", namun suatu standar untuk membuat suatu bahasa yang sesuai dengan kriteria XML. Berarti bahwa XML mendeskripsikan sintak yang dapat digunakan untuk membuat suatu bahasa sendiri. Hal ini dijelaskan dengan contoh di bawah ini :

<name>

<first>Jhon</first>

<last>Doe</last>

</name>

Dari contoh di atas, bisa dilihat kenapa bahasa markup seperti SGML ataupun XML disebut sebagai *self described*. Dari data di atas dengan mudah dapat dilihat informasi tentang <name>, kita juga tahu ada data <first> dan <last>.

Berikut ini adalah sepuluh sasaran XML seperti yang dinyatakan dalam spesifikasi XML resmi yang ditempatkan pada situs web W3C :

1. XML harus dapat dipakai langsung pada Internet.
2. XML harus mendukung berbagai macam aplikasi.
3. XML harus kompatibel dengan SGML.

STIKOMP SURABAYA

4. XML harus memudahkan penulisan program yang memproses dokumen XML.
5. Jumlah fitur opsional dalam XML dibuat seminim mungkin, idealnya nol.
6. Dokumen XML harus bisa dipahami oleh manusia dan jelas.
7. Desain XML harus dapat disiapkan dengan cepat.
8. Desain XML harus bersifat formal dan ringkas.
9. Dokumen XML harus mudah dibuat.
10. Keringkasan dalam markup XML tidak begitu dipentingkan

Dokumen XML memiliki dua macam validasi yaitu:

A. *Well formed*

Sebuah dokumen XML dianggap sebagai dokumen yang valid jika telah memenuhi struktur *tag* secara konsisten, serta memiliki elemen atau atribut yang saling berpasangan.

B. Valid

Sebuah dokumen XML yang telah melalui validasi well formed dapat dikatakan valid jika telah memiliki DTD(*Document Tipe Difinition*) atau dapat dibuatkan sebuah skema secara benar dengan definisi yang telah ada di dalam struktur tag dokumen XML tersebut. Di dalam Visual Studio .Net validasi XML akan membuat skema baru untuk dokumen XML dan namespace baru untuk dokumen XML tersebut.

1.3 Class XMLSerializer

XMLSerializer adalah suatu class pada Visual Basic .NET yang digunakan untuk merubah suatu obyek menjadi dokumen XML maupun sebaliknya dari dokumen XML menjadi suatu serialisasi obyek secara umum akan

STIKOMP SURABAYA

memanfaatkan kemampuan Visual Studio.NET untuk membaca memori stream dalam format tipe byte. Sehingga obyek yang dapat diserialisasi lebih bervariasi, misal: dokumen gambar, dokumen pdf hingga runtime library.

Class XMLSerializer adalah suatu class yang diturunkan dari namespace System.Runtime.Serialization dari lingkup .NET framework. Class tersebut nantinya mampu memanfaatkan kemampuan XML dalam memisahkan atribut ataupun elemen yang dihasilkan, sehingga serialisasi obyek dapat berjalan secara majemuk. Kemampuan ini akan banyak membantu para programmer untuk dapat melakukan pengelompokan dari beberapa dokumen secara bersamaan, bahkan untuk beberapa dokumen dengan tipe yang sama sekali berbeda.

Selain itu, keuntungan lain dari class tersebut adalah kemampuan untuk memilih format *encoding* file, sehingga dapat menyelaraskan diri dengan format dokumen internasional, dan juga bisa dimanfaatkan untuk mencegah pihak yang tidak berhak agar tidak mampu membaca hasil serialisasi, kecuali pihak tersebut memiliki kemampuan *decoding* dari format encoding yang telah ditentukan sebelumnya.

Class yang mendukung serialisasi dalam .NET framework mampu merepresentasikan hasilnya dalam dua bentuk yaitu bentuk biner (binary *formatter*) dan format yang mendukung SOAP (SOAP formatter). Deskripsi dari format hasil serialisasi sendiri adalah obyek yang dihasilkan dari proses serialisasi data.

