XML, Implementasi Struktur Data Pohon dalam Bentuk Berkas untuk Web

Inas Luthfi [1)] **– NIM 13506019**

1) Jurusan Teknik Informatika ITB, Bandung Indonesia 40132 , email: if16019@students.if.itb.ac.id

Abstrak – *Makalah ini akan memperkenalkan XML (eXtensible Markup Language), sebuah aturan pembangun representasi data dalam bentuk berkas yang dibuat sebagai cara fleksibel untuk berbagi data lintas aplikasi khususnya dalam dunia web. Data didalam berkas XML diformat sebagai struktur bersarang (nested structure), diciptakan untuk mudah dimengerti oleh manusia, dan menerapkan pola hierarki. Karena XML menerapkan pola hierarki, representasi pohon merupakan representasi yang paling tepat dalam penggambarannya. Makalah ini menjabarkan hubungan antara XML dengan struktur data pohon.*

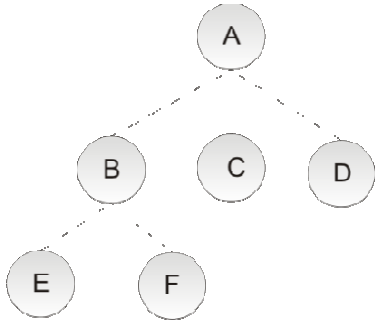
Sebuah struktur data pohon memerlukan metode untuk melakukan manipulasi terhadap elemen-elemennya, baik itu menambahkan simpul, menghapus simpul, maupun merubah isi simpul. Metode untuk memanipulasi pohon didalam XML adalah dengan memanfaatkan DOM (Domain Object Model). DOM sendiri adalah standar API (Application Programmable Interface) XML untuk memanipulasi elemen dalam sebuah berkas XML. Aplikasi XML dalam web yang paling utama adalah XHTML (eXtensible Hyper Text Markup Language). Sebagai turunan dari XML, XHTML sendiri menerapkan pola hierarki dalam menyimpan datanya dan dapat dianggap sebagai struktur data pohon.

Kata Kunci*: XML, struktur bersarang, pohon, markup, hierarki, XHTML, DOM*

1. PENDAHULUAN

1.1. Struktur data pohon

Struktur data pohon adalah struktur data dasar yang sering digunakan didalam Ilmu komputer. Pohon adalah graf khusus. Definisi formalnya adalah *graf tak-berarah terhubung yang tidak mengandung sirkuit* [1]. Pohon memiliki satu buah simpul khusus yang dinamakan **akar**. Tiap simpul pohon bisa mempunyai **anak simpul (child)**, bisa juga tanpa anak simpul, yaitu simpul yang berada dibawah pohon itu sendiri yang disebut **daun (leaf)**. Struktur data ini digambarkan kebawah, bukan ke atas seperti pohon yang sebenarnya. Simpul yang mempunyai anak simpul disebut simpul **orangtua (parent)**. Tiap simpul maksimal hanya mempunyai satu simpul orang tua. Setiap simpul di pohon memiliki jalur unik yang bisa ditelusuri mulai dari akar pohon tersebut. Simpul anak dari akar pohon dapat diliat sebagai subpohon (berarti pohon juga), oleh karena itu struktur data pohon secara alami dibangun berdasarkan algoritma rekursif.



Gambar 1: Penggambaran struktur data pohon sederhana

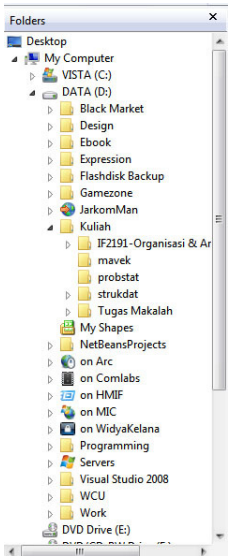
Hal yang penting kaitannya antara pohon dengan dunia pemrograman adalah 1) Pembangunan pohon dan 2) Manipulasi pohon. Pembangunan pohon harus ditentukan dulu fungsi pohon itu kedepannya, dan terdapat banyak algoritma untuk membangun pohon yang dibedakan berdasarkan fungsinya. Manipulasi pohon lebih ditekankan pada penambahan daun atau simpul, panghapusan suatu akar, pengubahan isi suatu simpul pohon, dan lain lain.

