BAB IV IMPLEMENTASI STUDI KASUS

Bab ini menyajikan pembahasan mengenai implementasi yang dilakukan terhadap studi kasus yang menjadi sasaran dari penelitian ini. Bagian Deskripsi Studi Kasus menjelaskan mengenai gambaran umum dari studi kasus yang didasarkan kepada penelitian yang dilakukan pada [5], sementara langkah-langkah implementasi dan eksperimen dijelaskan pada bagian Langkah-Langkah Implementasi.

Setiap tahap dalam implementasi studi kasus dijelaskan dalam bagian tersendiri, secara berturut-turut adalah bagian Spesifikasi yang memaparkan mengenai spesifikasi yang ditetapkan dari pekerjaan sebelumnya, bagian Penyesuaian yang membahas mengenai kemungkinan kecocokan implementasi verifikasi dengan T2, diikuti oleh bagian Implementasi Spesifikasi dalam T2 yang mendiskusikan mengenai implementasi dari spesifikasi untuk pengujian dengan T2. Di bagian terakhir adalah Pengujian dan Hasil yang memaparkan mengenai hasil dari proses verifikasi yang dilakukan dengan menggunakan T2.

4.1. Deskripsi Studi Kasus

Studi kasus yang ditetapkan adalah sebuah aplikasi pemungutan suara elektronik (E-Voting) [5] yang dibangun sebagai implementasi *component software* dengan JavaBeans [7]. Penelitian sebelumnya (*Penerapan Logika UNITY dalam Sertifikasi Perangkat Lunak Berbasis Komponen dengan Studi Kasus Aplikasi E-Voting*; Prastudy Mungkas Fauzi, 2007) [5] telah menetapkan spesifikasi untuk aplikasi tersebut dalam konteks sebagai *component software*. Spesifikasi yang ditetapkan dalam penelitian tersebut menjadi acuan dari spesifikasi yang diimplementasikan pada eksperimen, dengan beberapa spesifikasi lain ditambahkan secara khusus untuk penelitian ini.

Aplikasi E-Voting adalah sebuah aplikasi pemungutan suara yang bersifat *online* dan dibangun dengan pendekatan sebagai *component software*. Sebuah aplikasi E-Voting terdiri atas komponen-komponen yang dapat saling berkomunikasi, masing-masing diimplementasikan sebagai JavaBean. Masing-masing komponen dalam E-Voting dideskripsikan sebagai berikut.

1. Timing Bean – Komponen yang mengatur mengenai manajemen waktu dalam aplikasi E-Voting. Termasuk dalam fungsinya adalah menentukan waktu yang valid untuk sesi pemungutan suara.

2. Voting Bean – Komponen yang mewakili terminal tempat dilaksanakannya pemungutan suara. Komponen ini juga melakukan pengecekan mengenai valid atau tidaknya suara yang masuk ke dalam sistem. Sebuah aplikasi E-Voting dapat menggunakan satu atau lebih Voting Bean.

3. Calculator Bean – Komponen yang melakukan pengujian terhadap validasi suara yang masuk berikut perhitungannya. Komponen ini mengirimkan hasil perhitungan dan verifikasi ke Central Bean atau Calculator Bean lain. Sebuah aplikasi E-Voting dapat menggunakan satu atau lebih Calculator Bean.

4. Central Bean – Komponen yang berperan sebagai pusat data dari aplikasi E-Voting. Komponen ini menerima hasil validasi dan perhitungan, serta melakukan rekapitulasi hasil dari keseluruhan aplikasi E-Voting.

4.2. Langkah-Langkah Implementasi

Dalam penelitian ini, implementasi terhadap studi kasus dilakukan dengan langkah-langkah sebagai berikut.

1. Pemetaan spesifikasi dan komponen/method dari pekerjaan sebelumnya.

2. Penyesuaian (*preprocessing*) dari aplikasi studi kasus untuk keperluan verifikasi dengan T2.

3. Implementasi spesifikasi untuk pengujian dengan T2, dalam Java.

4. Eksperimen verifikasi aplikasi E-Voting terhadap spesifikasi yang telah diimplementasikan.

4.3. Spesifikasi

Bagian ini menjelaskan mengenai spesifikasi yang digunakan dalam pengujian terhadap studi kasus. Spesifikasi dalam penelitian ini digolongkan sebagai berikut:

1. Spesifikasi Awal – merupakan spesifikasi yang telah ditetapkan dalam pekerjaan sebelumnya, dinyatakan dalam UNITY [5].

2. Spesifikasi Tambahan – merupakan spesifikasi yang ditambahkan untuk keperluan penelitian ini, terdiri atas spesifikasi *method* berupa *precondition* dan *postcondition*.

Masing-masing spesifikasi dijelaskan dengan urutan berupa definisi dari spesifikasi yang bersesuaian, diikuti dengan penjelasan informal dari spesifikasi tersebut.

4.3.1. Spesifikasi Awal

Pada pekerjaan sebelumnya [5], telah ditetapkan spesifikasi dalam UNITY [16] terhadap aplikasi E-Voting. Spesifikasi yang telah ditetapkan tersebut didefinisikan sebagai berikut.

```
Component:
v :: Voting
t :: Timing
Constant:
LIMIT = 1000  //arbitrary
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 = []
```

Untuk kebutuhan verifikasi dalam penelitian ini, pemetaan dilakukan terhadap spesifikasi `Script` dan `Property` dari pekerjaan sebelumnya. Masing-masing dari spesifikasi ini kemudian diimplementasikan untuk keperluan verifikasi dengan T2.

Script di sini merupakan spesifikasi yang menyatakan bagaimana seharusnya sebuah aplikasi berjalan; dalam kasus E-Voting, hal ini meliputi jalannya aplikasi terhadap waktu. Di sisi lain, *property* adalah persyaratan atau kondisi yang dipenuhi seiring jalannya aplikasi. Secara sederhana, sebuah *script* dapat dipandang sebagai spesifikasi 'apa yang akan dijalankan oleh aplikasi', sementara *property* dapat dipandang sebagai 'apakah aplikasi berjalan dengan benar'.

Script

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

Script ini dapat dipandang sebagai dua bagian yang dipisahkan oleh operator (||), secara berturut-turut adalah `t.tick > LIMIT -> v.stopCounting()` dan `t.tick = NOTIME -> t.start()`.

