COMPONENT SOFTWARE: ABSTRAKSI VS IMPLEMENTASI

Prastudy Mungkas Fauzi
Fakultas Ilmu Komputer, Universitas Indonesia
e-mail: prastudy.mungkas@mhs.cs.ui.ac.id

ABSTRAKSI

Component Software, atau perangkat lunak berbasis komponen, adalah sebuah konsep dimana perangkat lunak dapat dibangun dengan mengkomposisikan komponen-komponen yang sudah tersedia. Komponen-komponen tersebut dapat dibangun oleh pihak ketiga, dimana pihak yang ingin menggunakan komponen tersebut hanya perlu menghetahui spesifikasinya, tanpa perlu mengetahui seperti apa implementasinya. Dalam hal ini, proses membangun sebuah perangkat lunak akan melibatkan pembuatan abstraksi dari perangkat lunak terlebih dahulu, sebelum kemudian dibuat implementasinya. Tingkat kesulitan untuk mengubah sebuah abstraksi menjadi implementasi yang konkrit masih perlu dipertanyakan. Paper ini akan mencoba memperkirakan tingkat kesulitan tersebut dengan sebuah studi kasus, yaitu perbandingan sebuah abstraksi sistem electronic voting (E-Voting) dalam bahasa UNITY dengan sebuah implementasi dari sistem tersebut dalam JavaBeans.

Kata kunci: *Component Software, JavaBeans, UNITY*

1. PENDAHULUAN

Teknologi *component software* [6] adalah trend yang menjanjikan dalam pengembangan perangkat lunak secara cepat. Hal ini disebabkan oleh sifat *reusability* dari sebuah komponen, dimana komponen, baik yang telah diciptakan sebelumnya maupun yang diperoleh secara independen, dapat digunakan kembali dan dikomposisikan menjadi aplikasi yang diinginkan [1] .

Salah satu hal yang sangat penting dalam teknologi *component software* adalah masalah abstraksi. Dengan sebuah abstraksi dari komponen (atau sistem yang terdiri dari kumpulan komponen), dapat dilakukan pembuktian kebenaran dari komponen tersebut. Selain itu, seandainya terdapat pemetaan yang jelas antara abstraksi dengan implementasi sebuah sistem, maka terdapat kemungkinan untuk menciptakan *tools* sehingga sebuah sistem dapat diciptakan secara *automated* dengan hanya memberikan abstraksi dari sistem tersebut. Oleh karena itu, pengetahuan tentang bisa atau tidaknya sebuah abstraksi diubah menjadi implementasi dengan prosedur yang standar akan sangat penting bagi perkembangan *component software.*

Paper ini akan memberikan sebuah perbandingan antara abstraksi dengan implementasi dari sebuah aplikasi E-Voting. Abstraksi akan dilakukan dalam bahasa UNITY [3]. Sementara implementasinya menggunakan bahasa pemrograman Java, dalam bentuk JavaBeans [4]. Kita akan membahas tiap bagian dari abstraksi tersebut, dan melihat ada atau tidaknya bagian implementasi yang bersesuaian. Kemudian, akan diperlihatkan bagaimana *properties* yang diberikan pada abstraksi UNITY dapat dibuktikan dalam implementasi JavaBeans. Terakhir akan diberikan analisa mengenai perbandingan yang telah kita lakukan.

2. COMPONENT SOFTWARE - STATE OF THE ART

Penelitian-penelitian terakhir dalam bidang *component software* berkaitan erat dengan masalah sertifikasi perangkat lunak, sebuah proses yang menunjukkan secara formal bahwa program memenuhi aturan keamanan (*safety policy*) tertentu [7]. Berikut adalah beberapa penelitian yang signifikan:

