SNATI
Seminar Nasional
Aplikasi Teknologi Informasi 2004
Yogyakarta, 19 Juni 2004

Pengkayaan UML dengan Metode Formal

Pujianto Yugopuspito
Jurusan Teknik Informatika – Universitas Pelita Harapan
Jl. M.H. Thamrin Boulevard, Lippo Karawaci, Tangerang, 15811
Telp. (021) 5460901, Faks. (021) 5460910
e-mail: yugopuspito@uph.edu

Abstrak

Unified Modeling Language (UML) is a de-facto object-orieted model standard. Many models can be visualized and designed by using UML tool, e.g. Rational Rose. We enrich the UML by means of adding a capability to produce a set of formal specifiation of Z notation. This additonal capability is a part of a studi on formal framework of transformation relational databases to object-relational databases. The enrichment of UML in formal specification will give an opportunity to explore UML class diagram in the sense of formal methods. This approach will enable to use the streght of formal methods within UML.

Keywords: *class diagram, UML, formal methods, formal specification, Z.*

1. Pendahuluan

Penelitian ini mencoba untuk mempelajari lebih mendalam tentang perubahan relational data model ke object-relational data model. Cara yang digunakan adalah dengan penerapan metode formal. Pendekatan metode formal memungkinkan untuk membuat verifikasi, validasi dan pembuktian dari suatu model.

Secara praktikal representasi suatu model dengan diagram akan lebih mudah dimengerti secara intuitif. Relational data model secara grafikal direpresentasikan dengan IDEF1X, Federal Information Processing Standard Publication (FIPS) No.184 diterbitkan oleh National Institute of Standard and Technology (NIST) pada tahun 1993. Standar ini banyak diguna dalam industri untuk mendesain relational data model. Sementara object-relational database model digambarkan dengan UML class diagram. Unified Modeling Language (UML) adalah *de-facto* standar untuk pemodelan berbasis obyek. UML adalah produk dari sebuah konsorsium Object Management Group. Standar yang digunakan pada penulisan ini adalah UML versi 1.3.

Pendekatan metode formal mampu menjelaskan secara matematis perubahan dari suatu diagram ke diagram lain. Pendekatan ini mengharuskan suatu cara yang cepat untuk mengekstrak suatu diagram ke suatu spesifikasi formal. Dalam penelitian ini, diagram IDEF1X dan UML class diagram diekstrak menjadi spesfikasi formal bahasa Z.

Tulisan ini bertujuan untuk membahas secara khusus tentang proses otomisasi pengekstrakan UML diagram menjadi satu set spesifikasi formal dalam notasi Z. Proses ini dapat disebut sebagai proses pengkayaan UML dengan metode formal. Pengkayaan UML ini dikerjakan dengan menambahkan kemampuan suatu sarana (tool) pembuat UML model.

Suatu UML model dibuat dengan sarana UML, Rational Rose, kemudian proses otomisasi dikerjakan dengan menambahkan modul pada Rational Rose yang dapat mengekstrak model tersebut sehingga dapat direpresentasikan dalam spesifikasi formal. Modul dibuat dalam bahasa Rational Script, yang disediakan oleh Rational Rose. Spesifikasi

formal yang dihasilkan berupa notasi Z. Lebih lanjut hasil spesifikasi formal tersebut digunakan sebagai masukan program formal Z/Eves. Hal-hal inilah yang menjadi rumusan masalah dari tulisan ini.

2. Landasan Teori

Dua hal yang menjadi landasan dalam pembuatan modul tambahan pada sarana UML, Rational Rose, yaitu difinisi dari UML class diagram yang digunakan untuk mengambarkan object-relational data model, dan aturan yang digunakan untuk menterjemahkan UML class diagram ke spesifikasi formal dalam notasi Z.

2.1 UML Class Diagram sebagai representasi dari Object-Relational Data Model.

Pada penelitian ini hanya dibahas beberapa gambar representasi dari object-relational data model, sesuai dengan gambar representasi relational data modelnya.

2.1.1 Konsep Entity

Konsep entity pada IDEF1X ada dua jenis, yaitu *identified-independent entity* (atau *independent entity*) dan *identified-dependent entity* (atau *dependent entity*). *Independent entity* dan *dependent entity* dapat dipetakan langsung ke konsep class. Nama entity menjadi nama dari class, sementara nomor entity dipetakan oleh sistem menjadi sejenis *object identity* (OID). Perbedaan independent entity dan dependent entity direpresentasikan dengan *stereotype*, << independent>> dan <<dependent>>.

