IMPLEMENTASI STRUKTUR POHON SEBAGAI KOMPONEN DI BERBAGAI *PLATFORM*

Doni Arzinal [1], Inggriani Liem [2]
Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha No.10, Bandung 40132
Telp. (022) 2508135, Faks. (022) 2500940
E-mail: if15109@students.if.itb.ac.id, inge@informatika.org

ABSTRAKS
Pada makalah ini kami memaparkan implementasi struktur pohon di berbagai platform dengan menganalisis beberapa model dan pengelolaan data berstruktur pohon dalam basis data relasional, serta penggunaannya pada beberapa aplikasi. Tujuan penelitian ini adalah memodelkan data berstruktur pohon baik secara konseptual, lojik dan fisik serta merancang beberapa komponen pengelolaan data berstruktur pohon untuk beberapa platform yang sedang populer saat ini agar pengembangan aplikasi dapat mudah dilakukan dengan menggunakan komponen yang sudah kami bangun, yang dapat di-reuse baik dari segi struktur data internal dan persisten, maupun software pengolahnya. Komponen sudah diuji secara independen dan dipakai untuk mengembangkan aplikasi herbarium maya dan aplikasi administrasi penduduk. Makalah ini ditulis berdasarkan penelitian yang merupakan Tugas Akhir di Program Studi Teknik Informatika ITB.

Kata Kunci: tree structure, hierarchical data, data model, database design, component base software development

1. PENDAHULUAN

Struktur pohon (*tree*) merupakan struktur yang penting dalam bidang informatika yang memungkinkan kita mengorganisasikan informasi berdasarkan struktur lojik. Struktur data pohon juga memungkinkan kita untuk mengakses suatu elemen dengan cara khusus. Struktur pohon banyak dipakai untuk menggambarkan data yang memiliki struktur seperti pohon (*tree-like structure*) atau ada juga yang menyebutkannya sebagai struktur yang hirarkis.

Struktur pohon banyak digunakan di berbagai area. Struktur organisasi suatu perusahaan disusun menyerupai struktur pohon. Sebagai contoh, seorang direktur di perusahaan membawahi beberapa wakil direktur, kemudian seorang wakil direktur juga akan membawahi beberapa kepala bagian, dan seorang kepala bagian juga akan membawahi beberapa orang karyawan. Contoh lainnya dapat kita temukan pada sistem pengklasifikasian (atau taksonomi) tanaman, di mana beberapa *family* akan terdiri dari beberapa *genus*, dan setiap *genus* juga akan terdiri dari beberapa *species*. Dalam sistem informasi kependudukan nasional atau sistem pemilihan umum, data provinsi, kabupaten, kecamatan sampai ke desa harus dimodelkan dalam struktur hirarkis di mana jumlah penduduk atau suara di suatu simpul akan merupakan penjumlahan suara dari semua simpul anaknya.

Namun, yang terjadi saat ini adalah beberapa orang yang dihadapkan dengan struktur seperti itu membangun model data dan komponen pengelolaan datanya secara sendiri-sendiri. Setiap orang membangun model data pohonnya sendiri sekaligus komponen pengelolaan datanya. Terlebih lagi, struktur pohon yang dikelola pada struktur data eksternal (contoh: basis data) tidak banyak dibahas di saat penggunaan basis data relasional yang sangat populer sekarang ini. Sedangkan kebutuhan akan model data berstruktur pohon beserta komponen pengelolaannya sangatlah tinggi karena secara natural, banyak struktur yang dikelola oleh sistem informasi adalah struktur pohon.

Oleh karena itu, pertanyaan yang ingin dijawab dalam penelitian ini adalah: (a) Bagaimana membangun model data pohon, baik secara konseptual, lojik, dan fisik yang dapat diterima secara umum? (b) Bagaimana membangun komponen pengelola data berstruktur pohon di berbagai *platform* berdasarkan model data yang sudah dibangun? (c) Bagaimana membangun aplikasi dengan menggunakan komponen tersebut untuk menunjukkan daya guna komponen?

