Penerjemah Lingu ke Java[1]

Oleh

Heru Suhartanto, heru@cs.ui.ac.id
Jimmy, jimmy@cs.ui.ac.id

Atas Nama TIM RUTI Fasilkom UI

Abstrak

Dokumen ini menjelaskan cara instalasi (pemasangan) dan pemakaian perangkat lunak (software atau program) penerjemah Lingu ke bahasa pemrograman Java. Lingu adalah bahasa spesifikasi perancangan program basis data yang dapat diverifikasi. Bahasa ini dikembangkan sebagai bagian dari Riset Unggulan Terpadu Internasional (RUTI) Fasilkom UI, lihat grammar (tatabahasa) nya di lampiran A. . Bahasa ini bersifat sederhana untuk memudahkan user memberikan spesifikasi program yang kemudian dapat diperiksa/verifikasi kebenarannya. Namun bahasa ini belum bisa dikompilasi untuk menghasilkan output yang bisa dipakai. Untuk itulah dikembangkan penerjemah Lingu ke Java agar dapat diperoleh output yang dapat mengimplementasikan spesifikasi yang direpresentasikan dalam bahasa Lingu.

Sebelumnya, penerjemah MuPL2Java dikembangkan dengan maksud yang serupa. Dari hasil penelitian yang dilakukan, MuPL dikembangkan lebih lanjut menjadi Lingu untuk penyederhanaan aspek verifikasi. Tehnik pengembangan MuPL2Java, yang dikembangkan dengan menggunakan JLex dan JavaCUP, berbeda dengan penerjemah Lingu ke Java yang menggunakan tehnik *attribute grammar* UUAG.

Pendahuluan

Secara umum, penerjemahan adalah suatu proses untuk mengubah bentuk, yaitu dari bentuk yang satu ke bentuk yang berbeda. Dalam istilah linguistik, penerjemahan berhubungan erat dengan proses transformasi dari suatu bahasa tertentu ke dalam bahasa lain, dengan mempertahankan arti yang terkandung di dalam bahasa yang diterjemahkan. Aktor dalam penerjemahan biasa kita sebut sebagai penerjemah. Istilah penerjemah dapat merujuk kepada seorang manusia maupun suatu alat bantu otomasi yang dibangun oleh manusia.

[1] Bagian dari proyek RUTI II tahun 2003, didanai oleh Kementerian Riset dan Teknologi, Republik Indonesia

Penerjemahan bukanlah suatu pekerjaan yang trivial, baik dikerjakan oleh manusia ataupun alat bantu. Untuk menghasilkan hasil penerjemahan yang akurat, diperlukan aturan-aturan mengenai tata cara penerjemahan yang harus dilakukan. Aturan-aturan ini terkait erat pada bahasa yang akan diterjemahkan dan bahasa yang akan dihasilkan. Selanjutnya, kita akan merujuk pada kedua bahasa tersebut dengan sebutan bahasa sumber dan bahasa target.

Untuk memperoleh hasil yang benar, terdapat aspek-aspek yang perlu diperhatikan. Pada dasarnya, aspek penting dalam penerjemahan terdiri atas tata bahasa dan semantik dari masing-masing bahasa sumber dan bahasa target. Tata bahasa menentukan aturan mengenai bagaimana sebuah bahasa itu ditulis atau digunakan, sedangkan semantik memberikan arti dari bahasa tersebut.

Dalam kaitannya dengan bahasa pemrograman, tata bahasa mendefinisikan aturan-aturan yang ketat dan kaku mengenai bagaimana layaknya kita menyusun suatu bahasa. Suatu bahasa tidak diperkenankan untuk memiliki arti yang ambigu, di mana suatu kalimat dalam bahasa tertentu hanya boleh memiliki satu dan hanya satu arti. Oleh karena itu, kita mendapatkan semantik yang unik untuk masing-masing kalimat yang unik.

Di sisi lain, bahasa-bahasa komunikasi seperti: Bahasa Inggris, Bahasa Indonesia, dan sebagainya, memiliki fleksibilitas lebih pada tata bahasanya. Meskipun kita melanggar aturan tata bahasa tertentu pada bahasa komunikasi, terkadang masih dimungkinkan untuk memahami arti dari kalimat tersebut. Ini adalah alasan mengapa bahasa komunikasi dikenal juga sebagai bahasa yang ambigu, di mana persepsi dari suatu kalimat antara satu pihak dengan pihak yang lain bisa berlainan.