Manipulasi pohon menerapkan metode melangkah ke tiap-tiap elemen dari pohon, yaitu dari hubungan antara orangtua dan anak, disebut 'berjalan di pohon' (*walking the tree*). Operasi ini dilakukan dari suatu simpul lalu 'melangkah' menuju ke simpul yang lain, jika langkah terhenti di suatu simpul yang tidak memiliki anak, berarti simpul tersebut adalah daun. Ada beberapa cara melakukan *walking the tree*, yaitu dengan **preorder**, **in-order**, **post-order**. **Pre-order** adalah berjalan dengan melewati simpul orangtua lebih dahulu, lalu baru ke simpul-simpul anaknya. **Inorder** adalah berjalan dengan melewati simpul anak pertama terlebih dahulu, lalu ke simpul orang tua, lalu kembali ke simpul-simpul anaknya yang tersisa. **Post-order** adalah berjalan dengan melewati simpul-simpul anaknya terlebih dahulu, lalu baru ke simpul orangtuanya.

1.2.Pohon dan hierarki

Struktur hierarki adalah struktur penempatan suatu elemen di posisi tertentu yang dibedakan berdasarkan peringkat. Posisi yang ditempati oleh sebuah elemen adalah unik. Struktur data pohon sesuai untuk

merepresentasikan pola hierarki dimana setiap elemen hierarki memiliki jalur khusus yang unik untuk mencapainya dibandingkan dengan elemen yang lain. Sebagai contoh adalah daftar folder-folder dari suatu sistem operasi dapat dilihat dalam sudut pandang pohon. Untuk mengimplementasikan hierarki ini, struktur data pohon dalam berbagai macam bentuk sering digunakan, bentuk yang umum adalah pohon n-er (n-ary) dimana suatu akar memiliki jumlah anak yang tidak tetap (dengan kata lain akar pohon bebas memiliki berapapun jumlah anak).

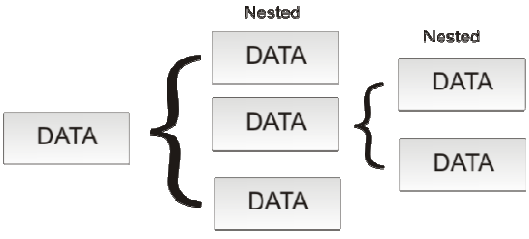


Gambar 2: Hierarki file dari folder di Windows *(tree view)*

Oleh karena itu struktur hierarki yang umum secara tidak langsung dapat dianggap sebagai sebuah pohon.

1.3. Hierarki , struktur bersarang (*nested structure*) dalam berkas teks, dan *markup*

Untuk menerapkan hierarki dalam berkas teks, kita dapat menempatkan data dalam bentuk struktur bersarang. Struktur bersarang menempatkan data dalam sebuah data yang lain secara bersarang, sehingga untuk mengakses data, kita harus melewati data yang lain tempat data yang kita cari bersarang.



Gambar 3: Penggambaran struktur bersarang

Dalam berkas teks, masalah yang timbul adalah bagaimana cara kita untuk memberikan keterangan bahwa suatu data menampung data yang lain atau dengan kata lain memberikan keterangan suatu data bersarang pada data lain? Suatu *markup* harus ditambahkan dalam berkas tersebut. *Markup* adalah informasi yang ditambahkan kedalam sebuah dokumen yang menambah arti dokumen tersebut dalam berbagai cara, sehingga bagian-bagian dari dokumen tersebut dapat dipartisi dan dibedakan antara satu bagian dan bagian yang lain, serta dapat jelas terlihat hubugan antar partisi. Seperti ketika kita membaca suratkabar, kita dapat membedakan tiap bagian teks dalam suratkabar dari penggunaan spasi dan penggunaan *font* yang berbeda. *Markup* melakukan partisi dokumen dengan menambahkan simbol-simbol. Bahasa *markup* adalah kumpulan simbol-simbol untuk memberikan label pembagi dokumen dalam bagian-bagian tertentu [2]. Markup penting bagi berkas teks karena berkas teks tersebut akan diproses oleh program komputer. Jika suatu berkas tidak memiliki label atau partisi, maka, program komputer tidak akan mengetahui cara untuk memisahkannya.