Berangkat dari model konseptual pohon, pada penelitian ini kami menganalisis dan mengajukan beberapa alternatif model data berstruktur pohon dengan pendekatan pemodelan ER pada model konseptual dan lojik. Sedangkan untuk model fisik kami menawarkan model relasional yang merupakan hasil dari pemetaan model ER ke model relasional pada model lojik yang dihasilkan sebelumnya.

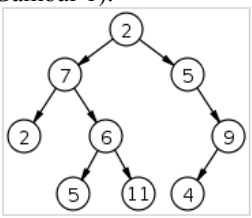
Model fisik yang berupa model relasional akan diimplementasikan ke berbagai RDBMS yang populer saat ini, seperti SQL, MySQL. Untuk komponen pengelolaannya, juga dibangun di berbagai *platform* yang populer, seperti .NET, dan PHP. Diharapkan ketersediaan komponen ini akan mempermudah pembangunan sistem informasi yang datanya mempunyai struktur pohon.

2. STUDI LITERATUR

2.1 Definisi Pohon

Suatu pohon merupakan sekumpulan simpul (*node*) yang saling terhubung satu sama lain dalam

kesatuan yang membentuk layaknya struktur sebuah pohon. Dalam teori graf, struktur ini merupakan graf *acyclic* di mana setiap simpul yang terhubung, memiliki atau tidak memiliki simpul anak (*children*), dan satu simpul ayah (*parent*). Simpul yang tidak memiliki simpul ayah dinamakan simpul akar (*root*). Dalam struktur pohon, hanya terdapat satu jalur (*path*) yang menghubungkan satu simpul ke simpul yang lain (lihat Gambar 1).



Gambar 1. Contoh struktur pohon

Definisi Formal: Dalam salah satu literatur yang cukup tua dan menjadi dasar dari banyak literatur. Knuth (1968) mendefinisikan pohon sebagai himpunan terhingga T dari satu atau lebih simpul (*node*) sedemikian sehingga: (a) Ada satu simpul yang ditunjuk khusus sebagai akar pohon, akar (T) dan (b) Sisa simpul kecuali akar yang dibagi menjadi m ≥ 0 himpunan *disjoint* Ti,…, Tm, dan masing-masing himpunan ini pada gilirannya akan membentuk pohon. Pohon-pohon Ti,…,Tm disebut sebagai sub pohon dari akar.

Definisi diatas adalah definisi rekursif karena struktur pohon merupakan bagian dari struktur pohon yang lain. Contoh: sebuah buku dipandang sebagai pohon. Judul buku adalah akar. Buku dibagi menjadi bab-bab. Masing-masing bab adalah sub pohon yang juga mengandung judul sebagai akar dari bab tersebut. Bab dibagi menjadi sub bab yang juga diberi judul. Sub bab adalah pohon dengan judul sub bab sebagai akar. Daftar isi buku dengan penulisan di-identasi mencerminkan struktur pohon dari buku tersebut.

2.2 Representasi Struktur Pohon

Secara umum, suatu pohon dapat direpresentasikan dalam dua jenis representasi, yaitu representasi lojik dan representasi fisik. Representasi lojik memperlihatkan semantik dari struktur pohon tersebut, sedangkan representasi fisik memperlihatkan bagaimana struktur pohon direalisasikan secara fisik oleh teknologi tertentu.

2.2.1 Representasi Lojik

Banyak cara untuk mepresentasikan struktur pohon secara lojik. Knuth (1968) merepresentasikan pohon secara lojik dalam diagram venn, notasi kurung, atau dengan notasi tingkat (*indentation*).

2.2.2 Representasi Fisik

Representasi pohon secara fisik terdiri dari dua bagian, yaitu representasi fisik internal dan representasi fisik eksternal.