Operator (||) memiliki makna bahwa masing-masing predikat dari spesifikasi tersebut berlaku secara *concurrent*; dalam satu waktu, kedua predikat tersebut harus dipenuhi oleh aplikasi yang diujikan. Pada spesifikasi *script* ini, dapat diperhatikan bahwa `t.tick` pada masing-masing predikat memiliki keadaan yang berbeda. Keadaan ini adalah bahwa `t.tick > LIMIT` pada bagian pertama, dan `t.tick = NOTIME` pada bagian kedua. Dalam keadaan ini, spesifikasi *script* dapat dipandang secara lebih sederhana sebagai disjungsi dari masing-masing predikat di dalamnya.

Di sini, `t.tick` menyatakan waktu yang berjalan dalam aplikasi E-Voting, dengan masing-masing konstanta NOTIME dan LIMIT menyatakan tahap-tahap seiring waktu berjalannya aplikasi E-Voting. Konstanta NOTIME menyatakan waktu pada awal pemilihan sebelum sesi pemungutan suara dimulai, sementara LIMIT menyatakan waktu pada akhir pemilihan di mana sesi pemungutan suara telah ditutup.

Secara informal, *script* tersebut menyatakan bahwa aplikasi E-Voting akan memulai pemungutan suara pada suatu saat setelah masa NOTIME, dan pemungutan suara akan dihentikan setelah waktu berjalan melewati masa LIMIT.

***Property* (1)**

```
wstable t.tick > LIMIT /\ V supseteq v.votes
```

Property (1) menyatakan spesifikasi terhadap jumlah suara yang masuk setelah berakhirnya masa pemungutan suara (atau secara teknis, setelah melewati masa LIMIT). Dalam spesifikasi ini digunakan *keyword* `wstable`, yang menyatakan bahwa spesifikasi ini bersifat *stable*. Dengan demikian, apabila *property* ini memiliki nilai kebenaran *true* pada keadaan awal, maka untuk seterusnya *property* ini akan terus memiliki nilai kebenaran yang sama untuk sembarang aksi yang dilakukan dalam aplikasi.

Predikat `t.tick > LIMIT` menyatakan kondisi waktu telah melewati masa LIMIT. Sementara, di sisi lain `V supseteq v.votes` dapat dinyatakan sebagai berikut:

Misalkan v sebagai sebuah *instance* dari komponen Voting, dan V adalah himpunan dari semua *vote* yang diperoleh. Dapat diperhatikan bahwa `V supseteq v.votes` menyatakan bahwa untuk sebuah himpunan V, maka V adalah superset atau ekuivalen dengan himpunan v.votes yang menyatakan *vote* yang diperoleh dari komponen `Voting`.

Dengan demikian, secara keseluruhan *property* ini menyatakan bahwa setelah waktu melewati LIMIT, jumlah keseluruhan *vote* yang masuk akan selalu lebih besar atau sama dengan jumlah *vote* yang masuk lewat sebuah *instance* dari komponen `Voting` dari aplikasi E-Voting yang bersesuaian.

***Property* (2)**

```
t.tick > LIMIT |--> v.stopFlag
```

Property (2) menyatakan bahwa apabila waktu telah melewati masa LIMIT, maka *stop flag* dari komponen `Voting` akan diaktifkan. Hal ini memiliki implikasi bahwa

komponen Voting harus memiliki mekanisme untuk mengaktifkan *stop flag*, yang pada gilirannya akan menghentikan proses penerimaan *vote* yang masuk.

Secara sederhana, *property* ini dapat dipandang sebagai persyaratan yang harus dipenuhi oleh aplikasi pada penutupan sesi pemungutan suara.

***Property* (3)**

```
forall X. X ~= NOTIME : t.tick = X |--> t.tick > X
```

Property (3) menjamin bahwa jalannya waktu dalam aplikasi E-Voting akan selalu bertambah. Di sini, variabel waktu dinyatakan dalam X, dengan atribut yang bersesuaian adalah `t.tick` yang berada di bawah komponen `Timing`. Dapat diperhatikan dari pernyataan tersebut bahwa apabila waktu tidak berada dalam keadaan NOTIME, maka nilai `t.tick` pada suatu waktu akan selalu lebih kecil dari `t.tick` pada sembarang waktu setelahnya.

Perlu diperhatikan bahwa di sini waktu tidak dapat di-reset. *Property* ini menjamin bahwa nilai `t.tick` akan terus bertambah, dan hal ini sesuai dengan keadaan bahwa waktu dalam aplikasi akan selalu berjalan maju.

***Property* (4)**

```
t.tick = NOTIME => v.votes = []
```

Property (4) menyatakan kondisi yang harus dipenuhi sebelum masa pengujian dimulai, yaitu pada masa NOTIME. Pada saat tersebut, harus dijamin bahwa himpunan *vote* yang telah dihitung adalah himpunan kosong. Secara sederhana, hal ini merupakan langkah untuk menjamin bahwa aplikasi tidak menerima *vote* sebelum sesi pemungutan suara dimulai.

4.3.2. Spesifikasi Tambahan

Selain spesifikasi awal yang telah ditetapkan pada pekerjaan sebelumnya, ditetapkan juga spesifikasi tambahan yang dirancang untuk keperluan penelitian ini. Spesifikasi tambahan dirancang dengan pendekatan yang lebih pragmatis, dibandingkan dengan spesifikasi awal yang lebih bersifat abstrak.

Perlu diperhatikan bahwa spesifikasi awal yang ditetapkan bersifat independen terhadap implementasi dalam JavaBeans. Hal ini terkait dengan definisi dari spesifikasi dalam UNITY yang bersifat abstrak dan tidak terkait proses implementasi. Di sisi lain, spesifikasi tambahan bergantung kepada implementasi dalam Java, atau secara khusus JavaBeans dalam studi kasus. Hal ini disebabkan oleh perbedaan tujuan dari masing-masing spesifikasi: spesifikasi awal ditetapkan dalam UNITY dengan karakteristik yang lebih abstrak dan tidak terkait dengan implementasi JavaBeans, sementara spesifikasi tambahan ditetapkan untuk keperluan verifikasi dengan T2 yang bergantung kepada *in-code specifications* sebagai media verifikasinya.