Beberapa tambahan informasi masih diperlukan untuk membuat UML class diagram yang lengkap. Ekstensi untuk *User Define Type* (UDT) dan batasan-batasan yang lain juga harus disiapkan, misal batasan *acyclic graph* pada konsep pewarisan (inheritance) dari class yang digunakan.

2.1.2 Konsep Attribute

Suatu *attribute* atau *key* pada IDEF1X dapat langsung direpresentasikan sebagai *attribute* dari sebuah class. Primary key, alternate key dan key yang lain dipetakan menjadi attribute pada UML class diagram pada kompartement ke dua. *Foreign key* dibuang karena konsep ini tidak diijinkan pada UML.

Informasi dari IDEF1X tidak cukup untuk mengisi seluruh properti dari sebuah attribute UML, seperti *visibility*, *multiplicity*, *initial value*, *changeability* dan *target scope*. Untuk itu digunakan nilai dasar (default value). Nilai dasar dari *visibility* adalah ***public***, *changeability* adalah ***changeable*** dan ***instance*** untuk *target scope*.

2.1.3 Konsep Relationship

Hubungan antara dua entity disebut sebagai relationship pada konsep IDEF1X. Relationship tersebut dapat dikategorikan menjadi *specific relationship* dan *non-specific relationship*. Kemungkinan jenis sebuah relationship adalah *identifying relationship*, *mandatory non-identifying relationship, optional non-identifying relationship*, *complete categorization relationship* atau *incomplete categorization relationship*. Sementara itu konsep hubungan antara dua class pada UML yang mungkin adalah *association*, *aggregation*, *composition* dan *generalization* (atau *inheritance*). Sehingga masih diperlukan analisa lebih lanjut untuk memetakan relationaship dari konsep IDEF1X ke konsep UML, walaupun demikian kesepadanan topografi dari kedua diagram memungkinkan pemetaan tersebut.

Cardinality dari satu relationship adalah limit dari suatu instance entity yang dapat berhubungan dengan entity yang lain pada relationship tersebut. Konsep *cardinality* ini berpadanan dengan konsep *multiplicity* pada UML class diagram. *Multiplicity* dapat dinyatakan dengan *role* pada ujung sebuah *assosciation*. Secara pasti, sebuah *multiplicity* adalah sebuah subset dari non-negative integer.

Identifying relationship adalah sejenis *specific connection relationship* di mana *child entity* selalu berjenis *dependent entity*. Secara umum *identifying relationship* dapat dipetakan menjadi sebuah *association*. Bila *child entity* adalah bagian dari *parent entity*, *association* ini diubah menjadi sebuah *aggregation*. Jika keberadaan *child entity* adalah sangat bergantung pada *parent entity*, *aggregation* dapat diubah menjadi sebuah *composition*.

Relationship label dapat digunakan sebagai *role name* dari *binary association* pada ke dua ujungnya. *Multiplicity* ditambahkan sesuai dengan *cardinality* dari relationship. Informasi properti yang tidak ada adalah *changeability*, *ordered/non-ordered*, *navigability*, *qualifier* dan *aggregation kind*. Nilai dasar yang digunakan untuk *navigability* adalah ***false***, *changeability* adalah ***true*** dan *aggregation kind* adalah ***unShared***.

Non-Identifying Specific Relationship adalah sejenis hubungan di mana *child entity* dan *parent entity* bertipe *independent entity*. *Non-Identifying relationship* dapat *dibagi menjadi mandatory non-identifying relationship dan optional* non-identifying relationship. *Non-identifying relationship* dipetakan menjadi *association* dengan *aggregation type*, ***none.*** *Mandatory non-identifying relationship* direpresentasikan sebagai sebuah *association* dengan ***multiplicity one***, sementara *optional non-identifying relationship* diterjemahkan sebagai sebuah *association* dengan ***mulitplicity zero-or-one***. Relationship name dapat dipetakan menjadi *role name* pada kedua ujung *association*. *Multiplicity* ditambahkan sesuai dengan *cardinality* pada *relationship*. Informasi yang kurang adalah *changeability*, *ordered/non-ordered*, *navigability*, *qualifier* dan *target scope*. Nilai dasar dari *navigability* adalah ***false***, *changeability* adalah ***true*** dan *target scope* adalah ***instance***.