Pada representasi internal, struktur pohon diimplementasi secara fisik dalam memori lokal. Pada kasus pohon biner, salah satu teknik merepresentasikan struktur pohon secara internal yaitu dengan menggunakan *linked list* secara rekursif. Gambar 2 adalah contoh implementasi pohon menjadi *linked list* dalam bahasa C.

```
/*Deklarasi struct*/
typedef struct Tree {
        int data;
        Tree *left;
        Tree *right;
}
/*Deklarasi variabel*/
Tree *pohon;
```

Gambar 2. Implementasi struktur pohon dengan *linked list*.

Representasi fisik eksternal juga disebut penyimpanan data persisten. Struktur pohon diimplementasi secara fisik dalam struktur eksternal (misalnya basis data atau XML). Berbeda dengan struktur internal yang banyak dibahas bahkan menjadi bahan wajib dari kuliah struktur data dasar, representasi struktur pohon secara eksternal dalam basis data belum banyak dibahas dan masih menjadi tantangan serius, terkait dengan pengelolaan data dan waktu respon. Sedangkan struktur data dalam sebuah XML pada dasarnya adalah pohon, oleh karena itu pembahasannya bukan pada struktur data pohon, melainkan langsung menjadi pembahasan XML.

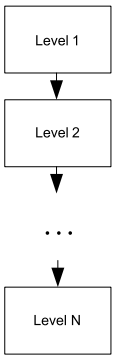
3. SPESIFIKASI KEBUTUHAN DAN PERANCANGAN MODEL DATA BERSTRUKTUR POHON

Spesifikasi kebutuhan data berstruktur pohon dihasilkan dari studi literatur yang dilakukan dan kebutuhan pengembangan aplikasi herbarium maya yang menjadi fokus penelitian kami. Namun, kami ingin membangun herbarium tersebut berdasarkan komponen pohon yang direncanakan dan akan digunakan ulang untuk aplikasi lain yang membutuhkan pengolahan data persisten berstruktur pohon. Beberapa spesifikasi tersebut adalah: (a) Mampu merepresentasikan beberapa atribut yang terdapat dalam struktur pohon, seperti: *node, root, children, parent, dan path*. (b) Mampu merepresentasikan data yang tersimpan dalam atribut tersebut. (c) Mampu merepresentasikan sebuah *node* yang terdiri dari *single entity object*, atau yang terdiri dari banyak (koleksi) *entity object*.

Dari spesifikasi model data tersebut, dilakukan proses pemodelan data dengan menggunakan pendekatan *generic data model*, yaitu dengan mengumpulkan beberapa model data sebelumnya dalam menyelesaikan persoalan pemodelan data terkait pemodelan data berstruktur pohon. Kemudian dilakukan beberapa modifikasi untuk menyesuaikan spesifikasi kebutuhan data.

3.1 Model Hirarkis

Model ini didasari susunan yang hirarkis. Suatu level tertentu pada struktur pohon menggambarkan entitas tertentu. Sehingga model ini digambarkan dengan entitas yang tersusun secara hirarkis.



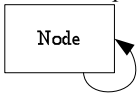
Gambar 3. Model Hirarkis

Sebagai gambaran, anggaplah struktur organisasi di perusahaan tersusun atas cabang (*branch*), departemen (*department*), dan seksi (*section*). Pada model hirarkis, maka struktur organisasi perusahaan ini akan digambarkan dengan entitas cabang (*branch*), departemen (*department*), dan seksi (*section*) yang berjenjang dari atas ke bawah. Cabang, departemen, seksi, dan seterusnya merupakan jenjang (level) di struktur pohon organisasi tersebut, sehingga model hirarkis ini dapat digambarkan seperti yang diperlihatkan oleh Gambar 3.