Saat proses serialisasi yang menghasilkan dokumen XML berlangsung nantinya akan melalui dua tahapan yaitu :

STIKOMP SURABAYA

1. Sebuah dokumen XML yang telah valid (dalam bentuk dokumen XSD skema) yang telah menyertakan semua properti dari obyek yang telah terserialisasi.
2. Dari XSD skema tersebut akan di encoding menjadi dokumen XML yang mampu dibaca dengan class *reader* ataupun *writer* dalam lingkup Visual Studio.NET.

Untuk melakukan transfer data antara suatu obyek dan XML memerlukan suatu pemetaan dari bahasa pemrograman dibentuk menjadi skema XML dan *vice versa*. Class XmlSerializer, dan *tools* seperti Xsd.exe, mampu menjembatani antara kedua teknologi ini pada saaat desain maupun pada saat implementasi. Pada saat desain, digunakan Xsd.exe untuk membuat suatu dokumen skema XML (.xsd) dari suatu class, atau untuk membentuk suatu class dari skema yang diberikan. Pada kasus yang sama, class tersebut diberi suatu atribut khusus untuk menginstruksikan XmlSerializer bagaimana untuk memetakan sistem skema XML ke runtime bahasa pemrograman tertentu. Pada saat program dijalankan, instan dari suatu class dapat dirubah menjadi dokumen XML yang mengikuti skema yang diberikan.

1.4 Kompresi LZW

1.4.1 Dasar LZW

Kompresi LZW adalah implementasi dari algoritma LZ78. Dicetuskan pertama kali oleh Jacob Ziv dan Abraham Lempel pada tahun 1977 yang disebut LZ77, kemudian direkayasa ulang pada tahun 1987 (LZ78). Algoritma kompresi ini menggunakan metode *dictionary-based* untuk mendapatkan rasio kompresi yang besar. Kemudian dikembangkan lagi oleh Terry Welch menjadi LZW.

STIKOM SURABAYA

Algoritmanya secara mudah adalah menggantikan kumpulan karakter menjadi satu kode, tanpa melakukan analisa teks yang akan dibaca, namun hanya menambahkan setiap kumpulan karakter yang terbaca ke dalam suatu tabel. Kompresi dilakukan saat hanya satu kode digunakan untuk menggantikan suatu kumpulan karakter.