Markup memiliki beberapa standar penempatan dan penulisan simbol-simbolnya. Agar simbol-simbol yang digunakan dalam dokumen dapat bermakna, tentu saja ada aturan-aturan yang harus diikuti. Untuk itulah XML *(eXtensible Markup Language)* diciptakan.

1.4. XML

Sejak program komputer diciptakan, mulai muncul permasalahan ketika sebuah program komputer ingin bertukar data dengan program komputer yang lain. Belum adanya standar yang sesuai untuk memproses data yang diberikan, membuat program yang dibuat oleh pengembang yang satu sulit berkomunikasi dengan program dari pengembang yang lain. Diantara pemrogram memiliki cara yang berbeda untuk memaknai data hasil dari proses programnya. Jika data yang dihasilkan adalah berkas teks, maka cara mempartisi bagian berkas teks tersebut akan berbeda-beda. XML adalah standar yang diberikan oleh W3C (*World Wide Web Consorsium*) berupa aturan pembangun representasi data dalam bentuk berkas yang dibuat sebagai cara fleksibel untuk berbagi data lintas aplikasi khususnya dalam dunia web [2]. Dengan XML, penulisan markup telah distandarisasi, dan dapat menghasilkan sebuah berkas teks berhierarki dan strukturnya bersarang. Berikut ini adalah contoh potongan dari berkas teks XML:

```
<message>
        <exclamation>
                Hello, world!
        </exclamation>
        <paragraph>XML is
                <emphasis>fun</emphasis> and
```

```
            <emphasis>easy</emphasis> to use.
            <graphic
            fileref="smiley_face.pict"/>
        </paragraph>
</message>
```

Sekilas langsung terlihat struktur bersarang yang ditampilkan dalam berkas teks XML. Sebuah tag *markup* seperti <message> dapat dianggap oleh program sebagai bagian awal dari pesan, begitu pula tag <paragraph> adalah *markup* untuk memulai sebuah paragraph.

Dalam contoh, muncul sebuah pola yang merupakan cirri khas dari sebuah berkas teks XML. Sebuah tag dapat merupakan pembatas sebuah daerah (*region)* atau dapat pula tempat data teks dimulai. Contoh potongan XML diatas meskipun hanya sedikit, namun mengandung informasi yang banyak [2]. Contohnya adalah informasi 1) Pembatas (*Boundary*), tag markup membatasi bagian dari teks. Sebuah markup memulai bagian teks (data) dan tag pasangannya mengakhiri teks tersebut. Hal ini disebut sebagai labelisasi dan merupakan cara untuk menerapkan struktur bersarang. 2) Peran (*Role*) , setelah teks dipartisi, sebuah tag *markup* memberikan informasi tambahan untuk teks tersebut. Peran suatu bagian teks tersebut dapat diterangkan melalui label tag markup tersebut. Misalnya, <paragraph> *teks* </paragraph>, suatu program dapat memproses teks yang berada didalam tag tersebut sebagai sebuah paragraph baru dari dokumen.

Tag *markup* dan isi dari sebuah dokumen XML bersama-sama berkontribusi terhadap nilai informasi dari sebuah berkas teks XML. Tag *markup* penting untuk memisahkan bagian dokumen dan isi diantara *markup* tersebut (teks biasa) adalah hal yang tepenting bagi pembaca berkas, namun isi tersebut harus dipresentasikan secara lebih baik kepada pembaca. XML membantu program komputer untuk memformat isi dari suatu dokumen agar lebih komprehensif bagi manusia.