Categorization Relationships dipetakan menjadi *inheritance* (*generalization*) pada konsep UML. *Generic entity* dipetakan menjadi *superclass* sementara *category entity* dipetakan menjadi *subclass*. *Complete* dan *incomplete set* dipetakan dengan menambahkan *stereotype* <<complete>> dan <<incomplete>>.

Non-Specific Connection Relationship dipetakan menjadi *association* dengan *aggregate type*- nya adalah ***none***. Properti yang kurang dan set nilai dasarnya sama dengan *non-identifying relationship*.

2.1.4 Operation

Pada *relational database management system*, *operation* dipisahkan dari data model. Hal ini sangat berbeda dengan konsep UML, di mana *operation* sebagai satu kesatuan dalam sebuah data model. Sehingga konsep *operation* harus ditambahkan.

2.2 Representasi Class Diagram dalam Notasi Z.

Spesifikasi formal bahasa Z atau notasi Z adalah bahasa fomal yang pada awalnya dikembangkan oleh J.R. Abrial pada akhir 1970. Pengembangan lebih lanjut dikerjakan oleh Hayes, Morgan, Sorensen, Spivey, Sulfrin dan lain-lain. Z adalah spesifikasi formal berbasis pada *first order predicate logic* dan *Zermelo-Fraenkel set theory*. Standar yang digunakan mengacu pada panel ISO/IEC JTC1/SC22/WG19.

Spesifikasi formal bahasa Z telah terbukti sangat berguna untuk memodelkan fungsi transformasi data yang kompleks. Baik pada dunia pendidikan maupun dunia industri, notasi

Z telah menjadi salah satu bahasa spesifikasi formal yang paling banyak digunakan. Hal-hal inilah yang mendasari pemilihan penggunaan spesifikasi formal bahasa Z.

Pembuatan spesifikasi formal bahasa Z dari UML diagram, lebih berarti sebagai terjemahan UML diagram ke dalam notasi Z. Notasi Z yang digunakan untuk merefleksikan struktur dari UML, tetapi tidak digunakan untuk mendifinisikan UML dalam notasi Z. Spesigikasi formal ini sangat terbatas pada UML class diagram yang berhubungan dengan object-relational data model.

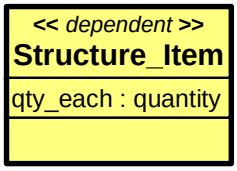
Secara umum, proses ini terdiri dari tiga bagian. Pertama, membuat formalisasi dari struktur class dasar, bukan *aggregated class* atau *subclass*. Setiap properti individual dari setiap class didifinisikan. Ke dua, membuat formalisasi dari struktur *aggregation* dan *generalization hierarchy*. Pada tahap ini, *aggregate* dan *subclass* didifinisikan dengan menggunakan class dasar yang telah dibuat. Terakhir, membuat formalisasi dari hubungan antar class. Hubungan diformalisasikan dalam batasan hubungan antar *instance schema*.

Class dasar dan attribute dispesifikasikan sebagai *schema* pada notasi Z. Nama class digunakan sebagai nama dari *schema* yang dibuat. *Attribute* dan *attribute type* diterjemahkan sebagai variable dengan tipe tertentu yang didifinisikan pada bagian deklarasi dari sebuah *schema*. Nama dari attribute diterjemahkan sebagai nama dari variable, dimulai dengan huruf kecil. *Attribute type* didifinisikan sebagai *given-type* atau schema yang lain. Deklarasi minimum dari sebuah schema adalah *ident*, dengan tipe ***ObjID***. Ini adalah representasi dari *object-identity*. Batasan *attribute* dan batasan class dapat ditambahkan pada bagian predikat dari suatu *schema*.

$\cup$***Basic ClassName***_____
$\rightarrow$*ident :Obj ID*
$\rightarrow$*OtherAttributesType*
$\angle$_____

Misalkan, ***StructureItem*** class mempunyai *qty_each* dengan tipe *Quantity*. Tipe ini dapat secara explisit didifinisikan dengan *abbreviation definition*. Kemampuan ini hanya benar di object-relational data model, karena relational data model tidak mempunyai fitur user-define type (UDT). Spesifikasi formal dari class dasar ini dapat diberikan sebagai berikut:

[ObjID]
Quantity $==$ N
$\cup$***StructureItem***_____
$\rightarrow$*ident : Obj ID*
$\rightarrow$*qtyEach : Quantity*
$\angle$_____



Gambar 1. Structure Item Class

Multiplcity merepresentasikan jumlah koneksi suatu class dengan class yang lainnya. *Multiplicity* ini dapat diterjemahkan sebagai *generic schema* dalam notasi Z.