1. Denney dan Fischer [7] mengembangkan framework dimana keamanan program secara dinamis menjamin keamanan program secara dinamis. Sebuah konsep umum mengenai *safety* dikembangkan, dengan membedakan *stateless properties* dan *stateful properties*. Kemudian, Denney dan Fischer mengembangkan definisi semantik dari keamanan, yang dapat digunakan untuk memeriksa kebenaran dan kelengkapan dari *safety policies*. Setelah itu, diberikan sebuah metode yang dapat mengembangkan aturan-aturan Hoare untuk menjelaskan *safety property* sembarang.
2. Denney dan Fischer [8] mencoba melakukan penggabungan antara sertifikasi dengan *automated code generation* dan metode verifikasi formal. Anotasi logika dihasilkan bersamaan dengan kode program, sehingga pembuktian dapat dilakukan secara otomatis. Pembuktian ini menjamin bahwa selama dieksekusi, kode yang dihasilkan tidak melanggar kondisi yang ditetapkan.
3. Mizugushi dan Matsuoka bersama CVS-AIST [9] menyelenggarakan penelitian mengenai verifikasi formal untuk perangkat lunak dengan dua arah penelitian : teoritis dan *case studies*. Arah teoritis berkaitan dengan metode formal, seperti *model checking* dan *theorem proving*. Sedangkan arah penilitian lainnya berkaitan dengan *case studies* mengenai metode verifikasi formal

4. Chang, Chipala, dan Necula [10] memberikan sebuah arsitektur untuk *Certified Program Analysis* yang terdiri dari 2 tahap: tahap instalasi, yang terjadi hanya sekali, serta tahap verifikasi, yang terjadi untuk tiap program input.

Penelitian-penelitian tersebut mengindikasikan bahwa terdapat kecenderungan untuk menggunakan metode pembuktian formal untuk melakukan sertifikasi *component software*.

3. PERBANDINGAN ABSTRAKSI DAN IMPLEMENTASI PADA KASUS E-VOTING

Sebuah sistem yang menangani E-Voting dapat diformulasikan dengan UNITY sebagai berikut [2, 3]:

```
System ElectronicVoting:
Component:
v :: Voting
t :: Timing
Constant:
LIMIT = 1000
Script:
t.tick > LIMIT -> v.stopCounting()
|| t.tick = NOTIME -> t.start()
Method:
enter = t.tick ~= NOTIME -> v.enter
report = v.report
Property:
[1] wstable t.tick > LIMIT /\ V supseteq
v.votes
[2] t.tick > LIMIT |--> v.stopFlag
[3] forall X. X ~= NOTIME : t.tick = X |-
-> t.tick > X
[4] t.tick = NOTIME => v.votes = []
```

Pada penelitian sebelumnya (*Case Study: Aplikasi Pemilihan Umum secara Elektronik*. Muhammad Fahrian, Prastudy Fauzi, Hanson. Penelitian Hibah B untuk Fakultas Ilmu Komputer UI, 2005) [5] telah dikembangkan sebuah simulasi sistem *electronic voting* yang terdiri dari 4 jenis *beans*:

1. Timing Bean
   Komponen yang mengatur, kapan *vote* yang masuk mulai bisa dikatakan *valid*, dan kapan berakhir.
2. Central Bean
   Komponen yang mengumpulkan bukti validasi dari setiap Calculator Bean dan memberikan hasil paling lengkap.
3. Calculator Bean
   Komponen yang melakukan pengujian validasi *vote* yang masuk dan penghitungan suara dari setiap voting atau Calculator Bean lain yang terhubung. Hasil perhitungan dan verifikasi dapat dikirimkan ke Central Bean atau ke Calculator Bean lainnya.
4. Voting Bean
   Komponen yang mewakili elektronik terminal tempat pemilihan suara dilakukan. Komponen dapat melakukan pengujian apakah *vote* yang masuk adalah *valid*.

Setelah dilakukan sedikit modifikasi dari *beans* tersebut, dapat terlihat banyak kesamaan antara formulasi UNITY, yang melibatkan komponen Voting dan Timing, dengan implementasi JavaBeans, terutama Voting Bean dan Timing Bean. Perbandingan antara bentuk UNITY dan bentuk JavaBeans adalah sebagai berikut:

A. Components

```
Component:
v :: Voting
t :: Timing
```

Voting Bean dan Timing Bean berturut-turut bersesuaian dengan komponen Voting dan Timing.