XML sendiri sebenarnya tidak membatasi tag-tag dalam dokumen XML, XML lebih berperan sebagai pembuat spesifikasi tag-tag markup bagi bahasa markup yang lain. Aplikasi XML yang terkenal di dunia web adalah XHTML, dimana tag-tag markup yang didefinisikan didalamnya berasal dari HTML (ada perbedaan yang signifikan dengan HTML). Selain XHTML terdapat aplikasi XML lain yang menjadi standar dalam pemrosesan dokumen di bidang ilmu tertentu seperti Mathematic Markup Language (MathML) untuk notasi matematika, atau Scalable Vector Graphics (SVG) untuk pembuatan grafik [4]

Contoh dibawah ini,

```
<?xml version="1.0"?>
<math xmlns="http://www.w3.org/TR/REC-
MathML/">
        <mi>F</mi>
        <mo>=</mo>
        <mi>G</mi>
        <mo>&InvisibleTimes;</mo>
        <mfrac>
        <mrow>
        <mi>M</mi>
        <mo>&InvisibleTimes;</mo>
        <mi>m</mi>
        </mrow>
        <apply>
        <power/>
        <mi>r</mi>
        <mn>2</mn>
        </power>
        </apply>
        </mfrac>
</math>
```

Dokumen MathML diatas merepresentasikan Hukum Newton, *Law of Gravitation*: $F = GMm / r^2$.

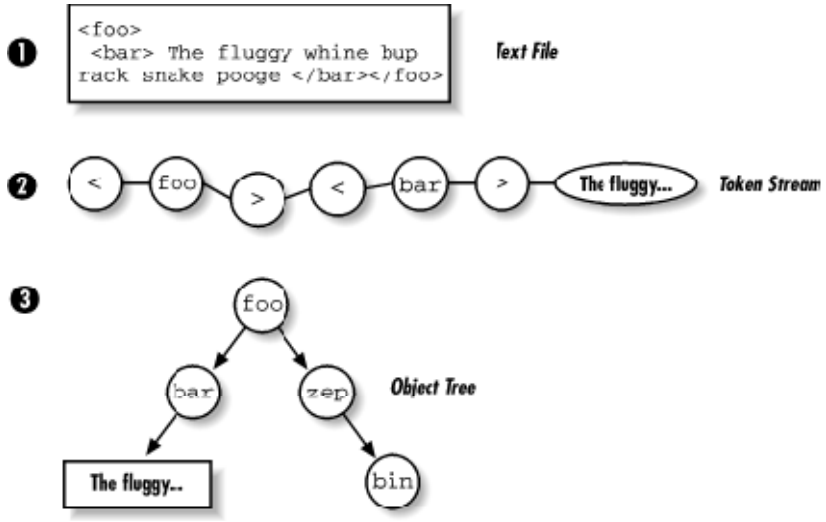
Bahkan jika perlu, anda bisa mendalami XML lebih lanjut dan membuat aplikasi XML lainnya melalui *Document Modeling*, membuat spesifikasi standar yang baru untuk mempermudah berbagi data melalui berkas teks.

2. STRUKTUR DATA POHON DALAM XML

Seperti telah dijelaskan dibagian sebelumnya, bahwa XML mengimplementasikan struktur bersarang dan pola hierarki, maka tidak aneh jika pemrosesan berkas teks XML akan mempergunakan struktur data pohon untuk mempartisi data dan mengambil informasi dari sebuah berkas teks XML.

2.1. *Parser,* pemroses berkas teks XML

Pemrosesan XML adalah proses ketika sebuah software membaca berkas teks XML dan melakukan sesuatu dengan berkas teks tersebut. Pemrosesan XML yang paling dasar adalah membaca berkas teks XML dan mengkonversinya menjadi representasi struktur data internal komputer agar aplikasi yang lain dapat mengaksesnya. Pemrosesan ini disebut dengan *parser* dan *parser* adalah bagian penting dari semua aplikasi yang menggunakan XML. *Parser* merubah rangkaian karakter dari berkas teks menjadi komponen kecil informasi bermakna yang disebut *token*. *Token* akan diubah menjadi struktur data dalam memori yang direpresentasikan sebagai pohon.



Gambar 4: Tiga langkah parser merubah ke struktur data pohon

Parser untuk memproses XML sangat ketat dalam pengecekan berkas XML. Jika berkas teks XML yang diberikan kekurangan sebuah tag *markup*. *Parser* akan menolak berkas teks XML tersebut untuk diproses lebih lanjut.