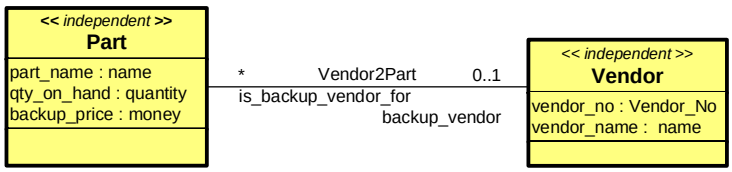
```
⎸[X, Y]_____________________________________________
→Many2Many : Π ( X φ Y )
→One2Many : Π ( X ♣ Y )
→One2EMany : Π ( X ⌋ Y )
→One2One : Π ( X ♥ Y )
→One2EOne : Π ( X φ Y )
→EOne2Many : Π ( X ϕY )
→EOne2EMany : Π ( X ∅ Y )
→EOne2EOne : Π ( X ℜ Y )
∩_____
→Ar : X φ Y ∞ r ε One2EOne ⇔ r ε ( X ♥ Y ) ƒ r ε ( X ⌋ Y )
∠_____________________________
```

Association adalah hubungan struktural antar dua obyek. Sebuah *instance* dari sebuah *association* disebut *link* yang merepresentasikan suatu *tuple* dari *association end*. Pada *binary association* hanya ada dua *assocition end*. Hubungan ternary dan n-ary yang lain susah untuk diterapkan dan secara praktikal akan dihindari. Bahkan pada kebanyakan relational data model hanya *binary relationship* yang diperbolehkan.

Setelah mendifinisikan setiap *schema* dari class yang digunakan untuk sebuah *association*, maka *association* dapat direpresentasikan sebagai sebuah *schema*. Pada bagian deklarasi, variable dari semua type yang terlibat dalam *association* tersebut dinyatakan. Pada bagian predikat, perlu dinyatakan bahwa *set* dari *intance* dan juga batasan *multiplicity*-nya. *Role name* adalah nama dari *domain* dan *range* dari *association* tersebut.

Misal class ***Vendor*** mempunyai *one-to-many multiplicity association* dengan class ***Part***. Bila *association* ini bernama ***Vendor2Part***, maka dapat dituliskan:

```
∪Vendor2Part_____________________
→parts : Π Part
→vendors : Π Vendor
→vendor2part : Vendor φ  Part
∩_____
→dom vendor2part ζ vendors
→ran vendor2part = parts
→ vendor2part ε One2Many [vendors, parts]
→A p1, p2 :  parts ∞ p1.ident = p2.ident ⇒ p1 = p2
→A v1, v2 :  vendors ∞ v1.ident = v2.ident ⇒ v1 = v2
∠_____________________________
```



Gambar 2. Diagram UML dari association vendor – part

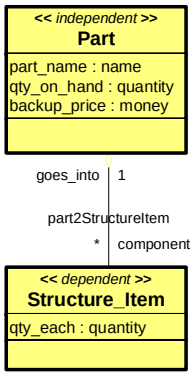
UML mendukung dua jenis *aggregation*, bentuk lemah (*weak*) dan kuat (*strong*). Bentuk lemah ini disebut *aggregation*, di mana *sharing component* antara *aggregate* diperbolehkan. Ini adalah bentuk khusus dari *association*. Bentuk kuatnya disebut *composition*, di mana hubungan *whole-part* terjadi dan keberadaan *part* sangat tergantung kepada *whole*. Multiplicity dari aggregate (whole) tidak boleh lebih dari satu (*one*), sementara part dapat dibuat dengan multiplicity lebih dari satu (*one-or-more*).