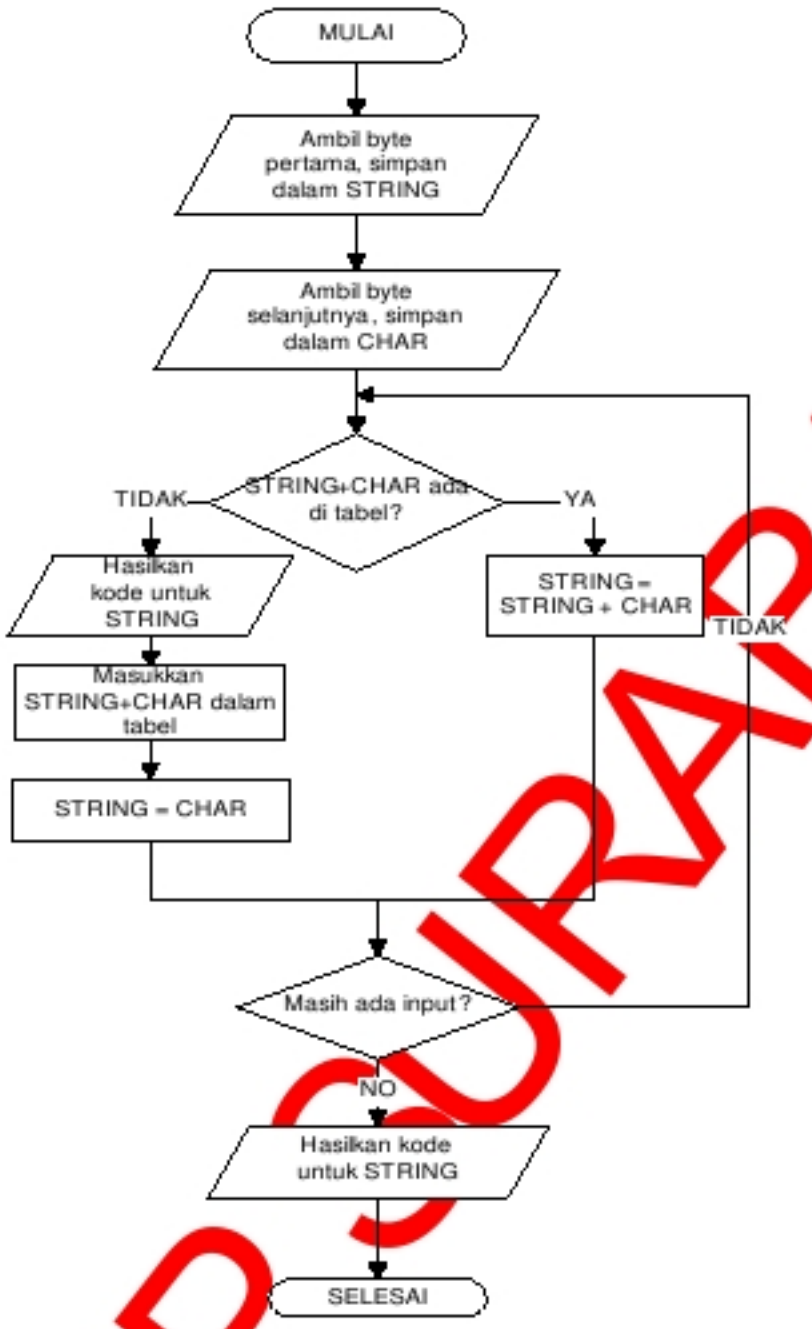
Kode dari output algoritma LZW bisa bermacam-macam *bit* panjangnya, namun harus memiliki bit yang lebih panjang dari satu karakter. Secara *default* 256 kode pertama (saat menggunakan karakter delapan bit) mewakili 256 karakter standar. Kode sisanya mewakili kumpulan *string* yang dimasukkan pada tabel selama algoritma itu berjalan.

Menurut Nelson (1992:14) kompresi LZW merupakan salah satu kompresi dengan jenis *Loseless Compression* yang menjamin akan menghasilkan duplikat yang sama persis dengan file aslinya. Sedangkan menurut Zeech (2003:13) algoritma kompresi LZW adalah yang paling umum digunakan. Hal ini yang menjadi alasan penggunaan algoritma kompresi LZW dalam tugas akhir ini.

1.4.2 Algoritma Kompresi LZW

Algoritma kompresi LZW ditunjukkan pada Gambar 2.1. Terlihat dari algoritma dibawah bahwa LZW selalu mengeluarkan kode output untuk string yang pernah dibaca. Dan setiap kali kode baru adalah output, string baru ditambahkan pada tabel string.

STIKOM SURABAYA



Gambar 2.1. Flowchart kompresi LZW

1. Mengambil byte pertama dari string dimasukkan dalam STRING
2. Mengambil byte selanjutnya dan dimasukkan dalam CHAR
3. Apakah STRING + CHAR ada dalam tabel?
4. Jika ya, STRING=STRING+CHAR, menuju (8)
5. Jika Tidak, Output kode dari STRING
6. Masukkan STRING+CHAR dalam tabel
7. STRING=CHAR
8. Apakah masih ada byte dalam string input?
9. jika ya kembali ke (2)

STIKOM SURABAYA

10. jika tidak maka selesai.

Contoh string yang digunakan untuk mencoba algoritma ini ditunjukkan pada Tabel 2.1. Input string adalah daftar kata-kata bahasa inggris dipisahkan dengan karakter '/'. Pada awal algoritma ini terlihat bahwa string yang pertama tidak melalui perulangan. Pengecekan dilakukan untuk mengetahui apakah string "/W" ada pada tabel. Karena tidak ada, output adalah kode '/', dan string "/W" ditambahkan pada tabel. Karena 256 karakter standar telah diwakili oleh kode 0-255 dalam tabel, maka string pertama yang dimasukkan tabel diwakili oleh kode 256. setelah huruf ketiga, "E" terbaca, kode string kedua , "WE" ditambahkan pada tabel dan kode output "W". Selanjutnya terus berlangsung sampai kata kedua, karakter "/" dan "W" terbaca, sesuai dengan string kode 256, sehingga output adalah kode 256, dan tiga karakter dimasukkan pada tabel. Proses terus berlangsung sampai teks input terbaca semua dan semua kode telah lengkap.