Model ini adalah model yang tidak begitu fleksibel untuk struktur pohon yang berkembang (*growing tree*) karena kita harus menetapkan dengan pasti berapa jumlah level yang ada di struktur tersebut, dan kita juga harus mengetahui urutan jenjang setiap level itu. Oleh karena itu, model ini tidak disarankan untuk digunakan pada struktur pohon yang berkembang.

3.2 Model Rekursif

Pada model rekursif, tidak dikenal pembagian level di struktur pohonnya. Setiap entitas pada tiap level digeneralisasi menjadi satu entitas. Sebagai contoh, suatu model organisasi perusahaan terdiri atas cabang (*branch*), departemen (*department*), dan seterusnya akan digeneralisasi menjadi unit-unit organisasi di mana unit-unit organisasi ini merepresentasikan suatu *node* di struktur pohon organisasi perusahaan tersebut. Sehingga model rekursif dapat digambarkan seperti yang diperlihatkan oleh Gambar 4. di mana relasi antar entitas merepresentasikan relasi *parent-child*.



Gambar 4. Model Rekursif

Model ini cocok untuk struktur pohon yang jumlah levelnya bersifat dinamis karena kita tidak perlu menambah entitas baru jika jumlah level di struktur tersebut bertambah.

4. SPESIFIKASI DAN PERANCANGAN KOMPONEN PENGELOLAAN DATA BERSTRUKTUR POHON

Sebuah komponen pengelolaan data berstruktur pohon, hendaknya mampu melakukan operasi-operasi dasar yang dilakukan untuk data berstruktur pohon, seperti menambah, meng-*update*, dan menghapus *node* serta memperlihatkan (*view*) pohon itu sendiri.

4.1 Add Node

Add Node adalah operasi untuk menambahkan *node* baru ke dalam struktur pohon yang telah didefinisikan. Dalam struktur organisasi perusahaan, operasi ini dapat dianalogikan dengan suatu penambahan departemen atau divisi baru.

4.2 Update Node

Update Node adalah operasi untuk meng-*update* informasi suatu *node*, baik itu informasi mengenai isi *node*, maupun informasi *parent* atau *children* dari *node* tersebut. Terdapat dua versi dari operasi ini: yang pertama adalah operasi *update* yang meng-*update parent*, sedangkan yang kedua adalah operasi yang meng-*update children*. Versi pertama memiliki pengecualian untuk *node* yang dianggap sebagai *root*, sedangkan versi kedua memiliki pengecualian untuk *node* yang dianggap sebagai *leaf*. Dalam struktur organisasi perusahaan, operasi ini dapat dianalogikan dengan suatu perubahan nama departemen atau suatu divisi yang mengganti departemen yang membawahinya.

4.3 Delete Node

Delete Node adalah operasi untuk menghapus suatu *node* dari struktur pohon. Dalam suatu struktur organisasi, operasi ini dapat dianalogikan dengan penutupan suatu departemen. Terdapat dua versi dari operasi ini: yang pertama adalah operasi *delete* yang menyertakan semua *node* yang berada di bawahnya, sedangkan yang kedua adalah operasi *delete* yang tidak menyertakan *node* yang dibawahnya, sehingga *children* dari *node* yang dihapus akan menjadi *orphan*. Pada versi pertama mungkin kita perlu menambahkan sub-operasi *getAllDescendant* untuk mendapatkan semua keturunan yang berasal dari *node* yang akan dihapus.

4.4 View Tree

View Tree adalah operasi untuk memperlihatkan (*enumerating*) sebagian atau keseluruhan *node* yang menyusun suatu struktur pohon. Seperti operasi *Delete Node* versi pertama, operasi ini juga dapat dilakukan dengan sub-operasi *getAllDescendant* untuk *node* yang dianggap *root*, karena *root* dan semua keturunannya merupakan kumpulan *node* yang menyusun struktur pohon.

