BAB II

LANDASAN TEORI

2.1 Model Based User Interface Development

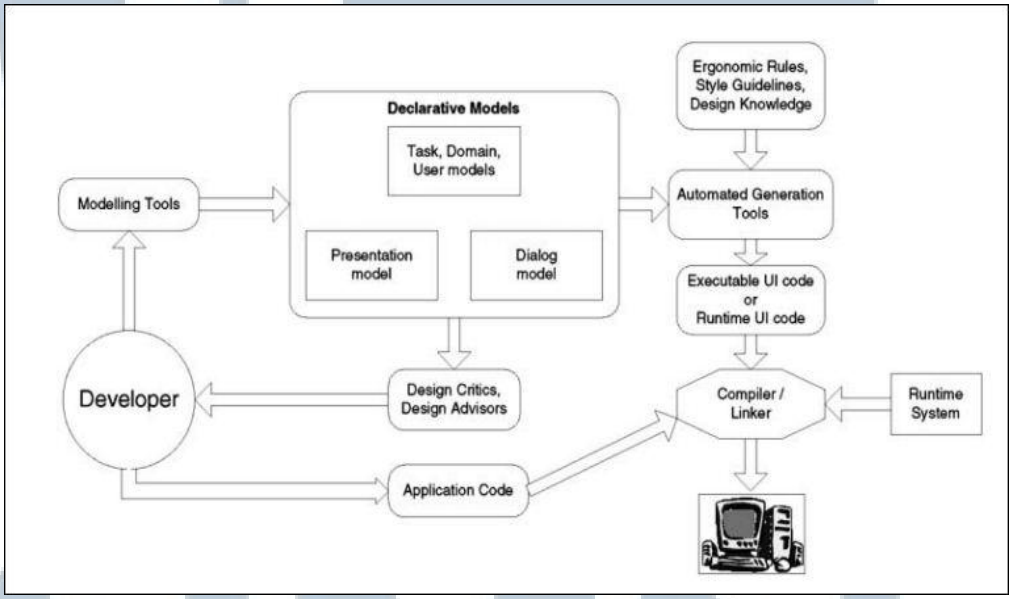
Model based user interface development (MB-UID) merupakan sebuah pendekatan dalam membuat UI secara otomatis berdasarkan spesifikasi yang terdapat dalam *declarative models* (Schlungbaum, 1996). MB-UID memiliki tiga fitur utama, yaitu mendukung pembuatan UI secara otomatis, memanfaatkan *declarative methods* dalam menentukan UI, dan mempunyai metodologi yang mendukung pengembangan UI (Griffiths, 2001). *Developer* yang menggunakan pendekatan ini membuat UI dengan membangun sebuah *model* yang menggambarkan UI yang diinginkan daripada menulis *source code* program untuk mendapatkan UI yang diinginkan (Szekely, 1996).

Menurut Szekely, penggunaan *declarative models* dalam membangun UI memberikan beberapa keuntungan yaitu.

a. *Design* dan *runtime tools* yang kuat. *Declarative models* memungkinkan pembuatan *tools* untuk membantu *developers* pada saat *design* dan *user* pada saat *runtime*.
b. Konsistensi dan *reusability*. Komponen-komponen dari sistem terdapat di dalam *model* sehingga mendukung konsistensi dalam sistem dan mendukung *reusability* dalam pembuatan UI baru.
c. Mendukung *conceptual design*. *Model* memungkinkan *designer* untuk mengambil keputusan dalam melakukan *design*, sehingga memberanikan

mereka untuk memikirkan lebih lanjut mengenai apa yang sedang mereka buat.