Tabel 2.1. Proses kompresi

Input String = /WED/WE/WEE/WEB/WET			
Character Input	Code Output	New code value	New String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE

STIKOMP SURABAYA

/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

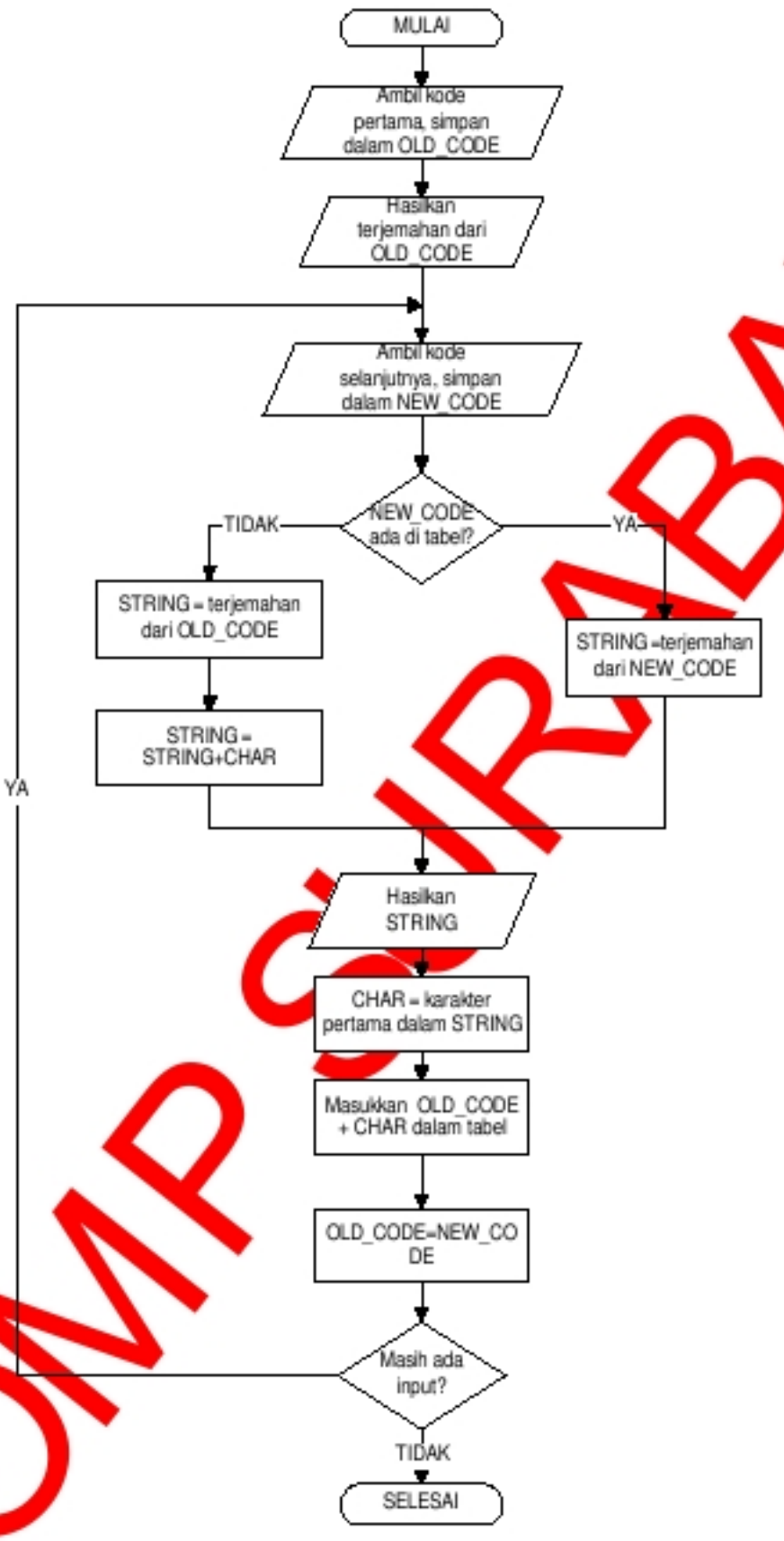
Sumber : http://marknelson.us/1989/10/01/lzw-data-compression/