5. IMPLEMENTASI KOMPONEN

Model yang dibangun sudah diimplementasi dan diuji untuk tiga *platform* yaitu SQL Server 2008, SQL Server 2005, serta PHP dan MySQL. SQL Server 2008 dipilih karena ternyata sudah menyediakan tipe primitif untuk pohon, pengelola data berstruktur pohon pada SQL Server 2005 sudah tersedia Crowley (2004) yang sangat menarik untuk dijadikan sebagai pembanding, sedangkan PHP dan MySQL merupakan *platform* yang masih memerlukan komponen yang kami bangun.

Dari ketiga komponen ini, model data yang dipergunakan adalah model rekursif. Seperti yang telah dijelaskan sebelumnya, model rekursif dipilih karena merupakan model yang fleksibel dan cocok untuk struktur pohon yang berkembang sehingga model ini lebih bersifat umum jika dibandingkan dengan model hirarkis dimana kita harus mengetahui dengan tepat jumlah level pada struktur pohon yang akan dimodelkan.

5.1 SQL Server 2008

SQL Server 2008 menyediakan tipe data khusus untuk mengelola data berstruktur pohon yang disebut dengan *hierarchyid*. Untuk membuat tabel dengan struktur hirarkis ataupun menunjuk ke data hirarkis di lokasi lain, kita dapat menggunakan tipe data ini. Untuk melakukan pengelolaan data hirarkis, tersedia *hierarchyid functions* yang dijelaskan pada literatur Microsoft (2009).

Pada SQL Server 2008, semua *node* yang ada di struktur pohon disimpan dalam satu tabel. Contoh: implementasi tabel dengan menggunakan tipe data *hierarchyid* dapat dilihat pada Gambar 5.

```
CREATE TABLE Node
   (
    NodeId hierarchyid PRIMARY KEY,
    NodeLevel AS NodeId.GetLevel(),
    NodeName nvarchar(50)
   ) ;
GO

CREATE INDEX Node_BreadthFirst ON
Node(NodeLevel, NodeId)
GO
```

Gambar 5. *Create* Pohon pada SQL Server 2008

Pengelolaan data berstruktur pohon untuk operasi *Add Node, Update Node, Delete Node*, dan *View Tree* dapat dilakukan sepenuhnya dengan memanfaatkan *hierarchyid functions* seperti yang dijelaskan dalam literatur Tegels (2008).

5.2 *Platform* .NET dan SQL Server 2005

Pada SQL Server 2005, tidak terdapat mekanisme khusus untuk menangani persoalan pengelolaan data berstruktur pohon. Dalam literatur Crowley (2004), pengelolaan data berstruktur pohon dilakukan dengan memakai model rekursif dimana terdapat atribut *depth* dan *lineage* yang merepresentasikan *level* dan *path* ke *root* dari suatu *node* dalam struktur pohon, seperti yang diperlihatkan oleh Gambar 6.

```
CREATE TABLE dfTree
   (
    id int PRIMARY KEY,
    parentId int,
    name nvarchar(50),
    depth int,
    lineage varchar(100)
   );
GO
```

Gambar 6. Struktur tabel merepresentasi pohon, SQL Server 2005. [Crowley (2004)]

Crowley (2004) mempergunakan *trigger* dan *stored procedure* untuk pengelolaan *Add Node, Update Node, Delete Node*, dan *View Tree* di level basis data. Untuk di level aplikasi, digunakan *Class* khusus yang merepresentasikan struktur pohon dalam struktur internal, mengeksekusi *stored procedure*, dan mengelola *interface*.

5.3 *Platform* PHP dan MySQL

Pada MySQL, belum ada literatur yang secara spesifik menjelaskan bagaimana mengelola data persiten berstruktur pohon. Oleh karena itu, dengan mengadaptasi implementasi komponen pada *platform* .NET dan SQL Server 2005, kami mencoba mengimplementasinya pada *platform* PHP dan MySQL.