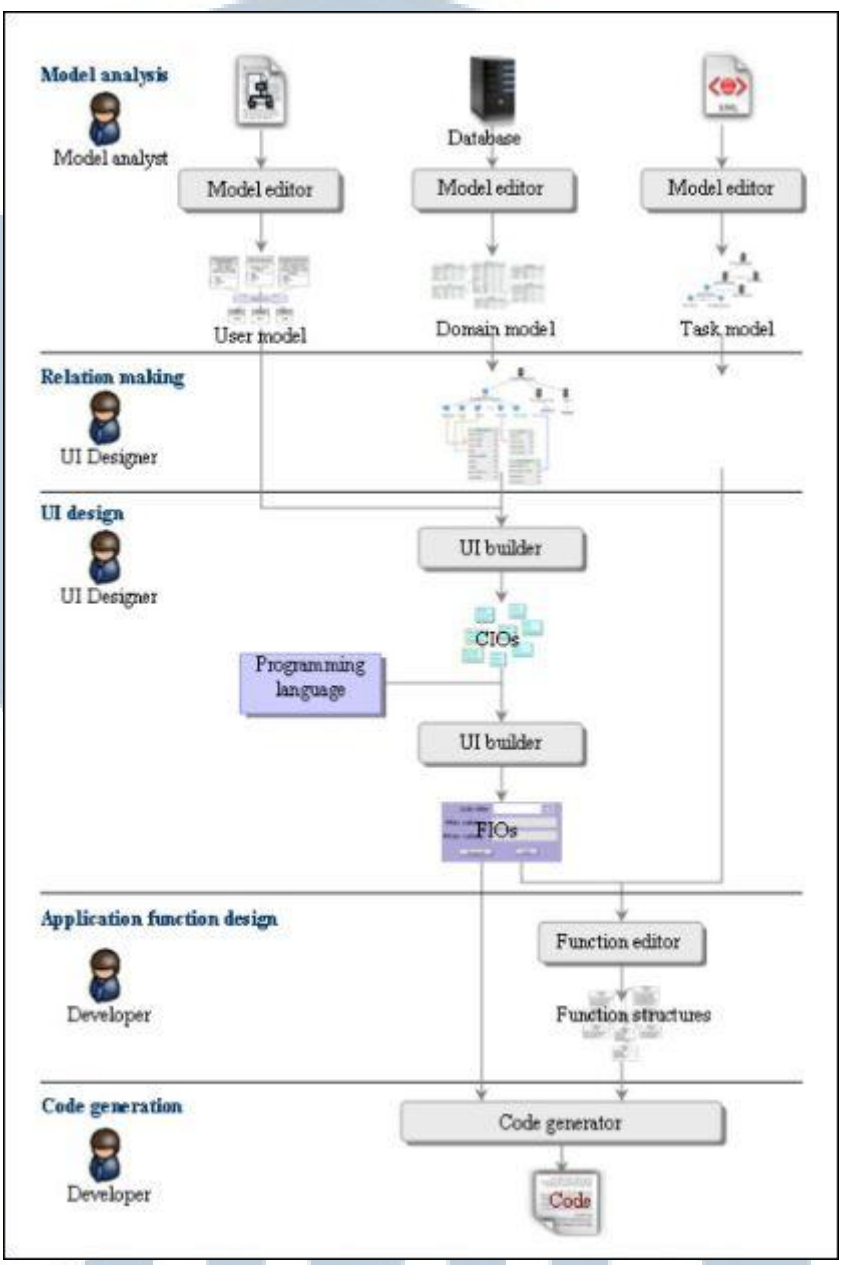
d. Pengembangan yang iteratif. *Model* dapat dieksekusi sebelum semua detil dari UI di-*design* sehingga *developer* dapat bereksperimen dengan *design* pada awal proses pengembangan dan menemukan cacat pada *design* sebelum nantinya sulit untuk diubah.



Gambar 2.1 *Model-based Development Life-Cycle* (Griffiths, 2001, hal. 3).

Gambar 2.1 merupakan sebuah siklus bagaimana *model based* UI *development* bekerja. *Developer* akan membuat model-model yang diperlukan dengan sebuah *tools*. Model-model tersebut kemudian diproses berdasarkan *rules* atau *guidelines* menjadi sebuah spesifikasi UI. Spesifikasi ini akan dihubungkan dengan *code* dari aplikasi dan dijalankan.