Untuk *non-recursive aggregation*, sebuah *schema* dapat dibuat setelah *schema* dari class dasar telah didifinisikan. Pada bagian deklarasi terdiri dari *object-identifier* dan *attribute* lain yang diperlukan. Jika variable yang direpresentasikan mempunyai multiplicity lebih dari satu, maka variable tersebut disimbolkan sebagai ***set*** (Π atau Φ), selain itu variable

direpresentasikan sebagai elemen tunggal. Pada bagian predikat dari schema, batasan *multiplicity* dari komponen dan batasan kekhasan (*uniqueness constraint*) diekspresikan. *Schema* tambahan mungkin diperlukan untuk menterjemahkan sebuah *composition*.

Misalkan class ***StructureItem*** adalah hasil aggregasi dari class *Part,* seperti pada Gambar 3. Bila *aggregation* yang terjadi diberi nama ***Part2StructureItem*** dengan multiplicity *exists-one-to-many* maka spesifikasi formal dapat ditulis sebagai berikut:

∪***Part2StructureItem***_____
→*parts* : Π *Part*
→*structureItems* : Π *StructureItem*
→*part2structureItem* : *EOne2Many* [*Part, StructureItem*]
∩_____
→**dom** *part2structureItem* ζ *parts*
→**ran** *part2structureItem = structureItem*
→*part2structureItem* ε *EOne2Many* [*parts,structureItems*]
→A p_1, p_2 : *parts* ∞ p_1.*ident* = p_2.*ident* ⇒ $p_1 = p_2$
→A si_1, si_2 : *structureItems* ∞ si_1.*ident* = si_2.*ident* ⇒ $si_1 = si_2$
∠_____



Gambar 3. Diagram UML dari aggregation part – structure_item

Inheritance atau disebut juga *generalization* atau *specialization*, didifinisikan sebagai hubungan antara *supertype* dengan *subtype* atau *superclass* dengan *subclass*. *Subtype* atau *subclass* mewarisi properti dari *supertype* atau *superclass.*

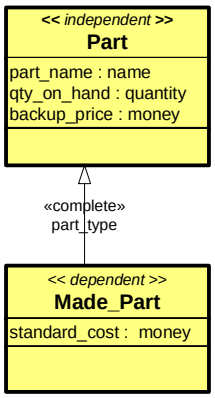
Sebuah *schema* dapat digunakan untuk menterjemahkan setiap *subtype* setelah *supertype* didifinisikan. Pada bagian deklarasi, *supertype* akan digunakan sebagai tipe dari *subtype.* Jika *supertype* tersebut adalah *abstract* atau ada ***subset*** dari *subtype* yang saling asing (*disjoint*) maka batasan (*constraint*) tetang kondisi tersebut harus didifinisikan pada bagian predikat dari schema. Perlu diingat bahwa *type* tersebut adalah sebuah class.

Contoh misalkan class ***MadePart*** adalah *subclass* dari class ***Part*** dan masih mempunyai *attribute* ***standard_cost*** bertipe ***Money,*** lihat Gambar 4. Jika *inheritance* tersebut disebut ***Part2MadePart*** maka spesifikasi formal dari *inheritance* tersebut dapat didinisikan sebagai berikut:

∪***MadePart***_____
→*ident* : *ObjID*
→*part* : *Part*
→*standardCost* : *Money*
∠_____

∪***Part2MadePart***_____
→*parts* : Π *Part*
→*madeParts* : Π *MadeParts*

∩_____
→A p_1, p_2 : $parts \propto p_1.ident = p_2.ident \Rightarrow p_1 = p_2$
→A mp_1, mp_2 : $madeParts \propto mp_1.ident = mp_2.ident \Rightarrow mp_1 = mp_2$
→©□.mp : $madeParts \propto mp.part$ □〉partition $part$
∠_____



Gambar 4. Diagram UML dari Inheritance Part – Made_Part

3. Metode Penelitian

Metode penelitian pada penelitian ini adalah *prototyping*. Sebuah modul prototype diintegrasikan pada sarana pembuatan UML model, dalam hal ini Rational Rose 98. Modul tersebut dibangun dengan memanfaatkan salah satu fitur dari Rational Rose yaitu modul ekstensibilitas.

4. Hasil Penelitian dan Pembahasan

Salah satu hasil dari penelitian ini adalah pengkayaan UML dengan metode formal. UML class diagram dapat secara otomasi dapat diekstrak untuk mendapatkan spesifikasi formal dalam notasi Z.

Modul dibangun dengan mengunakan Rose Scripting Language. Rose Scripting Language adalah versi ekstensi dari Summit BasicScript language. Ekstensi memang disiapkan oleh Rose untuk mengantisipasi kebutuhan mengotomisasi beberapa fungsi yang telah ada. Rose script editor dan compiler hanya berjalan pada lingkungan Rose.