Di sini, spesifikasi tambahan dinyatakan sebagai *precondition* dan *post condition* dari masing-masing *method* yang bersesuaian. Masing-masing spesifikasi tambahan dijelaskan sebagai berikut.

Spesifikasi *Method*: `voting.generateVote(String voteId, int cand)`

Spesifikasi ini ditetapkan untuk method `generateVote(String voteId, int cand)` yang berada di bawah komponen `Voting`. Parameter `voteId` merupakan kode unik yang dimiliki oleh masing-masing pemilih, sementara parameter `cand` merupakan nomor kandidat yang akan dipilih pada sesi pemungutan suara.

```
{P}S{Q}
P: cand > 0 /\ cand <= CANDIDATE_MAX
S: vote.generateVote(String VoteId, int cand)
Q: t.tick ~= VOTE, v.votes = v.votes'
```

Spesifikasi ini dapat dipandang bahwa *method* `generateVote(String voteId, int cand)` memiliki *precondition* bahwa nomor urut kandidat yang menjadi

parameter harus berada dalam *range* yang telah ditetapkan. Selain itu, *method* ini juga memiliki *postcondition* bahwa apabila waktu pemilihan tidak tepat (dengan kata lain, tidak dalam waktu VOTE), maka jumlah *vote* yang masuk tidak akan bertambah.

Spesifikasi *Method*: `central.getTotalVote(int cand)`

Spesifikasi ini ditetapkan untuk *method* `getTotalVote (int cand)` yang berada di bawah komponen `Central`. *Method* ini mengembalikan hasil perhitungan dari jumlah *vote* yang diperoleh masing-masing kandidat, berdasarkan nomor urut kandidat yang menjadi parameter.

Spesifikasi ini dinyatakan sebagai *precondition* dan *postcondition* untuk method tersebut, dinyatakan sebagai berikut.

```
{P}S{Q}
P: cand >= 0 /\ cand < CANDIDATE_MAX
S: central.getTotalVote(int cand)
Q: votes >= 0
```

Precondition ditetapkan untuk menjamin bahwa nomor urut kandidat yang menjadi parameter harus berada dalam *range* nomor urut kandidat yang valid. *Method* ini melakukan akses langsung ke *array* sehingga pembatasan *range* dari parameter berada di antara 0 (inklusif) dan CANDIDATE_MAX (eksklusif).

Spesifikasi ini merupakan implikasi dari konjungsi spesifikasi awal pada *property* (2) dan (4). Dari konjungsi kedua spesifikasi awal tersebut, dapat diturunkan keadaan yang setara dengan yang ditetapkan dalam spesifikasi ini. Perlu diperhatikan bahwa definisi spesifikasi ini dilakukan dalam pendekatan yang lebih pragmatis dari implementasi, dan hal ini membutuhkan pengetahuan terhadap beberapa detail dari proses implementasi (nama *method*, parameter dan *return type*). Dalam konteks ini, definisi spesifikasi dilakukan dengan pendekatan yang berbeda dibandingkan dengan spesifikasi awal – spesifikasi awal didefinisikan secara abstrak dan independen terhadap proses implementasi yang dilakukan, sementara definisi spesifikasi tambahan bergantung kepada implementasi dari aplikasi.

4.4. Penyesuaian

Aplikasi E-Voting adalah sebuah aplikasi berbasis Java yang dikembangkan sebagai *component software* dengan JavaBeans, dan dengan demikian verifikasi dengan T2 adalah hal yang memungkinkan untuk dilakukan. Meskipun demikian, perlu diperhatikan bahwa kapabilitas T2 yang telah diketahui sejauh ini tidak meliputi kemampuan analisis terhadap aplikasi Java yang dibangun sebagai *component software*.

Berdasarkan karakteristik dari JavaBeans yang merupakan turunan dari bentuk standar sebuah aplikasi Java, proses verifikasi dengan T2 merupakan hal yang dapat dilakukan dan sesuai dengan kapabilitas dari *tool* tersebut. Di sisi lain, perlu diperhatikan bahwa aplikasi E-Voting adalah sebuah aplikasi yang melibatkan fungsi-fungsi terkait *Graphical User Interface* dan *event handling*. Kapabilitas dari T2 tidak meliputi eksplorasi terhadap aspek-aspek tersebut, dan dengan demikian hal ini mungkin akan menimbulkan beberapa ketidaksesuaian dalam pelaksanaan proses verifikasi terhadap studi kasus.

Penjelasan mengenai ketidaksesuaian yang ditemukan pada proses verifikasi terhadap studi kasus dapat ditemukan pada bagian Ketidaksesuaian dan Pertimbangan, sementara penjelasan mengenai tindakan yang dilakukan untuk mengantisipasi ketidaksesuaian tersebut dapat ditemukan pada bagian Modifikasi untuk Verifikasi T2.

4.4.1. Ketidaksesuaian dan Pertimbangan

Secara umum, proses verifikasi yang dilakukan dalam penelitian ini memiliki kemungkinan-kemungkinan yang perlu diperhatikan, khususnya dari segi karakteristik aplikasi E-Voting yang tidak sama dengan aplikasi berbasis Java dalam bentuk umumnya. Perlu diperhatikan bahwa walaupun E-Voting adalah juga aplikasi berbasis Java, namun sifatnya sebagai *component software* dan utilisasi GUI memberikan pertimbangan tersendiri untuk verifikasi dengan T2.

Salah satu hal yang perlu diperhatikan adalah mengenai karakteristik dari aplikasi E-Voting yang terdiri atas komponen-komponen JavaBeans. Komponen-komponen ini

pada dasarnya bukanlah representasi dari sebuah aplikasi E-Voting yang utuh, dan dengan demikian hal ini tidak sesuai dengan karakteristik dari T2 yang melakukan verifikasi terhadap sebuah *class* yang diberikan. Proses verifikasi terhadap masing-masing *class* secara individual dapat dilakukan, namun dengan demikian lingkup pengujian hanya dapat dilakukan terhadap komponen yang bersangkutan.