Jadi, kita perlu mengerti tata bahasa dan semantik dari tiap-tiap bahasa secara mendalam sebelum kita dapat menerjemahkannya. Langkah pertama dalam penerjemahan adalah memahami arti dari apa yang hendak kita terjemahkan. Langkah ini dilakukan dengan mencocokkan masukan yang ada dengan tata bahasa dan semantik dari bahasa sumber. Kemudian arti yang diperoleh disusun ke dalam bahasa target berdasarkan tata bahasa dari bahasa target untuk merepresentasikan semantik yang sesuai. Langkah ini harus dilakukan secara berhati-hati untuk menghasilkan arti yang sama antara bahasa sumber dan bahasa target.

Penerjemah Lingu ke Java

Lingu adalah sebuah bahasa tingkat tinggi yang difokuskan pada transaksi basis data. Walaupun demikian, apabila dibandingkan dengan bahasa SQL, Lingu adalah bahasa yang lebih sederhana dan berbeda dari cara penulisan. Akan tetapi, Lingu menawarkan beberapa fitur-fitur yang dapat dipertimbangkan sebagai alternatif yang menarik, antara lain:

1. Lingu merupakan bahasa abstrak sehingga memungkinkan logika pada Lingu pada tingkatan yang lebih tinggi.
2. Lingu merupakan bahasa yang kecil dan sederhana. Dengan kesederhanaan Lingu, bahasa ini lebih mudah digunakan.

3. Lingu memiliki dukungan untuk uji coba dan validasi. Hal ini memberikan opsi untuk menjaga kesesuaian program yang dibuat dengan fungsi yang didefinisikan.
4. Lingu dapat diverifikasi dengan menghasilkan kondisi-kondisi verifikasi yang dapat dibuktikan dengan menggunakan alat bantu pembuktian teori (*theorem prover*).

Fitur-fitur yang diberikan Lingu memungkinkan penggunaannya sebagai landasan spesifikasi untuk suatu sistem. Dengan adanya kemampuan untuk memverifikasi dan mengujicobakan Lingu, maka implementasi sistem, terutama yang sifatnya kritikal, mampu dijamin validitasnya. Walaupun demikian, Lingu merupakan bahasa yang tidak dapat dieksekusi. Oleh karena itu, dibutuhkan transformasi dari bahasa Lingu di tingkat abstrak menjadi bahasa yang sifatnya kongkrit, yaitu program komputer. Java diambil sebagai pilihan untuk bahasa target dari hasil transformasi Lingu.

Ide pengembangan penerjemah Lingu ke Java adalah untuk mengotomatisasi transformasi bahasa Lingu menjadi bahasa Java. Lingu dapat digunakan untuk melakukan verifikasi terhadap program yang hendak dibuat. Dengan menggunakan penerjemah Lingu ke Java, spesifikasi yang sudah terbukti kebenarannya dapat dihasilkan program Java. Idealnya, program yang dihasilkan tidak terpaku pada bahasa Java saja. Fokus sistem hanya bertumpu pada spesifikasi yang dituliskan dalam Lingu dan kemudian dapat direalisasikan ke dalam berbagai macam bahasa pemrograman sebagai bentuk konkrit sesuai dengan kebutuhan.

Penerjemah Lingu ke Java dikembangkan dengan menggunakan sistem UUAG sebagai alat bantu. UUAG merupakan sistem *attribute grammar* yang dihasilkan oleh Universitas Utrecht. Sistem tersebut membantu pengembangan aplikasi yang tergolong dalam kelompok kompilator yang mencakup aplikasi penerjemah. Hasil akhir implementasi adalah aplikasi dalam bahasa pemrograman Haskell.

Penerjemah Lingu ke Java dapat dimodifikasi untuk menghasilkan penerjemah-penerjemah lainnya yang berfungsi untuk menerjemahkan Lingu ke dalam bahasa-bahasa pemrograman lain. Hal ini dilakukan dengan hanya mendefinisikan aksi semantik penerjemahan untuk target bahasa yang baru dan kemudian digunakan sebagai pengganti definisi aksi semantik penerjemahan yang lama.