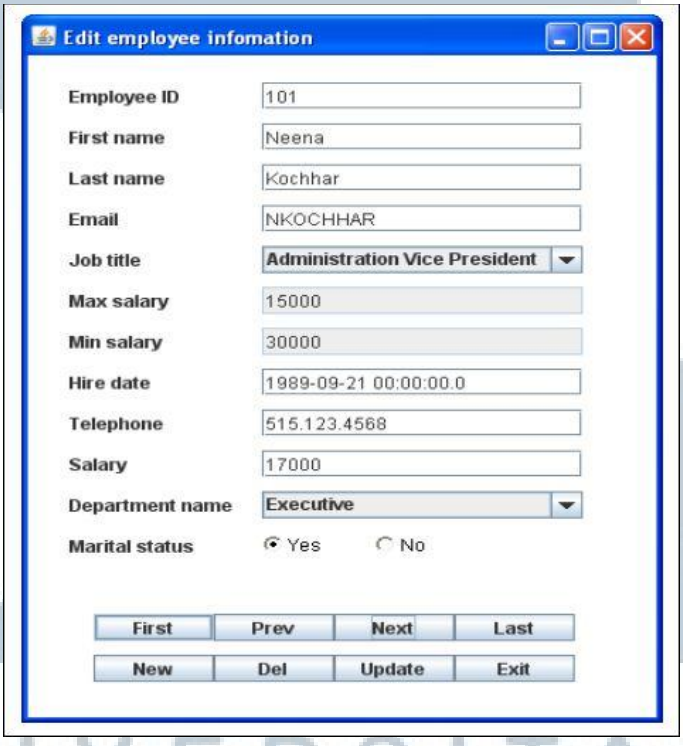
Salah satu contoh dari aplikasi yang menggunakan pendekatan *model-based* UI *development* adalah DB-USE (Tran, 2010).



Gambar 2.2 Arsitektur DB-USE (Vi Tran, 2010, hal. 6)

DB-USE menggunakan perpaduan dari *task model, domain model,* dan *user model* untuk membuat UI. *Model Editor* digunakan untuk memuat *task model* dari sebuah *file* XML, *user model* dari diagram, dan *domain model* dari sebuah *database*. Setelah model-model tersebut dimuat, maka akan dilakukan

verifikasi dan modifikasi oleh *Model Analyst*. Pada *relation making* dilakukan asosiasi antara *domain model* dengan *task model*. Pada tahap UI *design*, UI *builder* akan membuat *Concrete Interaction Object* (CIO) berdasarkan asosiasi tersebut dengan *user model*. *User model* digunakan untuk membuat UI dan memilih *control type* sesuai keinginan *user*. Setelah CIO dibuat, maka dengan bahasa pemrograman tertentu akan dibuat *Final Interaction Object* (FIO) berdasarkan CIO tersebut. Lalu, *Function Editor* akan menentukan *function* untuk FIO tersebut. Pada akhirnya *code* untuk UI dan aplikasi akan dibuat dengan *code generator*.