2.2. Anatomi XML

XML merupakan struktur bersarang, sehingga untuk merubah menuju struktur data pohon, diperlukan pembagian yang jelas dari berkas teks agar dapat diproses. Hal ini telah dibantu dengan adanya tag-tag markup.[3]

Semisal terdapat berkas teks XML seperti berikut ini:

```
<?xml version="1.0"?>
<time-o-gram pri="important">
        <to>Sarah</to>
        <subject>Reminder</subject>
        <message>Don't forget to recharge K-9
        <emphasis>twice a day</emphasis>.
        Also, I think we should have his
        bearings checked out. See you soon
        (or late). I have a date with
        some <villain>Daleks</villain>...
        </message>
        <from>The Doctor</from>
</time-o-gram>
```

Untuk proses pemilahan data, *markup* akan membantu membantuk partisi. Partisi tersebut bertujuan untuk mempermudah proses selanjutnya menjadi pohon. Proses partisi akan terjadi seperti berikut ini:

```
<?xml version="1.0"?>
<time-o-gram pri='important">
 <to>Sarah</to>
 <subject>Reminder</subject>
 <message>Don't forget to recharge K-9
  <emphasis>twice a day</emphasis>.
  Also, I think we should have his
  bearings checked out. See you soon
  (or late). I have a date with some
  <villain>Daleks</villain>...
 </message>
 <from>The Doctor</from>
</time-o-gram>
```

Gambar 5: XML setelah dipartisi

Selanjutnya *Parser* akan merubah partisi demi partisi berkas teks XML menjadi seperti gambar (6).

2.3. Pohon XML (XML tree)

Partisi teks (elemen) yang membagi berkas teks XML setelah diproses menjadi struktur pohon, dapat digambarkan sesuai diagram dalam gambar (6).

Gambar(6) merupakan pohon dimana kotak berwarna hitam dalam gambar tersebut merupakan simpul-simpul pembangun pohon. Dalam berkas teks XML hanya boleh terdapat satu simpul teratas (dalam contoh adalah <time-o-gram>) yang disebut sebagai akar. Sedangkan simpul yang tidak memiliki anak lagi disebut dengan daun berisi informasi atau data yang sebenarnya.

Dalam diagram tampak adanya kotak yang berwarna abu-abu (pri="important"). Isi dari elemen tersebut merupakan isi khusus dari tag sebagai informasi tambahan untuk tag orangtua dari elemen tersebut.

Untuk melihat bentuk pohon secara lebih jelas dari

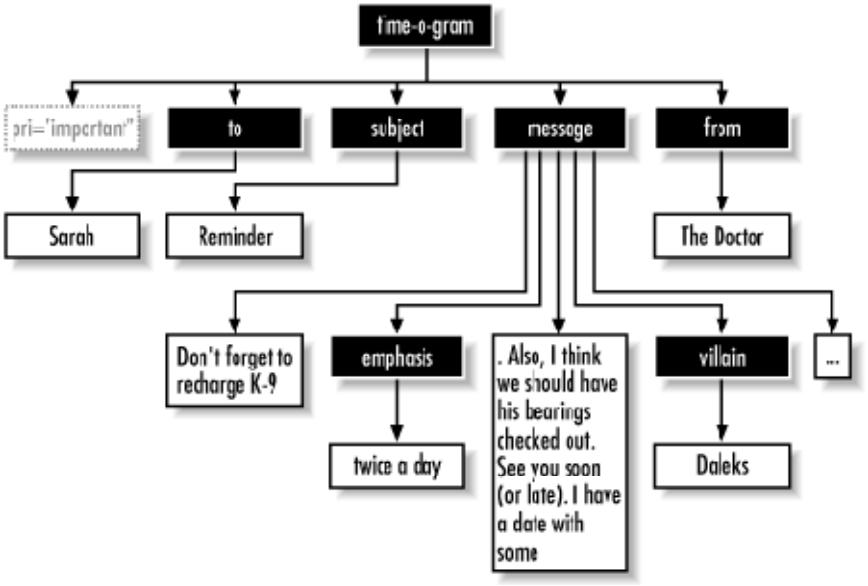
sebuah berkas teks XML, kita dapat membagi diagram dari gambar (6) menjadi subpohon seperti yang terlihat dalam gambar (7). Didalam Gambar (7) terdapat subpohon - subpohon yang membangun struktur data pohon. Karena didalam pohon, setiap simpul dapat dianggap sebagai akar dari subpohon, maka pemrosesan XML akan lebih mudah jika menganggap sebuah dokumen XML terdiri dari subpohon-subpohon, baru kemudian menggabungkan hasilnya menjadi sebuah pohon tunggal yang merupakan hasil akhir *parser.*