Isi Paket Perangkat Lunak

```
example
|-input.txt

generated
|-DBManager.java
|-Obj.java
|-VirtTable.java

src
|-LinguParser.hs
|-LinguScanner.hs
|-Main.hs
```

```
|-Prog.ag

tools
|-uuagc.exe

UUAG
|-Configuration
  |-...
|-SupportLib
  |-...
|-UU
  |-...
```

Catatan: Penggunaan penerjemah dilakukan dengan mengekstraksi paket yang tersedia. Akan tetapi, sebelumnya perlu dilakukan instalasi perangkat lunak pendukung yang dijabarkan pada petunjuk penggunaan di bagian berikut.

Petunjuk Penggunaan Penerjemah Lingu ke Java

Langkah-langkah penggunaan penerjemah adalah sebagai berikut:

1. Perangkat lunak yang harus tersedia
   a. *Interpreter* Haskell: Hugs 98
      Untuk menjalankan penerjemah Lingu ke Java diperlukan *interpreter* Haskell, seperti Hugs 98. Perangkat lunak Hugs 98 dapat di-*download* di situs Haskell (http://www.haskell.org).

   b. Kompilator UUAG
      Berkas .ag harus terlebih dahulu dikompilasi menggunakan kompilator UUAG menjadi berkas Haskell (.hs), sebelum penerjemah dapat dijalankan. Kompilator UUAG (uuagc.exe) telah tersedia dalam paket penerjemah dalam *folder* 'tools', atau dapat di-*download* dari http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem

   c. *Library* UUAG
      Penerjemah Lingu ke Java membutuhkan *library-library attribute grammar* yang dikembangkan oleh Universitas Utrecht. *Library* tersebut tersedia dalam paket penerjemah dalam *folder* 'UUAG', atau dapat di-*download* dari http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem
      Pastikan *library-library* ini termasuk dalam lokasi pencarian Hugs 98. Untuk mengatur lokasi pencarian Hugs 98, ketikkan perintah berikut ketika menjalankan Hugs:
      > `:set -P"`<lokasi1>`:`<lokasi2>`:…"`

   d. Java 2 Platform, Standard Edition (J2SE)
      Kompilator dan *interpreter* Java dibutuhkan untuk mengkompilasi hasil penerjemahan dan menjalankan hasil kompilasi tersebut. Perangkat lunak J2SE tersedia untuk *download* di situs Java Sun Microsystems (http://java.sun.com).

   e. *Library* untuk eksekusi hasil penerjemahan

Eksekusi hasil penerjemahan membutuhkan dukungkan beberapa berkas Java. Berkas-berkas tersebut diberikan pada *folder* 'generated'.

2. Kompilasi berkas .ag
   a. Kompilasi berkas Prog.ag yang terdapat pada *folder* 'src'
      > `uuagc -a --self Prog.ag`

   b. Hasil kompilasi: Prog.hs

3. Menjalankan penerjemah Lingu ke Java
   a. Jalankan Main.hs yang terdapat di *folder* 'src' menggunakan Hugs 98
      > `hugs -98 Main.hs`

   b. Untuk melakukan penerjemahan, gunakan perintah reducePrim.
      > `reducePrim "<nama_berkas>"`
      <nama_berkas> dapat berupa alamat lengkap apabila berkas tidak terletak pada posisi di mana Hugs dijalankan.

      Sebagai contoh, terjemahkan berkas input.txt
      > `reducePrim "input.txt"`

      Untuk berkas input dengan alamat lengkap dapat dituliskan sebagai berikut:
      - Pada sistem operasi Windows
        > `reducePrim "C:\\input.txt"`
      - Pada sistem operasi Linux
        > `reducePrim "/home/test/input.txt"`