Implementasi tabel dalam MySQL yang merepresentasikan struktur pohon dapat dilihat pada Gambar 7 yang merupakan penyesuaian dari *platform* SQL Server 2005 yaitu dengan menggunakan atribut *depth* dan *lineage* untuk merepresentasikan *level* dan *path* ke *root* dari suatu *node* .

```
CREATE TABLE IF NOT EXISTS `tree` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `parentId` int(11) DEFAULT NULL,
  `name` varchar(20) NOT NULL,
  `depth` int(11) DEFAULT NULL,
  `lineage` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
AUTO_INCREMENT=1 ;
```

Gambar 7. Representasi pohon pada MySQL

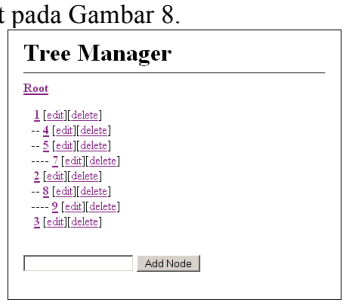
Untuk pengelolaan struktur data persisten tersebut, kami mengimplementasi sebuah TreeClass.php, yaitu sebuah kelas yang merepresentasikan pohon dalam struktur internal dan memiliki beberapa operasi untuk melakukan *Add Node*, *Update Node*, *Delete Node*, dan *View Tree*.

6. PENGUJIAN KOMPONEN DAN APLIKASI

Terdapat dua tahap pengujian dalam pengembangan perangkat lunak berbasis komponen. Tahap pertama adalah pengujian terhadap komponen itu sendiri. Pengujian komponen ini dilakukan untuk memastikan semua spesifikasi kebutuhan komponen telah terpenuhi atau tidak. Setelah semua spesifikasi kebutuhan komponen telah dipastikan terpenuhi, maka pengujian tahap pertama ini telah selesai dan dilanjutkan ke pengujian tahap kedua. Dari hasil pengujian tahap pertama yang kami lakukan,

 ISSN: 1907-5022

komponen yang kami bangun telah memenuhi semua spesifikasi kebutuhan. Untuk melakukan pengujian tahap pertama ini, kami membuat suatu *driver* yang berinteraksi dengan pengguna untuk melakukan *Add Node*, *Edit Node*, *Delete Node*, dan *View Tree* dimana *user-interface* dari *driver* ini dapat dilihat pada Gambar 8.

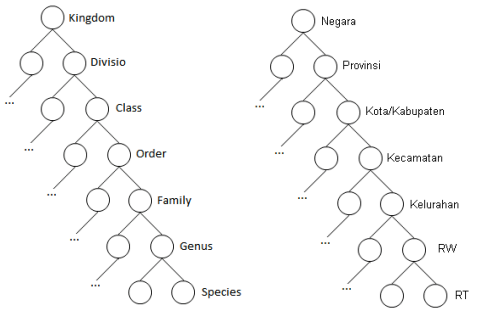


Gambar 8. *Driver* komponen pengelolaan struktur pohon

Pengujian tahap kedua dilakukan dengan mengintegrasikan komponen ke dalam suatu sistem. Pengujian tahap kedua ini dimaksudkan untuk melihat apakah komponen bekerja dengan baik setelah diintegrasikan ke dalam suatu sistem dan membandingkan *cost* pengembangan perangkat lunak berbasis komponen dengan pengembangan perangkat lunak yang tidak berbasis komponen.

Pada tahap ini, pengujian dilakukan pada dua sistem yang sedang kami kembangkan, yaitu herbarium maya dan SIP (Sistem Informasi Penduduk). Komponen yang kami kembangkan digunakan untuk mengelola struktur data pohon di dua sistem tersebut.

Struktur data pohon pada sistem herbarium dapat dilihat dari data taksonomi dan data penyebaran tanaman di wilayah geografis atau administratif yang disusun dengan struktur pohon (lihat Gambar 9). Sedangkan struktur data pohon pada SIP dapat dilihat dari pengelompokan alamat penduduk berdasarkan wilayah administrasi atau geografis tempat tinggalnya (lihat Gambar 9 sebelah kanan).