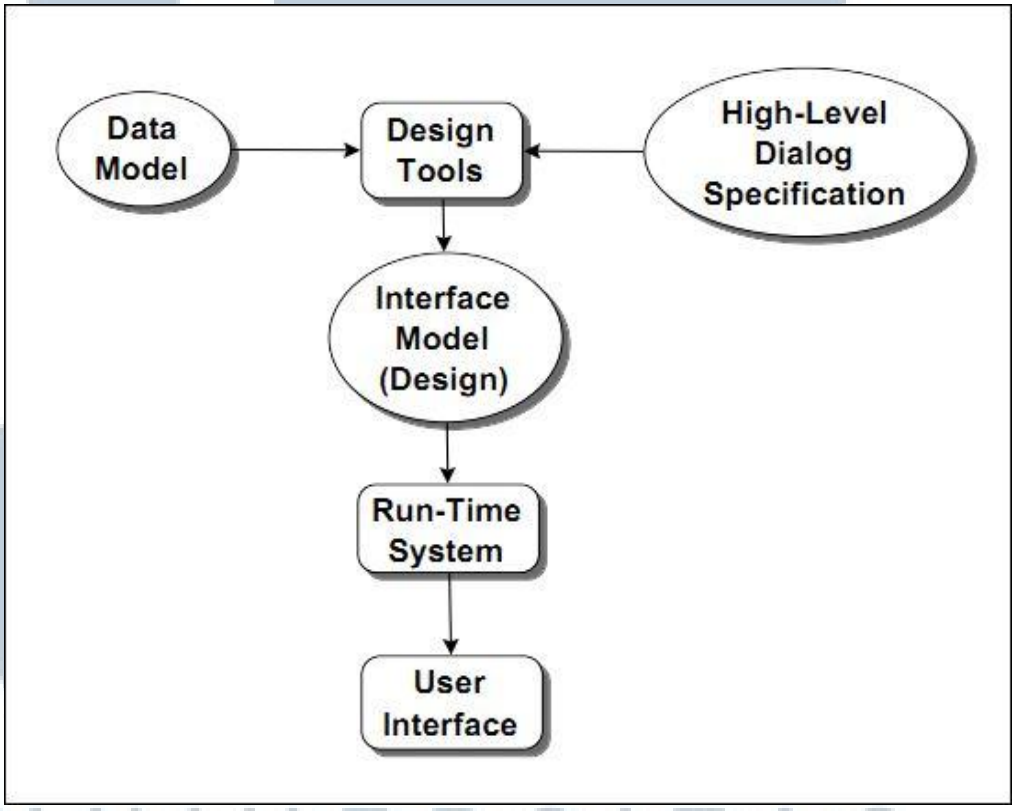


Gambar 2.3 Contoh tampilan halaman yang dibuat dengan DB-USE (Vi Tran, 2010, hal. 11)

Gambar 2.3 merupakan contoh tampilan untuk halaman *edit employee* yang dibuat menggunakan DB-USE dengan bahasa pemrograman Java.

2.2 Data Model dalam Model Based User Interface Development

Data model merupakan jenis *model* pertama yang digunakan dalam MB-UID (Puerta, 1997). *Model* ini merupakan struktur data yang terdapat pada aplikasi. *Model* ini terbukti dapat membuat sebuah *layout* statis untuk *interface* dengan pemberian beberapa *rule* (Puerta, 1994).

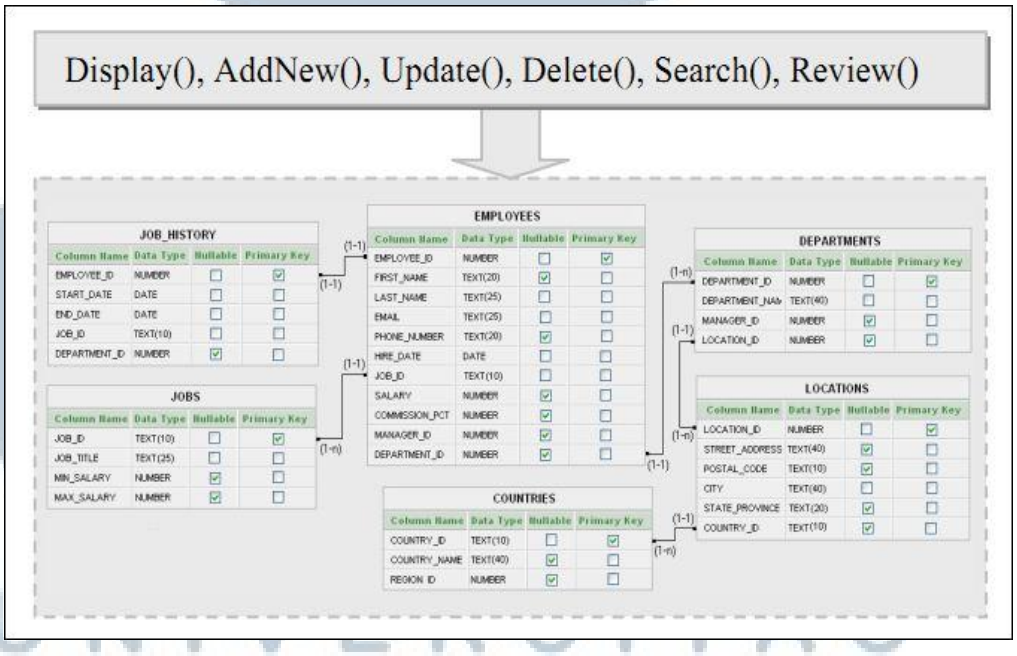


Gambar 2.4 *Automatic* UI *Generation* menggunakan *data models* (Puerta, 1994, hal. 4).