Contoh Penerjemahan

Masukan:

```
type RegistrationTable = Record
       {| ID             :: String;
          Name           :: String;
          Sex            :: Integer;
          Category       :: Integer;
          StudyProgramme :: String;  |}

type AnswerFormTable = Record
       {| ID        :: String;
          Name      :: String;
          SheetCode :: String;
          Answer    :: String;  |}

type SETdb = Dbase
       {| SubmitTab          :: Table AnswerFormTable;
          MasterTab          :: Table RegistrationTable;
          UnknownAFormTab    :: Table AnswerFormTable;
          DoubleAFormTab     :: Table AnswerFormTable; |}

class SETutility (d::SETdb) {

  method filterUnknown(SubmitTab :: Table AnswerFormTable,
                       MasterTab :: Table RegistrationTable,
                       UnknownAFormTab :: Table AnswerFormTable
                       ) :: ()
  ids, okids :: Table {| ID :: String; |};
  do
  {
     ids := findAll s<-d.SubmitTab where T found s.ID,s.Name;
     insertAll i<-ids,r<-d.MasterTab where i.ID==r.ID to okids;
     delete ids where ids.ID in okids.ID;
     insertAlls<-d.SubmitTab, i <-ids where s.ID==i.ID to d.UnknownAFormTab;
    }

  method filterDouble(i :: Integer) :: ()
  do
  {
     ids := findAll s<-d.SubmitTab, r<-d.SubmitTab
            where s.ID==r.ID /\ s <> r found s.ID;
     insertAll s<-d.SubmitTab, i<-ids where s.ID==i.ID to d.DoubleAFormTab;
  }
}
```

Keluaran (AnswerFormTable.java):

```
public class AnswerFormTable
{private String ID ;
 public void setID(String ID)
 {
 this.ID = ID;
 }
 public String getID()
 {
 return this.ID;
 }
 private String Name ;
 public void setName(String Name)
 {
 this.Name = Name;
 }
 public String getName()
 {
 return this.Name;
 }
 private String SheetCode ;
 public void setSheetCode(String SheetCode)
 {
 this.SheetCode = SheetCode;
 }
 public String getSheetCode()
 {
 return this.SheetCode;
 }
 private String Answer ;
 public void setAnswer(String Answer)
 {
 this.Answer = Answer;
 }
 public String getAnswer()
 {
 return this.Answer;
 }}
```

Keluaran (RegistrationTable.java):

```
public class RegistrationTable
{private String ID ;
 public void setID(String ID)
 {
 this.ID = ID;
 }
 public String getID()
 {
 return this.ID;
 }
 private String Name ;
 public void setName(String Name)
 {
 this.Name = Name;
 }
 public String getName()
 {
 return this.Name;
 }
 private Integer Sex ;
 public void setSex(Integer Sex)
 {
 this.Sex = Sex;
 }
 public Integer getSex()
 {
 return this.Sex;
 }
 private Integer Category ;
 public void setCategory(Integer Category)
 {
 this.Category = Category;
 }
 public Integer getCategory()
 {
 return this.Category;
 }
 private String StudyProgramme ;
 public void setStudyProgramme(String StudyProgramme)
 {
 this.StudyProgramme = StudyProgramme;
 }
 public String getStudyProgramme()
 {
 return this.StudyProgramme;
 }}
```

Keluaran (SETdb.java):

```
public class SETdb
{private AnswerFormTable[] SubmitTab ;
 public void setSubmitTab(AnswerFormTable[] SubmitTab)
 {
 this.SubmitTab = SubmitTab;
 }
 public AnswerFormTable[] getSubmitTab()
 {
 return this.SubmitTab;
 }
 private RegistrationTable[] MasterTab ;
 public void setMasterTab(RegistrationTable[] MasterTab)
 {
 this.MasterTab = MasterTab;
 }
 public RegistrationTable[] getMasterTab()
 {
 return this.MasterTab;
 }
 private AnswerFormTable[] UnknownAFormTab ;
 public void setUnknownAFormTab(AnswerFormTable[] UnknownAFormTab)
 {
 this.UnknownAFormTab = UnknownAFormTab;
 }
 public AnswerFormTable[] getUnknownAFormTab()
 {
 return this.UnknownAFormTab;
 }
 private AnswerFormTable[] DoubleAFormTab ;
 public void setDoubleAFormTab(AnswerFormTable[] DoubleAFormTab)
 {
 this.DoubleAFormTab = DoubleAFormTab;
 }
 public AnswerFormTable[] getDoubleAFormTab()
 {
 return this.DoubleAFormTab;
 }}
```