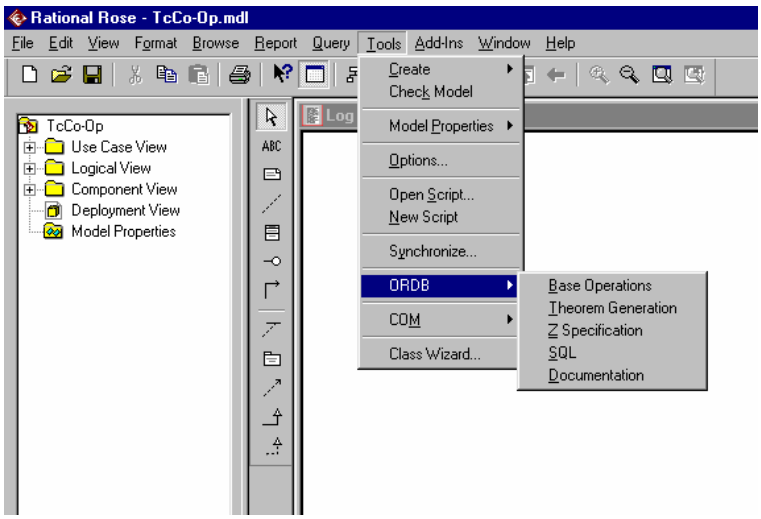
Hasil ekstraksi ini berupa ASCII file dalam format LaTeX., yang dapat langsung digunakan sebagai masukan pada program Z/Eves. Z/Eves adalah sarana (*tool*) Z yang sangat maju. Sarana Z ini mengunakan *Eves formal method*. Menurut pengkategorian sarana metode formal oleh NASA (1997), Eves dikategorikan sebagai sarana metode formal level empat, artinya sangat formal yang mendukung mekanisasi. Dengan Z/Eves, spesifikasi formal dengan notasi Z dapat dikaji *syntax* dan *type checking*, *schema expansion*, *precondition calculation*, *domain checking* dan *theorem proving*.

Pengunaan spesifikasi formal dalam notasi Z lebih lanjut telah digunakan untuk membuat proses refinement dalam object-relational data model, Yugopupito dan Araki (2000). Proses refinement ini mengunakan pendekatan *pictorial refinement*. Contoh dari refinement melalui gambar adalah penambahan satu class baru untuk mengubah satu *many-to-many relationship* menjadi dua *one-to-many relationship*. Penerapan *no-repeat rule refinement* dan *smallest key rule refinement* juga dapat dibuktikan dengan mengunakan notasi Z.

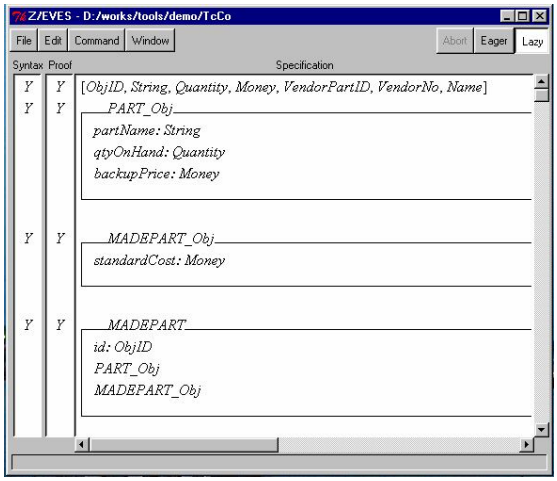
Lebih lanjut, penambahan modul ini juga telah dikembangkan untuk keperluan object-relational database seperti otomisasi pembuatan SQL sintaks. Penambahan operasi dasar manipulasi database dan untuk mengotomisasi pembuatan laporan dokumen.

Format dari ekstraksi SQL adalah ASCII file, yang dapat dengan mudah digunakan untuk keperluan pendifinisian database. Sintaks dari SQL ini mengacu pada standar SQL-92.