B. Variables

```
Constant:
LIMIT = 1000
```

Dalam implementasi JavaBeans, tidak diberikan limit waktu tertentu dalam melakukan *voting*. Waktu akan selesai bila tombol “Stop” pada Timing Bean ditekan.

C. Script

```
Script:
t.tick > LIMIT -> v.stopCounting()
|| t.tick = NOTIME -> t.start()
```

`t.tick > LIMIT` dapat dibandingkan dengan *method* `endVoteTime()` pada Timing Bean

```
public void endVoteTime(TimeListener
listener){
    Calendar now = getTime();
    TimeEvent evt = new TimeEvent(now,
    this, 3);
        listener.handleTime(evt);
}
```

Perhatikan bahwa *method* `getTime()` adalah implementasi dari `t.tick`.
`v.stopCounting()` dalam Voting Bean merupakan sesuatu yang dilakukan bila terjadi TimeEvent dari Timing Bean, yang dilakukan pada *method* handleTime(). Dalam hal ini, tipe event “1” berarti *vote* baru dimulai, sementara tipe event “3” berarti waktu untuk melakukan *vote* telah selesai.

```
public synchronized void
 handleTime(TimeEvent te) {
// bila waktu pemilihan habis,
// berhenti melayani vote
    int type = te.getType();
    if ( type == 1 ) {
     validTime = true;
```

```
     textField1.setText( " Please do
     your vote ");
    current = te.getDate();
    }else if (type == 3){
    textField1.setText( " Vote time
     ended.");
    validTime = false;
    }else{
     current = te.getDate();
     }
}
```

Implementasi dari `t.start()` dalam Timing Bean terjadi bila tombol "Start" (button1) ditekan, kemudian dalam `button1ActionPerformed()` dilakukan `startVoteTime()`.

```
public void StartVoteTime(
 TimeListener listener){
    Calendar now = getTime();
    TimeEvent evt = new
  TimeEvent(now, this, 1);
    listener.handleTime(evt);
}
```

Perhatikan bahwa tipe event "1" menandakan bahwa waktu untuk melakukan *vote* baru dimulai.

D. Methods

```
Method:
enter = t.tick ~= NOTIME -> v.enter
report = v.report
```

`v.enter` dalam Voting Bean terjadi bila terdapat *vote* yang masuk. Implementasinya dapat dilihat dari *method* `generateVote()` dan `fireVote()`

```
public void generateVote( String
 voteId , int cand ){
   votingIdentifier = voteId;
   String send = sender_id +"|" +
     votingIdentifier+ "|"+cand;
  //asumsi vote id memakai nomor saja

  if(validID(voteId) && notVoted(
    voteId) ) {
    record.add( voteId );
    fireVote( send );
  }
}

public void fireVote(String message){
   timer.handleAskTime( new
     AskTimeEvent(this) );
   VoteEvent evt = new
       VoteEvent(current, message,
       this);
   listener.handleVote(evt);
}
```