Contoh output dari string ditampilkan pada Tabel 2.1, bersama tabel string yang dihasilkan. Seperti yang terlihat, tabel string bertambah dengan cepat, karena string baru ditambahkan pada tabel setiap kali kode baru adalah output. Pada input ini, 5 kode pengganti sebagai output, bersama dengan 7 karakter. Jika digunakan kode 9 bit untuk output, 19 karakter input akan berkurang menjadi 13.5 byte output.

1.4.3 Algoritma Dekompresi LZW

Algoritma dekompresi harus mampu mengambil stream dari kode output hasil dari algoritma kompresi, dan menggunakannya untuk membentuk kembali input stream yang benar-benar sama. Salah satu alasan efisiensi dari algoritma LZW adalah tidak perlu adanya pemindahan tabel string dari proses kompresi ke proses dekompresi. Tabel string dapat dibentuk sama persis seperti yang terbentuk pada proses kompresi, menggunakan input stream sebagai data. Hal ini dimungkinkan karena algoritma kompresi selalu mengeluarkan komponen STRING dan CHARACTER sebelum digunakan dalam output stream. Hal ini berarti bahwa data yang terkompresi tidak dibebani dengan tabel string yang besar.

STIKOMP SURABAYA



Gambar 2.2. Flowchart algoritma kompresi LZW

1. Menerima input kode pertama, disimpan dalam variabel OLD_CODE
2. Output translasi dari OLD_CODE
3. Menerima input kode selanjutnya, disimpan dalam NEW_CODE
4. Apakah NEW_CODE ada dalam tabel?

STIKOM SURABAYA

5. Jika ya, STRING=translasi dari NEW_CODE, menuju (8)
6. Jika tidak, STRING=translasi dari OLD_CODE
7. STRING=STRING+CHAR
8. Output string
9. CHAR=karakter pertama di STRING
10. Masukkan OLD_CODE+CHAR dalam tabel
11. OLD_CODE=NEW_CODE
12. Apakah masih ada code untuk input?
13. Jika ya, kembali ke (3)
14. Jika tidak, maka selesai.

Algoritma dekompresi ditunjukkan pada Gambar 2.2. Sama seperti algoritma kompresi, algoritma ini menambahkan string ke dalam tabel setiap kali membaca kode baru. Yang perlu dilakukan sebagai tambahan hanyalah menterjemahkan setiap kali membaca inputan berupa kode dan mengirikan menjadi output.

Tabel 2.2. menunjukkan output dari algoritma dengan inputan file kompresi yang didapat dari algoritma kompresi diatas. Yang perlu diperhatikan adalah bahwa hasil akhir dari tabel string yang terbentuk sama persis dengan tabel yang tebentuk pada algoritma kompresi. String hasil dekompresi juga sama persis dengan string inputan pada proses kompresi. Perhatikan juga bahwa 256 kode awal juga telah didefinisikan untuk menterjemahkan satu karakter, sama seperti pada algoritma kompresi.

STIKOMP SURABAYA

Tabel 2.2. Proses dekompresi

Input Codes : / W E D 256 E 260 261 257 B 260 T

Input/ NEW_CODE	OLD_CODE	STRING / Output	CHARACTER	New Table Entry
/	/	/		
W	/	W	W	256=/W
E	W	E	E	257=WE
D	E	D	D	258=ED
256	D	/W	/	259=D/
E	256	E	E	260=/WE
260	E	/WE	/	261=E/
261	260	E/	E	262=/WEE
257	261	WE	W	263=E/W
B	257	B	B	264=WEB
260	B	/WE	/	265=B/
T	260	T	T	266=/WET

Sumber : http://marknelson.us/1989/10/01/lzw-data-compression/

STIKOM SURABAYA