Proses verifikasi dengan T2 untuk seluruh komponen ini kemudian dilakukan dengan penggunaan *mock class*. Dalam konteks ini, sebuah *mock class* adalah sebuah *class* yang mengandung *instance* dari masing-masing komponen dari aplikasi E-Voting. Spesifikasi *class invariant* dan *method specifications* juga didefinisikan dalam *mock class*, sebelum akhirnya *class* ini diajukan sebagai *class* yang diverifikasi oleh T2. Pembahasan lebih detail mengenai implementasi *mock class* disajikan pada bagian berikut dari laporan ini.

Selain itu, perlu diperhatikan bahwa kapabilitas T2 dalam melakukan verifikasi terhadap aplikasi yang menggunakan GUI masih belum diketahui secara mendalam. Meskipun demikian, hipotesis sementara adalah bahwa penggunaan GUI oleh aplikasi E-Voting tidak akan menghambat jalannya proses verifikasi dengan T2. Hipotesis tersebut diperoleh dengan pertimbangan bahwa verifikasi oleh T2 dilakukan berdasarkan pengecekan terhadap *method specifications* dan *class invariant.* Dengan spesifikasi yang dirancang dengan mengabaikan aspek GUI, proses verifikasi difokuskan kepada aspek logika dari aplikasi. Pada tahap ini, penggunaan GUI pada aplikasi tidak memiliki relevansi dalam konteks verifikasi yang dilakukan.

Hal lain yang perlu diperhatikan adalah masalah mengenai utilisasi *thread* pada aplikasi E-Voting. Sebagai sebuah aplikasi yang bekerja dengan utilisasi *event object* (`java.util.EventObject`) dan *event listener* (`java.util.EventListener`), aplikasi E-Voting menggunakan *thread* untuk menunggu setiap *event* yang mungkin terjadi selama aplikasi dijalankan. Hal ini diperkirakan akan memiliki dampak terhadap pelaksanaan verifikasi dengan T2; aplikasi mungkin tidak akan langsung berhenti setelah proses verifikasi selesai, dan terus menunggu *event* baru yang mungkin terjadi. Perlu diperhatikan bahwa di sisi lain T2 sendiri tidak memiliki kapasitas untuk melakukan terminasi aplikasi pada keadaan tersebut.

Meskipun demikian, hal ini diperkirakan tidak akan memiliki dampak yang akan mengganggu proses verifikasi oleh T2. Dalam keadaan ini diperkirakan T2 akan tetap mampu memberikan hasil verifikasi sesuai kapasitasnya, namun dengan catatan bahwa proses terminasi aplikasi harus dilakukan secara manual oleh pengguna.

4.4.2. Modifikasi untuk Verifikasi T2

Telah dijelaskan sebelumnya bahwa terdapat beberapa kemungkinan kekurangsesuaian dari aplikasi E-Voting untuk verifikasi dengan menggunakan T2. Dalam penelitian ini, masing-masing kemungkinan kekurangsesuaian tersebut diatasi dengan penyesuaian-penyesuaian yang akan dijelaskan pada bagian ini.

Implementasi *Mock Class*

Aplikasi E-Voting adalah sebuah aplikasi berbasis Java yang dikembangkan sebagai *component software* dengan JavaBeans. Masing-masing komponen merupakan bagian dari sebuah aplikasi E-Voting yang utuh, dan saling berkomunikasi dengan utilisasi *event object* dan *event listener* yang disediakan oleh Java. Perlu diperhatikan bahwa masing-masing komponen ini tidak merepresentasikan suatu aplikasi E-Voting; komponen-komponen ini harus diatur untuk dapat berkomunikasi satu sama lain. Meskipun demikian, masing-masing komponen ini tetap memiliki karakteristik dasar sebagai sebuah aplikasi berbasis Java.

Menyesuaikan dengan kapabilitas T2 yang melakukan verifikasi berdasarkan pengujian terhadap *sequence* pemanggilan *method* dari sebuah *class*, hal ini diatasi dengan pembuatan sebuah *mock class* yang dilakukan secara khusus untuk melakukan verifikasi dengan T2. Dalam *mock class* ini, dilakukan instansiasi dari masing-masing komponen dengan termasuk di dalamnya adalah fungsi-fungsi yang dimiliki oleh komponen yang bersesuaian.

Pendekatan dengan implementasi *mock class* ini didasari oleh keadaan bahwa T2 hanya dapat melakukan verifikasi terhadap satu buah *class* dalam satu waktu. Dalam keadaan ini, sebuah aplikasi E-Voting harus diinstansiasi dalam sebuah *class* yang

mewakili seluruh komponen berikut fungsi-fungsi terkaitnya. Perlu diperhatikan juga bahwa untuk dapat diverifikasi sebagai aplikasi E-Voting yang utuh, *class* ini harus memfasilitasi proses komunikasi antara masing-masing komponen yang menjadi bagian dari aplikasi tersebut. Keadaan ini diantisipasi dengan implementasi sebuah *mock class* yang memiliki *instance* dari masing-masing komponen E-Voting sebagai anggotanya. Implementasi *mock class* ini juga meliputi utilisasi *event object* dan *event listener* untuk komunikasi masing-masing komponen, namun dengan mengabaikan aspek-aspek terkait GUI pada aplikasi.

Pembuatan *mock class* dilakukan dengan mengabaikan aspek GUI yang dimiliki oleh komponen, dan dengan demikian eksplorasi terhadap fungsi yang dimiliki komponen dilakukan sebatas fungsi utama terkait logika aplikasi, di antaranya meliputi manajemen waktu pemungutan suara dan pengelolaan suara yang masuk ke dalam sistem.

Di sisi lain, pembuatan *mock class* ini dilakukan secara independen terhadap masing-masing komponen JavaBeans yang terkait. Masing-masing komponen tidak mengalami modifikasi terkait proses verifikasi pada penelitian ini, kecuali pada beberapa perubahan minor yang akan dijelaskan pada bagian berikutnya dari laporan ini.

Tambahan *Method* pada Komponen JavaBeans

Termasuk dalam penyesuaian yang dilakukan adalah perubahan yang dilakukan terhadap terhadap beberapa komponen dari aplikasi E-Voting. Penambahan fungsi ini dilakukan sehubungan dengan adanya beberapa fungsi yang diimplementasikan dalam bentuk *action listener* yang mengutilisasi aspek GUI dari aplikasi.