Sementara itu, `v.report` merupakan suatu *method* yang memberikan informasi mengenai *vote* yang masuk, dan biasanya cukup dilakukan satu kali saat *vote* telah berakhir. Dalam implementasi JavaBeans, *reporting* dilakukan setiap saat, dan informasi mengenai *vote* yang masuk di-*update* tiap terjadi event *vote* masuk. *Reporting* ini dilakukan pada *method* `update()` pada Calculator Bean (setelah melakukan `handleVote()`) dan method `update()` pada Central Bean (setelah melakukan `handleCalculation()`) . Misalnya pada Calculator Bean:

```
public synchronized void handleVote(
 VoteEvent ve) {

   // check sender ID sudah pernah vote
   // atau belum

   if(isInvalid(ve.getVotingID()))
     return;

   // check apakah tanggal valid atau
   // tidak
        if(!isValidDate(ve.getDate()))
    {
       textField3.setText("Time is
          invalid");
       return;
    }

    // tambahkan vote ID ke daftar vote
    // ID yang sudah mendaftar

    int v = Integer.parseInt(
       ve.getVote() );
    totalCalc[v-1]++;
    record.add(ve.getVotingID());
    update();
    textField3.setText("A vote was
       accepted");

    //langsung kirim lagi ke central

    timer.handleAskTime( new
       AskTimeEvent(this) );
    String message = ve.getSender()+"|"
       + ve.getVotingID()
       +"|"+ve.getVote() ;
    central.handleCalculation(new
       CalculationEvent(  current,
          message,this ));
}

public void update(){

javax.swing.SwingUtilities.invokeLater(
   new Runnable()
  {
     public void run(){
     String total = "";
     for (int i=0; i<totalCalc.length;
      i++){
       total+=(i+1)+"\t";
      }

     textField1.setText(total);
     total = "";
     for (int i=0;
      i< totalCalc.length; i++){
       total += totalCalc[i]+"\t";
      }

     textField2.setText(total);
     }
    });
 }
```

E. Properties

```
Property:
[1] wstable t.tick > LIMIT /\ V
supseteq v.votes
[2] t.tick > LIMIT |--> v.stopFlag
[3] forall X. X ~= NOTIME : t.tick = X
|--> t.tick > X
[4] t.tick = NOTIME => v.votes = []
```

Untuk melihat apakah pembuktian *properties* di atas dapat dilakukan secara informal pada JavaBeans, kita akan melihat makna dari tiap *properties*.

1. `wstable t.tick > LIMIT /\ V supseteq v.votes`
   Property ini menyatakan bahwa bila keadaan `t.tick > LIMIT /\ V supseteq v.votes` bernilai benar, maka untuk selanjutnya akan terus bernilai benar. Artinya, bila waktu telah melebihi batas yang ditentukan, maka akan terus demikian, dan waktu tidak dapat di-reset. Selain itu, *vote* yang masuk tidak dapat bertambah dari yang diperoleh pada saat waktu pemilihan telah habis. Tentu saja, jumlah *vote* dapat berkurang bila ditemukan *vote* yang tidak valid.

   Untuk menunjukkan hal tersebut pada JavaBeans, *method* dan *variable* yang bersesuaian adalah:
   Pada Timing Bean: `getTime()`
   Pada Voting Bean: `button1ActionPerformed(), fireVote()`
   Pada Calculator Bean: *method* `handleVote()`, array integer `totalCalc[]`
   Pada Central Bean: *method* `handleCalculation()`, array integer `totalVote[]`

   Perhatikan bahwa `t.tick` bersesuaian dengan *method* `getTime()` pada Timing Bean, sehingga cukup ditunjukkan bahwa bila waktu yang didapat pada `getTime()` melebihi LIMIT untuk suatu konstanta LIMIT, maka akan selalu bersifat demikian (bandingkan dengan *property* 3).
   Untuk menunjukkan bahwa jumlah *vote* tidak akan bertambah, harus ditunjukkan bahwa penambahan *vote* hanya dapat terjadi dengan adanya pemanggilan *method* `fireVote()` pada saat tombol "Vote" ditekan pada Voting Bean (menyebabkan pemanggilan *method* `button1ActionPerformed()`), yang di dalamnya memanggil *method* `handleVote()` pada Calculator Bean, yang di dalamnya memanggil *method* `handleCalculation()` pada Central Bean. Padahal, jumlah *vote* hanya dapat berubah pada `handleCalculation()` tersebut.

2. `t.tick > LIMIT |--> v.stopFlag`
   Property ini menyatakan bahwa kondisi `t.tick > LIMIT` akan menyebabkan dijalankannya `v.stopFlag`. Artinya, bila waktu pemilihan telah habis, ada *flag* yang menyatakan waktu tidak valid lagi.