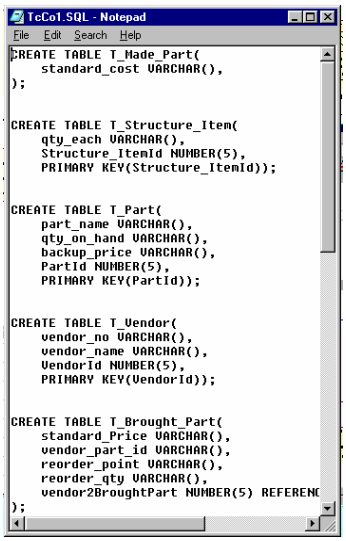
Format ekstraksi pelaporan dokumen dari UML class diagram yang telah didifinisikan mengunakan format Microsoft Words.



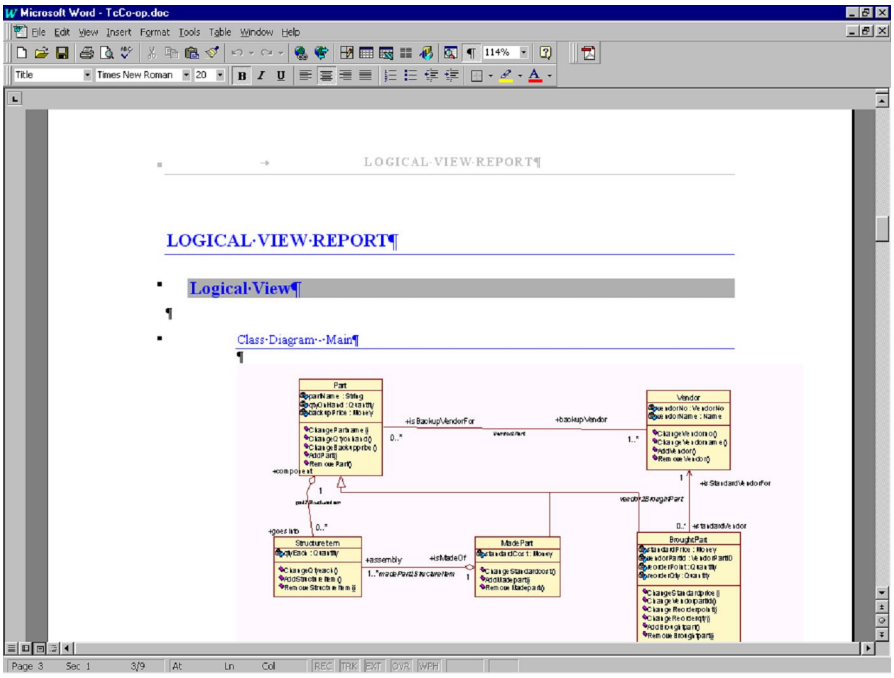
Gambar 5. Menu tambahan pada rational rose.



Gambar 6. Hasil masukan pada Z/Eves



Gambar 7. Hasil SQL sintaks



Gambar 8. Hasil Dokumentasi Pelaporan

5. Kesimpulan

Mengacu pada hasil dari penelitian ini maka dapat disimpulkan bahwa modul yang telah dibuat mampu mengekstraksi UML class diagram menjadi spesifikasi formal dalam notasi Z. Hal ini memberikan bukti bahwa pengkayaan UML dengan metode formal telah berhasil seperti yang diharapkan.

6. Keterbatasan Penelitian

Penelitian ini hanya terbatas pada pengekstrakan UML class diagram ke spesifikasi formal bahasa Z.

7. Penelitian Lebih Lanjut

Penelitian serupa sedang dikembangkan lebih jauh untuk mengakomodasi semua jenis diagram pada UML dengan mengunakan standar yang lebih baru, UML 2.0. Sehingga pada akhir penelitian diharapkan ada UML tool sejenis Rational Rose dengan basis *open-source* yang telah diperkaya dengan metode formal.

Daftar Pustaka

Holloway, C.M., (1997). Why Engineers should Consider Formal Methods. *Proceeding of the 16th Digital Avionics System Conference 1997*.

NASA (1997). *Formal Methods, Specification and Verification Guidebook for Software and Computer System – A Practitioner's Companion*

Yugopuspito, P dan Araki, K (2000). UML Refinement for Object-Relational Database. *Proceeding of International Symposium on Future Software Technology (ISFST) 2000*

Yugopuspito, P dan Araki, K (2001). Yugopuspito, P. and Araki, K.: Transformational Object-Relational Database Model in Formal Methods, *IPSJ Transaction on Mathematical Modeling and Its Applications*, 46(4), 71-80.