Sebagai contoh, fungsi untuk memulai dan menghentikan waktu pemungutan suara dituliskan sebagai fungsi yang bereaksi apabila pengguna melakukan klik pada *button* tertentu. Hal ini mengakibatkan ketidaksesuaian untuk proses verifikasi dengan T2, dan untuk mengantisipasi hal ini dibuat sebuah *method* tambahan dengan fungsi yang serupa.

Method-method tambahan inilah yang kemudian dipanggil dari *mock class*, dan dengan demikian fungsi verifikasi dari T2 melakukan akses terhadap *method* tambahan dan tidak terhadap fungsi yang di-*embed* pada *action listener*.

Berikut adalah *method* yang ditambahkan pada masing-masing komponen, untuk kebutuhan verifikasi dengan T2.

1. void timing.startStop()

method ini ditambahkan sehubungan dengan fungsi untuk memulai dan menutup sesi perhitungan suara sebelumnya di-*embed* pada implementasi dari *action listener* pada *button* 'Start/Stop'.

Untuk keperluan pengujian dengan T2, fungsi ini dipindahkan ke dalam *method* `timStartStop()` yang memiliki fungsi serupa, namun tidak menerima parameter berupa *action event* (`java.awt.event.ActionEvent`). Hal ini dilakukan untuk mengakomodasi karakteristik dari T2 yang tidak memiliki kapabilitas khusus untuk melakukan pengujian terhadap aspek GUI dan *event handling* dari aplikasi yang diujikan.

2. boolean voting.isValidTime()

Pada implementasi dari pekerjaan sebelumnya, nilai kebenaran dari *stop flag* yang dimiliki oleh komponen `Voting` diketahui dengan melakukan akses terhadap variabel publik. Hal ini dapat diakomodasi oleh T2, namun dengan kesulitan tersendiri dalam proses spesifikasi karena diperlukan pengetahuan mengenai proses internal dari komponen. Adanya kebutuhan terhadap pengetahuan terkait proses internal ini dapat mempersulit, khususnya dalam keadaan bahwa proses verifikasi akan dilakukan terhadap aplikasi yang telah dibangun sebelumnya – pada keadaan ini, umumnya akan dibutuhkan peninjauan terhadap *sourcecode* untuk menemukan variabel publik yang digunakan dalam fungsi yang bersesuaian. Di sisi lain, implementasi dengan teknik tersebut bertentangan dengan konsep *component software* yang dimiliki oleh aplikasi E-Voting; dalam konsep

component software, seharusnya tidak diperlukan pengetahuan mendalam mengenai karakteristik internal komponen.

Dengan pertimbangan tersebut, akhirnya ditambahkan sebuah *method* yang memiliki fungsi untuk mengakses *stop flag* pada *komponen* `Voting`. *Method* inilah yang kemudian akan diakses oleh *mock class* untuk verifikasi oleh T2.

Pemetaan *Method* pada *Mock Class*

Dari penjelasan pada bagian sebelumnya, telah ditetapkan bahwa proses verifikasi dilakukan dengan perantara sebuah *mock class*. *Mock class* inilah yang kemudian berkomunikasi dengan T2 dalam proses verifikasi.

T2 merupakan *verification tool* yang bekerja berdasarkan pengujian terhadap instansiasi *class* pemanggilan *method*. Dengan demikian, masing-masing *method* yang berada di bawah komponen dan terkait logika aplikasi harus dipetakan ke dalam *mock class*.

Pemetaan *method* pada *mock class* dinyatakan sebagai berikut.

Tabel 4: Pemetaan Method pada *Mock Class*

Komponen	Method	Pemetaan *Mock Class*	Fungsi
`Timing`	`startStop()`	`timStartStop()`	Memulai dan menghentikan sesi pemungutan suara
`Timing`	`getTime()`	`timGetTime()`	Mengembalikan *instance* dari waktu yang telah berjalan, representasi `java.util.Calendar`

`Voting`	`isValidTime()`	`votIsValidTime()`	Akses stopflag pada komponen Voting
`Voting`	`generateVote (String, int)`	`votGenerateVote (String, int)`	Memasukkan suara dari pemilih dengan ID tertentu
`Central`	`getTotalVote (int)`	`cenGetTotalVote(int)`	Mengembalikan perolehan suara kandidat dengan nomor urut tertentu
`Central`	`getTotalCount ()`	`cenGetTotalCount()`	Mengembalikan jumlah semua suara yang telah diterima aplikasi

Catatan:

- **pemetaan method mengabaikan aspek GUI dari komponen-komponen bersangkutan**
- **masing-masing komponen tidak mengalami modifikasi kecuali yang telah dijelaskan sebelumnya**

4.5. Implementasi Spesifikasi dalam T2

Masing-masing spesifikasi yang telah ditetapkan kemudian diimplementasikan dalam Java untuk keperluan verifikasi oleh T2. Proses implementasi terhadap spesifikasi meliputi definisi terhadap *class invariant* dan *method specifications* (meliputi *precondition* dan *postcondition*), serta predikat-predikat yang bersifat temporal (*temporal properties*). Khusus spesifikasi terkait *temporal properties*, implementasi dilakukan dengan pendekatan berupa pemetaan aplikasi ke dalam beberapa *state* yang berubah terhadap waktu.

4.5.1. Emulasi *State* untuk *Temporal Properties*

Sebagaimana telah dinyatakan dalam spesifikasi awal, aplikasi E-Voting memiliki tiga buah *state* yang berubah berdasarkan waktu. Masing-masing *state* tersebut dinyatakan sebagai berikut.

1. NOTIME – merupakan keadaan saat aplikasi E-Voting baru dijalankan; sesi pemungutan suara belum dimulai pada saat ini.
2. VOTE – merupakan keadaan saat aplikasi E-Voting memasuki sesi pemungutan suara; pada saat ini penerimaan dan penghitungan suara akan dilakukan.
3. LIMIT – merupakan keadaan saat sesi pemungutan suara telah ditutup; suara yang masuk pada saat ini tidak akan diperhitungkan.

Dalam konteks aplikasi E-Voting yang sebenarnya, *state* dari aplikasi akan mengalami transisi ketika tombol *Start/Stop* pada menu ditekan. Untuk keperluan verifikasi dengan T2, fungsi ini diwakilkan oleh sebuah method `timStartStop()` pada *mock class*. Method ini merepresentasikan fungsi tersebut, yang pada konteks aslinya dimiliki oleh Timing Bean.