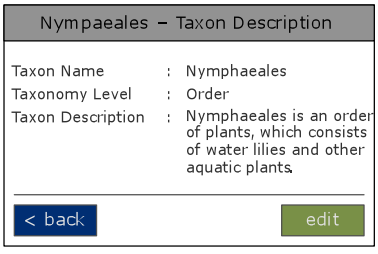
Gambar 9. Struktur ranking utama taksonomi tanaman (kiri) dan struktur pembagian wilayah administratif di Indonesia (kanan)

6.1 Sistem Herbarium Maya

Hasil pengujian yang dilakukan pada sistem herbarium maya memperlihatkan bahwa kode program untuk pengelolaan data taksonomi menjadi lebih ringkas sehingga waktu pengembangan sistem relatif menjadi lebih singkat. Hal ini dikarenakan kami tidak perlu membuat modul khusus untuk operasi-operasi seperti *Add Species*, *Add Genus*, *Edit Species* dst. karena hanya tinggal memanggil prosedur *Add Taxon,* dan *Edit Taxon* (*Taxon* merepresentasikan *node* pada pohon herbarium) pada komponen ini. Gambar 10 memperlihatkan *user-interface* dari komponen pengelolaan data berstruktur pohon yang diimplementasi pada sistem herbarium maya.

Untuk mendeskripsikan suatu *node* dalam pohon herbarium tersebut, kami menggunakan pendekatan konsep pemodelan berorientasi objek dan konsep pemodelan *metadata* seperti yang telah dijelaskan pada literatur Liem (2010) mengenai pemodelan data berstruktur banyak. Dengan pendekatan ini, suatu *node* dapat dideskripsikan melalui *metadata*-nya yang didapatkan dari *node* itu sendiri atau diturunkan melalui *node* para leluhurnya. Contoh: *node* Nympaeales dideskripsikan melalui *taxon name*, *taxonomy level*, dan *taxon description* yang didapat dari *metadata* leluhurnya yaitu Plantae.

Gambar 10. *User-interface* pengelolaan taksonomi di herbarium maya



Gambar 11. Halaman deskripsi takson pada herbarium maya

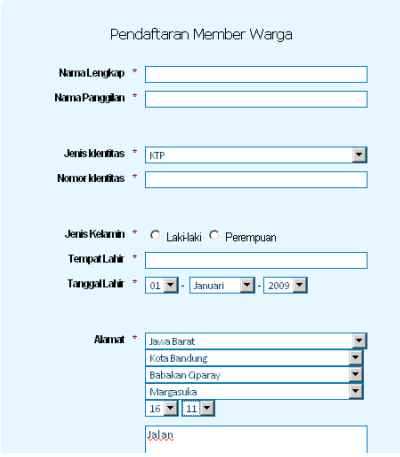
Pada Gambar 10, untuk melihat deskripsi salah satu takson, dapat dilakukan dengan meng-klik salah satu takson sehingga akan muncul halaman deskripsi takson seperti yang diperlihatkan oleh Gambar 11.

6.2 Sistem Informasi Penduduk (SIP)

Pada SIP, komponen pohon dipakai menggantikan modul pengelolaan data alamat

penduduk yg sebelumnya diimplementasi tanpa komponen. Model data pada modul pengelolaan data alamat penduduk sebelumnya diimplementasi dalam tabel Provinsi, Kota/Kabupaten, Kecamatan, Kelurahan, RT dan RW. Kemudian dikelola dengan sebuah *Class* AlamatController.php yang nantinya akan mengakomodasi proses pendaftaran data penduduk. Implementasi dengan komponen menggantikan beberapa tabel pada sistem lama, untuk membuktikan bahwa komponen pohon yang dibangun dapat dipakai dengan lebih efesien.