Keluaran (SETutility.java):

```
public class SETutility
{private DBManager dbm;
 public SETutility()
 {this.dbm = new DBManager("lingusql");
  this.dbm.connect();}
 public void filterUnknown(AnswerFormTable[] SubmitTab,RegistrationTable[]
MasterTab,AnswerFormTable[] UnknownAFormTab)
 {VirtTable vtables = new VirtTable();
  vtables.clearColumnList();
  vtables.addColumn("ID","String");
  vtables.create("ids");
  vtables.clearColumnList();
  vtables.addColumn("ID","String");
  vtables.create("okids");
  this.dbm.findAll("SubmitTab as s", "ids", "T", "s.ID,s.Name");
  this.dbm.insertAll("ids as i,MasterTab as r", "okids", "i.ID = r.ID");
  this.dbm.delete("ids", "ids.ID in (select ID from okids)");
  this.dbm.insertAll("SubmitTab as s,ids as i", "UnknownAFormTab", "s.ID = i.ID");
  }
 public void filterDouble(Integer i,AnswerFormTable[] SubmitTab)
 {VirtTable vtables = new VirtTable();
  this.dbm.findAll("SubmitTab as s,SubmitTab as r", "ids", "s.ID = r.ID and s <>
r", "s.ID");
  this.dbm.insertAll("SubmitTab as s,ids as i", "DoubleAFormTab", "s.ID = i.ID");
  }}
```

Lampiran A: Grammar Lingu

Program →
    TypeDeclaration ClassDeclaration+*

TypeDeclaration →
    type *Conid* = *Type_TD*
Type_TD →
    Type_Record | *Type_Dbase*

ClassDeclaration →
    class *Conid* ([*Parameter (, Parameter)**]?)
    {
        *Method**
        *Validation**
    }

Method →
    method *Conid* ([*Parameter (, Parameter)**]?) :: *Type*
    *VariableDeclaration**
    do
    {
        *Statement**
    }

Validation →
    validation *Conid* ([*Parameter (, Parameter)**]?) :: *Type*
    {
        *VariableDeclaration**
        do
        {
            *Statement**
        }
        return *Expr* ;
        pre *Expr* ;
        post *Expr* ;
    }

Statement →
    Statement_FindAll | *Statement_InsertAll* | *Statement_Delete*
    | *Statement_Call* | *Statement_If* | *Statement_Find*

Statement_FindAll →
    Expr := findAll *Expr_Container* where *Expr* found *Expr* [, *Expr*]* ;

Statement_InsertAll →
    insertAll *Expr_Container* [, *Expr_Container*]* where *Expr* to *Expr* ;

Statement_Find →
    Expr := find *Expr_Container* where *Expr* found *Expr* otherwise *Expr* ;

Statement_Delete ->
    delete *Expr* where *Expr* in *Expr_Delete* ;

Statement_Call →
    call *Varid* ([*Argument (, Argument)**]?) ;

Statement_If →
    if (*Expr*)
    then {*Statement*+}
    else {*Statement*+}

Type →
    Type_Boolean | *Type_Integer* | *Type_String*
    | *Type_Record* | *Type_Table* | *Type_Dbase*

Type_Boolean → Bool
Type_Integer → Integer
Type_String → String

Type_Record →
    Record {| *Element*+ |}

Type_Table →
    Table {| *Element*+ |}
    | Table *Conid*

Type_Dbase →
    Dbase {| *Element*+ |}

Parameter →
    identifier :: *Type*

Element →
    identifier :: *Type* ;

VariableDeclaration →
    identifier :: *Type* ;

Argument →
    identifier

Expr → *Lev0*

Lev0 → *Lev1* | *Lev1* [*Operator Lev1*]+

Lev1 →
    Constant | *VarName* | *RecordApp*
    | *UnaryExpr* | *ParenthesizedExpr*

Constant →
    BoolConstant | *IntConstant*
BoolConstant →
    T | F
IntConstant →
    integer

VarName →
    identifier

RecordApp →
    identifier [. *identifier*]+

UnaryExpr →
    Operator+ *Expr*

ParenthesizedExpr →
    (*Expr*)

Operator →
    + | - | * | / | ∧ | ∨ | == | <> | < | > | >= | <= | # | !! | <-

Expr_DB →
    identifier | *identifier* . *idenfier*
Expr_Container →
    Varid <- *Expr_DB*
Expr_Delete ->
    identifier | *identifier* . *idenfier*