Implementasi dari emulasi *state* pada aplikasi E-Voting dituliskan dalam *class invariant*, dan dinyatakan sebagai berikut.

```
    private boolean classinv() {

        if (aux_laststep.equals("timStartStop"))
            aux_count++;

        if (aux_count % 2 == 1)
            aux_state = VOTE;
        else if (aux_count % 2 == 0)
            aux_state = LIMIT;

        switch (aux_state) {
            case NOTIME :
                //specifications in NOTIME
```

```
            break;

            case VOTE :
                //specifications in VOTE

            break;

            case LIMIT :
                 //specifications in LIMIT

            break;
        }

        return true;
    }
```

Perlu diperhatikan bahwa variabel `aux_laststep` adalah variabel yang disediakan oleh T2 untuk menyimpan nama dari *method* terakhir yang dipanggil. Variabel ini dinyatakan sebagai sebuah String. Dua variabel lain adalah `aux_state` dan `aux_count`, masing-masing dinyatakan sebagai sebuah integer yang digunakan khusus untuk proses verifikasi dengan T2. Perlu juga diperhatikan bahwa kedua variabel ini tidak memiliki karakteristik khusus seperti halnya `aux_laststep`, kecuali dalam konteks sebagai *auxilliary variables* untuk keperluan pengujian dengan T2.

Pada implementasi *state* ini, masing-masing perubahan *state* dinyatakan untuk setiap kali pemanggilan *method* `timStartStop()`. Dapat diperhatikan bahwa setiap kali pemanggilan terhadap `timStartStop()` dilakukan, variabel `aux_count` akan di-*increment*. Setelah itu, dilakukan pengecekan dan *update* dari *state* berdasarkan nilai dari variabel `aux_count`.

Sehubungan dengan karakteristik aplikasi E-Voting yang memungkinkan sesi pemungutan suara dibuka ulang setelah ditutup, pendekatan terhadap emulasi *state* ini dilakukan berdasarkan hasil modulo dari besarnya nilai `aux_count` yang akan terus bertambah.

4.5.2. Implementasi per Spesifikasi

Bagian ini menjelaskan mengenai implementasi dari masing-masing spesifikasi yang telah ditetapkan, baik dari spesifikasi awal maupun spesifikasi tambahan. Spesifikasi yang ada diimplementasikan dalam Java, baik dalam *class invariant* maupun *method specifications*.

Script

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

Secara umum, spesifikasi ini menyatakan bahwa aplikasi akan berpindah *state* pada suatu saat setelah masa NOTIME dan akan berhenti melakukan penerimaan dan penghitungan suara setelah masa LIMIT.

Sebagaimana telah dijelaskan sebelumnya, operator (||) memiliki makna bahwa masing-masing predikat dari spesifikasi tersebut berlaku secara *concurrent*, yaitu kedua predikat tersebut harus dipenuhi dalam satu waktu. Namun dengan keadaan `t.tick` yang berbeda pada masing-masing predikat, secara sederhana hal ini dapat dipandang sebagai disjungsi dari masing-masing predikat pada *script* tersebut.

Untuk bagian [`t.tick > LIMIT -> v.stopCounting()`], *in-code specification* dinyatakan sebagai berikut. Spesifikasi berikut dinyatakan untuk keadaan setelah aplikasi E-Voting melewati masa LIMIT.

```
case LIMIT :

    //script #1: stop calculation by v past limit
    //property #2: raise stopflag immediately after limit
     assert !votIsValidTime() : "PROP#2";

     [...]

break;
```

Secara teknis, implementasi dari spesifikasi ini beririsan dengan *property* (2) yang menyatakan mengenai dijalankannya *stop flag* dari komponen `Voting`. Hal ini terjadi

karena perbedaan pendekatan antara spesifikasi yang dinyatakan secara formal dengan implementasinya yang diterjemahkan ke dalam pendekatan yang lebih teknis.

Di sisi lain, implementasi dari spesifikasi [`t.tick = NOTIME -> t.start()`] tidak dinyatakan dalam *in-code specification*. Hal ini dilakukan dengan asumsi bahwa satu siklus aplikasi E-Voting akan selalu memiliki perubahan *state* dari keadaan NOTIME, dan dengan demikian pernyataan spesifikasi ini sebagai *in-code specification* menjadi tidak signifikan. Meskipun demikian, perlu diperhatikan bahwa bagian ini tetap perlu dinyatakan untuk keperluan definisi spesifikasi secara formal, walaupun tidak signifikan dalam implementasi pada *in-code specification*.

***Property* (1)**

```
wstable t.tick > LIMIT /\ V supseteq v.votes
```

Property (1) menyatakan bahwa setelah masa LIMIT, himpunan V yang merepresentasikan keseluruhan suara yang masuk akan selalu berlaku sebagai superset atau ekuivalen dengan himpunan v yang merepresentasikan jumlah suara yang dihitung pada suatu *instance* komponen `Voting`.

In-code specification untuk *property* ini dinyatakan sebagai berikut.

```
case LIMIT :

[...]

     //property #1: votes are superset or equivalent past limit
     aux_votes = 0;
     for (int i=0; i<CANDIDATE_MAX; i++)
           aux_votes += cenGetTotalVote(i);
     assert cenGetTotalCount() >= aux_votes : "PROP#1";

[...]

break;
```

Pada tahap ini, *instance* dari komponen `Voting` ditetapkan terdapat hanya satu buah dalam aplikasi E-Voting. Dengan demikian, himpunan total *vote* masuk haruslah

merupakan superset atau sama dengan jumlah *vote* yang melewati *instance* dari komponen tersebut. Perlu diperhatikan bahwa mungkin saja terdapat suara yang tidak valid dan dibatalkan, dan dengan demikian *property* ini diformulasikan dengan menggunakan superset dalam definisinya.