Dapat kita lihat pada Gambar 12, pada *user-interface* pendaftaran warga, *form* alamat diisi dengan memilih provinsi, kota/kabupaten di provinsi tersebut, kecamatan di kota/kabupaten tersebut, kelurahan di kecamatan tersebut dan seterusnya. Sehingga, pada AlamatController.php terdapat beberapa operasi seperti: `daftarProvinsi()`, `daftarKota($id_prov)`, `daftarKecamatan($id_kota)`, dan seterusnya untuk setiap level di struktur pembagian wilayah administratif.



Gambar 12. *User-interface form* pendaftaran penduduk

Dengan menggunakan komponen pengelolaan data berstruktur pohon, AlamatController.php cukup memiliki satu operasi yang berlaku untuk semua jenjang wilayah administratif, yaitu operasi `getChildren($nodeId)`. Selain itu, dengan memakai komponen ini, kinerja DBMS relatif lebih baik dari yang sebelumnya, karena semua data wilayah disimpan dalam satu tabel saja.

7. KESIMPULAN

Dari hasil penelitian ini, diperoleh komponen struktur pohon di berbagai *platform*, terutama pada *platform* PHP dan MySQL yang kami kembangkan. Pengujian komponen dilakukan pada sistem herbarium maya dan SIP. Dari pengujian tersebut, terbukti penggunaan perangkat lunak berbasis komponen dapat mengurangi waktu pengembangan (*development time*), meningkatkan produktifitas dan kualitas, serta mengurangi kompleksitas pengembangan sistem seperti yang telah disebutkan pada literatur Council (2000) mengenai manfaat pengembangan perangkat lunak berbasis komponen. Perlu diperhatikan juga, kelebihan dari pengembangan perangkat lunak berbasis komponen adalah sifat *reuse*-nya, oleh karena itu, manfaat dari penggunaan komponen kurang dirasakan jika komponen ini hanya dipakai untuk satu atau dua aplikasi saja. Manfaatnya akan terasa jika telah banyak di-*reuse* oleh aplikasi-aplikasi yang lain.

8. SARAN

Seperti yang telah disebutkan sebelumnya, pengembangan perangkat lunak berbasis komponen hanya akan terasa manfaatnya jika komponen ini dipakai secara luas. Agar komponen ini dapat diterima secara luas, tentu perlu ditingkat daya guna komponen ini.

Pengembangan komponen data di berbagai *platform* juga tidak terbatas hanya untuk struktur pohon saja. Pengembangan komponen data di berbagai *platform* juga berlaku untuk struktur-struktur kompleks yang lain (misalnya: *graf*). Oleh karena itu, perlu dilakukan juga pengembangan komponen data untuk struktur kompleks yang lain.

PUSTAKA

Council, William T. dan Heineman, George T., (2000), "Component-Based Software Engineering." Addison-Wesley: Upper Saddle River.

Crowley, James, (2004) *Tree structures in ASP.NET and SQL Server*. Diakses pada 8 Januari 2010 dari http://www.developerfusion.com/article/4633/tree-structures-in-aspnet-and-sql-server/

Liem, Inggriani dan Azizah, F.N., (2010). *Metadata Approach in Modeling Multi Structured Data Collection Using Object Oriented Concepts*. In Proceedings of the International Conference on Networking and Information Technology, Manila, 2010.

Knuth, D. E., (1968), *The Art of Computer Programming*, Addison-Wesley.

Microsoft (2009). *Using hierarchy Data Types (Database Engine)*. Diakses pada 8 Januari 2010 dari http://technet.microsoft.com/en-us/library/bb677173.aspx

Tegels, Kent (2008), *Model Your Data Hierarchies With SQL 2008*. Diakses pada 8 Januari 2010 dari http://msdn.microsoft.com/en-us/magazine/cc794278.aspx