3. MANIPULASI POHON DENGAN DOM

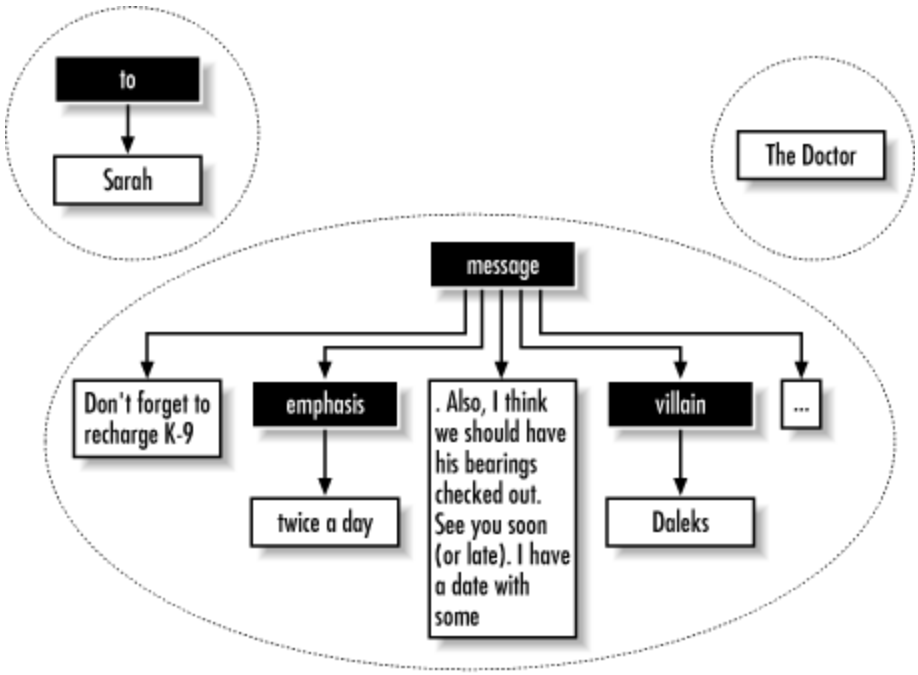
3.1. *Interface* DOM

Untuk memanipulasi struktur data pohon yang dihasilkan melalui *parser* diperlukan sebuah *interface* standar bagi pemrogram untuk menambah simpul, menghapus simpul, maupun mengubah isi sebuah simpul (daun) dari pohon XML.

W3C membuat sebuah standar terbuka untuk meangakses berkas teks XML dari program lain, standar tersebut bernama DOM *(Domain Object Model).*



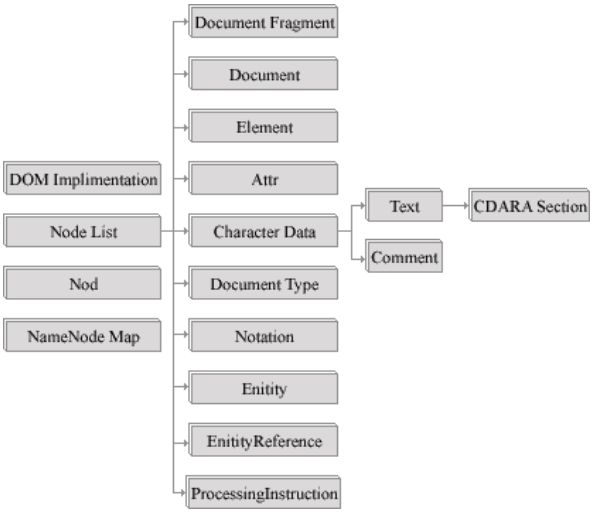
Gambar 6: *tree view* dari XML setelah dipartisi



Gambar 7: beberapa *subtree view* dari XML setelah dipartisi (subpohon)

W3C menspesifikasikan bahwa DOM adalah sebuah bahasa yang netral terhadap *platform*-nya atau yang lebih umum disebut *interface*. *Interface* didefinisikan untuk objek-objek XML tapi tidak ada implementasi yang spesifik, sehingga setiap bahasa pemrograman dapat mengimplementasikannya, seperti Java, C++, maupun bahasa scripting yang *embedded* dalam browser (aplikasi penjelajah web) seperti Javascript.