   Pada JavaBeans, *method* dan *variable* yang bersesuaian adalah:
   Pada Timing Bean: *method* `endVoteTime()`
   Pada Voting Bean: *method* `handleTime()`, boolean `validTime`

   Untuk membuktikan kebenaran dari *property* tersebut, harus dibuktikan bahwa *method* `endVoteTime()` menyebabkan pemanggilan `handleTime()`, pada Voting Bean. Dalam *method* `handleTime()` nilai `validTime` di-set menjadi *false*. `validTime` ini dapat dikatakan sebagai *flag* yang menyatakan waktu telah tidak valid lagi.

3. `forall X. X ~= NOTIME : t.tick = X |--> t.tick > X`
   Property ini menyatakan bahwa `t.tick` akan selalu bertambah nilainya.

   Pada JavaBeans, method yang bersesuaian adalah `getTime()` pada Timing Bean.

   Untuk menujukkan kebenaran dari *property* tersebut, waktu yang dihasilkan oleh *method* `getTime()` harus dibuktikan selalu bertambah nilainya, dan hal ini bergantung pada implementasi pada class Calendar, dimana waktu berhubungan dengan banyaknya waktu yang berlalu sejak *Epoch*, January 1, 1970 00:00:00.000 GMT.

4. `t.tick = NOTIME => v.votes = []`
   Property ini menyatakan bahwa pada saat awal, *vote* yang masuk masih kosong.

   Pada JavaBeans, method yang bersesuaian adalah:
   Pada Timing Bean: `startVoteTime()`
   Pada Voting Bean: `button1ActionPerformed(), fireVote()`
   Pada Central Bean: constructor `Central()`, array integer `totalVote[]`

   Untuk menunjukkan *property* di atas, harus ditunjukkan bahwa pada awalnya `totalVote[]` adalah kosong (tiap indeks bernilai 0), kemudian hal ini tidak akan mungkin berubah sebelum dilakukan `fireVote()` pada Voting Bean. Padahal, `fireVote()` tidak akan dijalankan pada *method* `button1ActionPerformed()`, kecuali waktu adalah *valid*, sementara waktu tidak akan bernilai *valid* sebelum pemanggilan *method* `startVoteTime()` pada Timing Bean

4. ANALISA

Perbandingan yang telah kita lakukan di atas menunjukkan banyaknya masalah yang dihadapi

dalam pemetaan bentuk abstraksi program menjadi bentuk konkrit dalam JavaBeans. Masalah-masalah tersebut dapat dikategorikan sebagai berikut:

1. Masalah penamaan
   Salah satu masalah terbesar dalam melakukan perbandingan tersebut adalah adanya perbedaan nama variabel atau *method* pada abstraksi dengan nama variabel atau *method* yang bersesuaian pada implementasi.
   Hal ini akan dapat diatasi bila digunakan konvensi bahwa nama-nama variabel atau *method* pada implementasi harus sama dengan yang ada pada abstraksi.
2. Ketidaksesuaian tipe data
   Pada abstraksi dalam UNITY, tidak dispesifikasikan tipe data apa yang digunakan untuk variabel-variabelnya. Hal ini dapat menimbulkan masalah pada implementasi. Misalnya, pada abtraksi E-Voting, `t.tick` hanya merupakan sebuah integer, namun implementasinya pada JavaBeans merupakan tipe Calendar. Sebenarnya ada banyak kemungkinan tipe data lain yang dapat merealisasikan `t.tick` tersebut, diantaranya GregorianCalendar dan Integer. Dari contoh ini terlihat bahwa terdapat kesulitan dalam menentukan tipe data mana yang harus digunakan.
   Salah satu kemungkinan untuk mengatasi hal tersebut adalah dengan memberikan gambaran tipe data yang dibutuhkan pada abtraksi dari sistem.
3. Masalah input/output
   Dalam membuat sebuah abstraksi, terdapat kesulitan dalam menjelaskan input yang masuk, serta output yang dihasilkan oleh sistem. Misalnya, `v.report` pada abstraksi E-Voting mempunyai implementasi yang cukup rumit berupa penulisan String pada sebuah JTextField. Penelitian ini belum menemukan solusi yang memadai untuk permasalahan ini.