Gambar 2.4 merupakan sebuah *framework* umum dalam pembuatan UI secara otomatis menggunakan *data model*. *Design* dari UI dihasilkan menggunakan *tools* yang memeriksa *data model*. *Design* tersebut kemudian akan diimplementasi oleh sistem pada saat *runtime* menjadi sebuah UI.

Data model biasanya terdapat di dalam *domain model* yang merupakan representasi objek dan hubungan antar objek tersebut. *Domain model* menyediakan fitur-fitur khusus dalam membuat UI. Fitur-fitur tersebut adalah atribut dari setiap objek yang ada dalam *domain model* dan hubungan antara objek tersebut (Tran, 2010).

Salah satu contoh aplikasi yang menggunakan *domain model* untuk membuat *user interface* adalah DB-USE (Tran, 2010).



Gambar 2.5 *Data Model* pada DB-USE (Sumber: Vi Tran, 2010, hal. 5)

Gambar 2.5 merupakan contoh *data model* yang digunakan pada DB-USE untuk membuat UI. Atribut-atribut dari setiap entitas pada *data model* tersebut digunakan untuk mengidentifikasi atribut dari *concrete* UI atau komponen dari UI.

2.3 Framework

Framework adalah sebuah aplikasi yang *reuseable*, *semi-complete* yang dapat digunakan untuk mengembangkan sebuah *custom application* dan pada umumnya ditujukan untuk *business unit* dan *application domain* tertentu (Wu, 2011). *Frameworks* dibangun dari kumpulan objek, sehingga *design* dan *code* dari *frameworks* dapat digunakan ulang (*re-use*) (IBM, 1997). Melalui sifat *re-use* ini, penggunaan *frameworks* dalam pengembangan aplikasi memberikan produktivitas yang lebih tinggi dan waktu pengembangan yang lebih singkat (Riehle, 2000).

Sebuah *framework* digunakan untuk menyelesaikan sebuah *problem* tertentu dengan melakukan kustomisasi pada abstraksinya dan *class-class*-nya, yang memungkinkan arsitektur dari *framework* untuk dapat digunakan ulang oleh semua solusi spesifik untuk suatu *problem domain* (Basili, 1996). Para *developer* yang menggunakan *framework*, akan menggunakan ulang *design* dan implementasi *framework* tersebut (Riehle, 2000). Oleh karena itu, pengembangan aplikasi dapat dilakukan dengan lebih cepat.

Sebuah *framework* dikembangkan dengan melakukan langkah-langkah sebagai berikut (IBM, 1997).

1. Identifikasi abstraksi utama. Identifikasi abstraksi yang diperlukan oleh *client* untuk menyelesaikan *problem* mereka dan memberikan solusi dengan abstraksi tersebut. Cara paling mudah untuk mengidentifikasi abstraksi tersebut adalah dengan memeriksa solusi-solusi yang telah ada sebelumnya.
2. Rancang bagaimana *framework* akan digunakan. Fokus pada bagaimana interaksi terhadap *framework*. *Class* dan *function* apa saja yang akan digunakan.
3. Implementasi, uji coba, dan perbaikan rancangan.

Agar sebuah *framework* dapat digunakan dengan baik maka *framework* tersebut memiliki kriteria sebagai berikut (IBM, 1997).

a) *Complete*. *Framework* mendukung fitur-fitur yang dibutuhkan dan memberikan implementasi awal dan *built-in function* jika memungkinkan.
b) *Flexible.* Abstraksi dapat digunakan untuk konteks yang berbeda
c) *Extensible.* Fungsionalitas dari *framework* dapat dengan mudah ditambah dan dimodifikasi.
d) *Understandable.* Interaksi *framework* jelas dan *framework* didokumentasi dengan baik. Ikuti standar-standar perancangan dan panduan *coding*, serta sediakan contoh aplikasi yang mendemonstrasikan penggunaan *framework*.