Struktur dokumen XML dapat direpresentasikan sebagai sebuah simpul pohon yang dapat menampilkan semua elemen dan relasi mereka satu dengan yang lainnya[5]. Dengan DOM, setiap bagian dari XML dianggap sebagai sebuah simpul, baik elemen maupun atribut XML tersebut. DOM adalah cara akses dokumen XML karena dengan DOM, berkas teks XML dapat dirubah menjadi representasi yang lebih abstrak sebagai kumpulan simpul dari struktur data pohon. Otomatis, pemrogram tidak perlu lagi khawatir tentang tatacara penulisan dari sebuah berkas teks XML, karena informasi dan logika yang ada sudah dapat langsung diakses melalui DOM.



Gambar 8: Spesifikasi W3C untuk DOM XML level 1

3.2. XHTML sebagai turunan XML

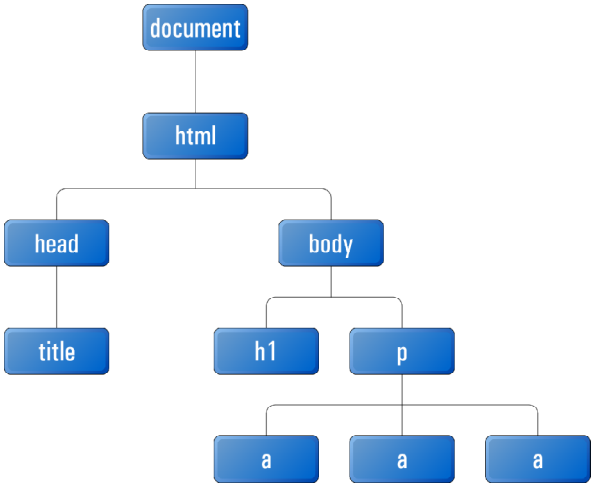
Dalam dunia web, aplikasi XML yang paling sering digunakan saat ini adalah XHTML. XML digunakan untuk menspesifikasikan tag-tag *markup* apa saja yang diperbolehkan untuk muncul di dokumen XHTML. Sekarang browser modern seperti Internet Explorer 7, Mozilla Firefox, dan Opera sudah mendukung halaman web dalam format XHTML. Tentu saja karena XHTML adalah salah satu aplikasi turunan dari XML, kita dapat menggunakan DOM untuk memodifikasi elemen-elemen XHTML didalamnya.

Browser modern sudah menyediakan fasilitas untuk memanfaatkan DOM XHTML, dengan cara memanfaatkan bahasa *scripting* Javascript. Halaman web yang dinamis seperti Yahoo! dan Gmail, selain memanfaatkan pemrosesan server dan database , sebenarnya juga memanfaatkan DOM untuk merubah bentuk halaman XHTML setiap kali ada *event* khusus yang terjadi dalam web, seperti menekan tombol link. Manipulasi DOM dalam XHTML memungkinkan pengunjung web berinteraksi dengan sebuah dokumen web [7].