Ada berbagai keterbatasan dalam penelitian ini. Pertama, bagian abstraksi yang diperlihatkan dalam melakukan perbandingan di atas adalah *code* secara umum, sementara untuk memberikan pemetaan yang lebih tepat diperlukan juga informasi mengenai *contract* masing-masing komponen pada sistem. Kemudian, karena hanya memperlihatkan sebuah studi kasus, hasil penelitian ini tidak dapat digeneralisasi dengan baik untuk kasus-kasus lainnya.

5. PENUTUP

Dalam tulisan ini ditunjukkan sebuah abstraksi dari sebuah aplikasi E-Voting dalam UNITY, yang kemudian dibandingkan dengan sebuah implementasi dalam JavaBeans. Perbandingan ini memperlihatkan bahwa terdapat kemungkinan untuk menguji kebenaran sebuah aplikasi JavaBeans dengan memanfaatkan metode pembuktian formal. Namun, kita bisa melihat masih terdapat banyak permasalahan dalam melakukan pemetaan bentuk abstraksi menjadi implementasi yang konkrit. Selain itu, penelitian ini hanya menggunakan satu buah studi kasus, dan belum mengetahui apakah studi kasus lainnya juga akan memberikan hasil yang baik dengan pendekatan metode formal.

Untuk ke depannya perlu diadakan studi kasus tambahan yang dapat memberikan gambaran lebih baik mengenai pantas atau tidaknya metode dalam penelitian ini dikembangkan lebih lanjut menjadi sebuah metodologi baru dalam pengembangan sebuah program. Selain itu, akan lebih baik bila ada penelitian lanjutan yang lebih menitikberatkan pada proses pembuktian yang lebih konkrit daripada yang diberikan pada tulisan ini.

Untuk pembaca yang tertarik dengan implementasi dalam JavaBeans yang dibahas dalam paper ini, source code dapat anda dapatkan di http://cvt.cs.ui.ac.id/files/EVoting.zip

PUSTAKA

[1] Azurat, A., Prasetya, I.S.W.B., Towards Reliable Component Software: Light-weight Formalism, *8th International QIR Proceeding*, 9-10 Aug 2005.

[2] Prasetya, I.S.W.B., Vos, T.E.J., Azurat, A., Swierstra, S.D., *A UNITY-based Framework towards Component Based Systems*, University of Utrecht, 2003.

[3] Chandy, K.M., Misra, J., *Parallel Program Design – A Foundation*, Addison-Wesley Publishing Company Inc., 1988.

[4] Vanhelsuwe, L., et al., *Mastering Javabeans*, Sybex Inc, 1997

[5] Azurat, A., *Laporan Akhir Penelitian Verifikasi dan Visualisasi Component Software*, Fakultas Ilmu Komputer UI, 2006

[6] Szyperski, C., *Component Software*, Addison-Wesley, 2002.

[7] Denney, E., Fischer, B., Correctness of Source-Level Safety Policies, *The 12th International FME Symposium*, 2003.

[8] Denney, E., Fischer, B., Software Certification and Software Certificate Management Systems, *ACM International Conference on Automated Software Engineering*, 2005.

[9] Mizugushi, D., Matsuoka, S., A Report of the Current Situation on Software Certification in Japan, ACM CERTSOFT06: *An International Workshop on Software Certification*, 2006.

[10] Chang, B.E., Chipala, A., Necula, G.C., A Framework for Certified Program Analysis and Its Applications to Mobile-Code Safety, Verification, *Model Checking and Abstract Interpretation (VMCAI)*, 2006.