***Property* (2)**

```
t.tick > LIMIT |--> v.stopFlag
```

Property (2) menyatakan bahwa apabila waktu dalam aplikasi E-Voting telah melewati masa LIMIT, maka sebuah *stop flag* akan diaktifkan, dan dengan demikian sesi pemungutan suara ditutup. Implementasinya dalam *in-code specification* dinyatakan sebagai berikut.

```
case LIMIT :

    //script #1: stop calculation by v past limit
    //property #2: raise stopflag immediately after limit
     assert !votIsValidTime() : "PROP#2";

     [...]

break;
```

Sebagaimana telah dijelaskan sebelumnya, implementasi dari *property* ini secara teknis beririsan dengan spesifikasi pada *script*. Di sini, dilakukan pengecekan mengenai aktivasi *stop flag* yang berada di bawah komponen `Voting`.

***Property* (3)**

```
forall X. X ~= NOTIME : t.tick = X |--> t.tick > X
```

Property (3) menyatakan bahwa waktu dalam aplikasi E-Voting akan selalu bertambah nilainya. Secara khusus, *property* ini menyatakan bahwa apabila waktu

yang telah lewat dinyatakan sebagai besaran dalam satuan waktu, maka nilai dari besaran tersebut akan bertambah pada suatu saat di masa setelahnya.

Hal yang perlu diperhatikan adalah bahwa spesifikasi ini menyatakan bahwa apabila diberikan sebuah nilai *x* untuk nilai waktu tertentu, maka di masa depan akan berlaku bahwa nilai `t.tick` akan bernilai lebih besar daripada *x*. Perlu diperhatikan bahwa hal ini tidak berarti bahwa nilai `t.tick` akan segera berubah setelah rentang waktu tertentu; spesifikasi ini menyatakan bahwa nilai `t.tick` akan lebih besar daripada *x* pada suatu saat setelahnya.

Secara umum, *property* ini memiliki kesulitan tersendiri dalam implementasinya; Java sebagai bahasa pemrograman tidak mendukung untuk definisi spesifikasi yang bersifat eventual (dalam konteks ini, bahwa nilai `t.tick` akan bertambah *suatu saat* setelah suatu waktu *t*). Dalam konteks ini, pendekatan yang paling memungkinkan adalah implementasi spesifikasi dengan definisi predikat yang lebih ketat bahwa nilai `t.tick` dipastikan akan bertambah pada *sembarang waktu* setelahnya.

```
private java.util.Calendar aux_cal =
java.util.Calendar.getInstance();

    private boolean classinv() {

    [...]

      switch (aux_state) {

        [...]

        case VOTE :

          //property #3: t.tick eventually increment outside NOTIME
          aux_cal = timGetTime();

        break;

              case LIMIT :

              [...]

        //property #3 with weaker assumption -- unused
        //assert !aux_cal.after(timGetTime()) : "PROP#3";

        //property #3: t.tick eventually increment outside NOTIME
        assert timGetTime().after(aux_cal) : "PROP#3";
        break;

    }
```

```
        return true;
    }
```

Pada pekerjaan sebelumnya, implementasi dalam aplikasi E-Voting dilakukan dengan menggunakan *class* `java.util.Calendar`. Pada penelitian ini, proses penghitungan waktu yang telah berlalu disesuaikan dengan keadaan tersebut. Untuk keperluan verifikasi dengan T2, *class* `java.util.Calendar` digunakan sebagai media untuk melakukan perhitungan waktu.

Pendekatan lain yang dapat dilakukan untuk implementasi dari *property* (3) adalah dengan negasi terhadap fakta bahwa waktu dapat berjalan mundur. Hal ini dapat diperhatikan pada potongan *code* di atas (secara khusus, pada komentar untuk spesifikasi *property* (3)).

```
assert !aux_cal.after(timGetTime()) : "PROP#3";
```

Pada pendekatan ini, spesifikasi dilakukan dengan asumsi yang lebih longgar bahwa 'waktu tidak dapat berjalan mundur' dibandingkan dengan makna dari spesifikasi bahwa 'waktu pasti akan berjalan maju'. Meskipun demikian, implementasi dengan asumsi ini tidak memenuhi makna dari definisi awal *property* (3) – secara umum predikat yang dihasilkan lebih lemah daripada definisi awal dari *property* (3). Hal ini mengakibatkan implementasi terhadap spesifikasi ini akhirnya dilakukan dengan asumsi yang lebih ketat sebagaimana disajikan dalam *code* yang disertakan.

***Property* (4)**

```
t.tick = NOTIME => v.votes = []
```

Pada *property* (4), dinyatakan bahwa suara yang masuk pada masa awal sebelum pemungutan suara dimulai (masa NOTIME) adalah himpunan kosong. Hal ini juga dapat dipandang bahwa suara yang masuk tidak akan diproses sebelum sesi pemungutan suara dimulai.

Implementasi dari *property* tersebut adalah sebagai berikut.

```
case NOTIME :

    //property #4: votes are empty upon NOTIME
    for (int i = 0; i < CANDIDATE_MAX; i++)
        assert cenGetTotalVote(i) == 0 : "PROP#4a";
    assert cenGetTotalCount() == 0 : "PROP#4b";

break;
```

Di sini, dilakukan dua kali pengecekan karena jumlah total suara dapat diperoleh dari dua *method* yang berbeda. Masing-masing *method* ini secara berturut-turut adalah `getTotalVote(int index)` dan `getTotalCount()` dari komponen `Central`.

***Added Specification* (1)**

```
{P}S{Q}
P: cand > 0 /\ cand <= CANDIDATE_MAX
S: vote.generateVote(String VoteId, int cand)
Q: t.tick ~= VOTE, v.votes = v.votes'
```

Spesifikasi ini ditetapkan untuk menjamin bahwa nomor kandidat yang dimasukkan berada dalam *range* yang sesuai. Selain itu, ditetapkan juga bahwa jumlah *vote* yang masuk tidak akan berubah apabila masa pemilihan sedang tidak dibuka (atau secara teknis, aplikasi tidak berada pada masa VOTE). Secara informal, hal ini memiliki makna bahwa suara yang diperhitungkan hanyalah suara yang masuk untuk kandidat dengan nomor yang valid pada saat yang diizinkan.