Semisal kita memiliki berkas XHTML seperti berikut:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
lang="en-US">
<head>
        <title>text… </title>
</head>
<body>
        <h1>
                Text…
        </h1>
        <p>
                Text…
                 <a>text.. </a>,
                <a >text.. </a>
                Text..
                <a>text.. </a>.
        </p>
</body>
</html>
```

Jika kita buka melewati browser web seperti firefox, maka browser tersebut akan langsung merubah halaman tersebut menjadi representasi internal struktur pohon dan dapat diakses melalui DOM.



Gambar 9: XHTML dalam bentuk pohon

3.3. DOM *Scripting* dengan Javascipt

Untuk memanipulasi struktur pohon dengan memanfaatkan DOM. Tentu saja terdapat metode-metode khusus yang dispesifikasikan di bahasa *scripting* Javascript.

Implementasi di bahasa pemrograman lain tentu saja berbeda namun aspek manipulasi pohon dalam DOM tidak akan berubah

3.3.1. Mengakses simpul tertentu

Javascript menyediakan beberapa fungsi DOM untuk mengakses simpul yang telah dibuat pada pohon.

Akar tertinggi pada DOM XHTML adalah document [6]&[7]

document.getElementsByTagName('tag')[num]
mengakses berdasarkan nama tag markup

document.getElementsById('id')
mengakses berdasarkn id, atribut suatu tag markup

3.3.2. Manipulasi isi simpul tertentu

Simpul yang telah diakses dapat dimanipulasi isinya. Lebih lengkap dapat melihat [6] & [7]

node.getAttribute('attribute')
mengambil informasi isi dari suatu atribut

node.setAttribute('attribute', 'value')
set isi atribut dengan isi yang baru

node.nodeType
membaca tipe simpul DOM yang diakses

node.nodeValue
membaca dan menset isi simpul yang bertipe teks

3.3.3. Mengakses anak, orangtua, saudara kandung dari suatu simpul

Setelah mendapatkan akses ke sebuah simpul dalam pohon, kita dapat mengakses simpul yang lain dari simpul awal. Lebih lengkapnya dapat melihat [7]. Implementasi DOM dalam mengakses simpul yang lain merupakan proses berjalan dalan pohon (*walking the tree)*

node.previousSibling
mengembalikan saudara kandung sebelumnya dari simpul yang diakses sekarang

node.nextSibling
mengembalikan saudara kandung sesudahnya dari simpul yang diakses sekarang

node.childNodes
mengembalikan semua anak dari sebuah simpul

node.parentNode
mengembalikan orangtua dari sebuah simpul

3.3.4. Menambah simpul dan menghapus simpul

Aspek terakhir dari manipulasi pohon adalah menambah simpul dan menghapus simpul. Dapat dilihat lebih lanjut di [7]**.**

document.createElement(string)
Membuat elemen DOM baru dengan nama string

document.createTextNode(string)
Membuat simpul teks baru dalam pohon

node.appendChild(newNode)
Menambahkan simpul baru sebagai simpul anak dari node

node.removeChild(oldNode)
Menghapus simpul dari pohon

4. KESIMPULAN

Untuk mempermudah komunikasi data antar aplikasi terutama dalam berkas teks, muncullah standar dari W3C yaitu XML yang dapat menghasilkan berkas teks yang menerapkan struktur bersarang dan hierarki. Untuk memproses berkas teks XML digunakan sebuah parser, dimana pemrosesan yang dilakukan oleh parses tidak lain adalah pembentukan struktur data pohon (*XML tree*). Untuk memanipulasi struktur data pohon yang dibentuk, muncullah standar dari W3C sendiri yaitu DOM, dimana pemrogram diberikan suatu *interface* untuk mengolah informasi yang terdapat dalam berkas teks XML.

Aplikasi Struktur data pohon kadang memang tidak disadari di kehidupan kita, berkas teks XML dan XHTML yang semula kita kira sebagai berkas teks biasa, ternyata dapat kita anggap sebagai struktur data pohon pula. Kemajuan Matematika Diskrit sebagai ilmu yang mendalami representasi pohon (graf khusus) secara langsung berdampak terhadap teknologi ini, karena inti dari pemrosesan XML itu sendiri tidak lain adalah penerapan manipulasi pohon.

DAFTAR REFERENSI

[1] Munir, Rinaldi, *Matematika Diskrit,* Program Studi Informatika, Institut Teknologi Bandung. 2007.

[2] Erik T. Ray, *Learning XML*, O'Reilly Media, 2001.

[3] Wayne Brookes, *Computing Theory With Relevance*, Faculty of Information Technology University of Technology, Sydney, 2007.

[4] Wikipedia 2007 http://en.wikipedia.org/wiki/XML Tanggal akses: 30 Desember 2007 pukul 13:00.

[5] Layout Galaxy http://www.layoutgalaxy.com/xml/object.php4 Tanggal akses: 30 Desember 2007 pukul 13:10.

[6] Kevin Yank, *Simply Javascript,* SitePoint Pty. Ltd, 2007.

[7] Christian Heilmann, *Beginning JavaScript with DOM Scripting and Ajax,* Apress, 2007