Sebagaimana telah dijelaskan pada bagian sebelumnya, spesifikasi tambahan ini dinyatakan sebagai penetapan *precondition* dan *postcondition* dari *method* `generateVote(String VoteId, int cand)` yang berada di bawah komponen `Voting`. Di sini P dan Q secara berturut-turut adalah *precondition* dan *post condition*, diterjemahkan dalam *in-code specification* sebagai berikut.

```
//added #1a: candidate ID is to be within range
assert cand > 0 && cand <= CANDIDATE_MAX : "PRE";
aux_voteMe = cenGetTotalVote(cand);
```

```
vote.generateVote(voteId, cand);

//added #1b: assures generateVote complies to stopflag
if(aux_state != VOTE)
     assert cenGetTotalVote(cand) == aux_voteMe : "POST";
```

***Added Specification* (2)**

```
{P}S{Q}
P: cand >= 0 /\ cand < CANDIDATE_MAX
S: central.getTotalVote(int cand)
Q: votes >= 0
```

Spesifikasi tambahan ini didefinisikan untuk memastikan bahwa hasil perolehan suara untuk masing-masing kandidat berada dalam *range* yang valid, yaitu tidak bernilai negatif. Dalam spesifikasi ini juga didefinisikan bahwa nomor kandidat yang menjadi parameter dari *method* ini harus berada dalam *range* yang valid. Secara informal, spesifikasi ini menyatakan bahwa untuk masing-masing kandidat dengan nomor urut yang valid, perolehan suara masing-masing harus berada dalam interval nilai yang valid pula.

Dalam spesifikasi ini, *method* yang menjadi fokus adalah `getTotalVote(int cand)` yang berada di bawah komponen `Central`. *Precondition* dan *post condition* dinyatakan secara berturut-turut oleh P dan Q pada spesifikasi di atas, dengan implementasi sebagai berikut.

```
//added #2a: candidate ID is to be within range
assert cand >= 0 && cand < CANDIDATE_MAX : "PRE" ;

int ret = cen.getTotalVote(cand);

//added #2b: votes are always non-negative
assert ret >= 0 : "POST";
```

Di sini, spesifikasi *precondition* memiliki tujuan yang serupa dengan *precondition* spesifikasi tambahan sebelumnya, yaitu untuk menjamin bahwa nomor urut kandidat berada dalam *range* yang sesuai. Sementara itu, *post condition* dari *method* ini ditetapkan untuk menjamin bahwa jumlah suara yang dikembalikan tidak bernilai negatif.

4.6. Pengujian dan Hasil

Setelah proses penyesuaian dan implementasi dari masing-masing spesifikasi untuk keperluan verifikasi dengan T2, aplikasi E-Voting dinyatakan siap untuk menjalani verifikasi. Bagian ini menjelaskan mengenai proses verifikasi berikut hasil pengujian yang dilakukan dengan T2. Termasuk dalam pembahasan pada bagian ini adalah mengenai konfigurasi yang dilakukan pada T2 untuk proses verifikasi yang dilakukan.

4.6.1. Konfigurasi pada T2

Sebagai sebuah *verification tool*, T2 memiliki set konfigurasi yang dapat dimodifikasi untuk keperluan pengujian. Konfigurasi ini menyangkut beberapa hal terkait proses pengujian; di antaranya adalah instansiasi objek dalam proses verifikasi, ruang *domain* untuk tipe data pengujian, dan ruang lingkup modifikasi nilai pada aplikasi.

Berdasarkan percobaan awal sebelum eksperimen, ditemukan bahwa terdapat konfigurasi yang perlu diperhatikan untuk menjamin hasil verifikasi yang valid dari pengujian studi kasus. Penggunaan konfigurasi pada T2 berikut penjelasannya dinyatakan sebagai berikut.

1. **`--nullprob=-1`**

   T2 melakukan verifikasi dengan mengeksekusi perintah yang dinyatakan dalam *constructor* dan *method* dari aplikasi. Meskipun demikian, T2 juga mengeksplorasi kemungkinan yang ada dengan melakukan instansiasi objek berupa *null*.

   Sebagai contoh, misalkan pada *class C* yang akan diverifikasi terdapat proses instansiasi dari *class D* berikut pemanggilan terhadap *method* dari *D*. Dalam keadaan *default*, T2 mungkin akan menginstansiasi *D* sebagai *null* dengan probabilitas tertentu. Dalam kasus seperti ini, pemanggilan *method* yang

berada di bawah *D* akan mengakibatkan terjadinya *violation* oleh aplikasi yang diverifikasi.

Perlu diperhatikan bahwa ketika *mock class* melakukan instansiasi terhadap komponen-komponen dari aplikasi E-Voting, *instance* berupa *null* akan menghasilkan *violation* berupa `NullPointerException` dalam proses verifikasi. Dengan asumsi bahwa sebuah *instance* dari komponen dalam sebuah aplikasi E-Voting tidak mungkin berupa *null*, maka eksplorasi terhadap kemungkinan ini menjadi tidak relevan untuk proses verifikasi pada penelitian ini.

Untuk kasus seperti ini, T2 menyediakan konfigurasi `--nullprob=`*`(float)`* untuk mengatur probabilitas instansiasi objek sebagai *null*. Konfigurasi ini menggunakan parameter sebuah bilangan *floating point* untuk mengatur probabilitas tersebut, dengan nilai negatif dimasukkan untuk meniadakan probabilitas tersebut.

Di sini, konfigurasi `--nullprob=-1` digunakan untuk meniadakan probabilitas instansiasi objek sebagai *null*, dengan asumsi yang telah dijelaskan sebelumnya.

2. `--excldefault`

Salah satu hal yang dilakukan oleh T2 dalam proses verifikasi meliputi modifikasi terhadap nilai dari *field* dan variabel yang dideklarasikan dalam suatu *class*. Hal ini tidak dilakukan untuk *access modifier* yang bersifat *static* atau *private*, namun di sisi lain hal ini mempengaruhi untuk keadaan di mana *access modifier* untuk suatu variabel bersifat *default*.

Dalam sebagian kasus, hal ini mengakibatkan hasil verifikasi yang tidak tepat karena terjadinya nilai yang tidak sesuai antar komponen dalam aplikasi. Untuk kasus seperti ini, T2 menyediakan konfigurasi `--excldefault` yang membatasi akses ke variabel dengan *access modifier* yang telah dijelaskan sebelumnya.

4.6.2. Hasil Pengujian

Berikut adalah hasil eksperimen yang dilakukan dengan verifikasi menggunakan T2. Secara umum, sebagian besar spesifikasi yang ditetapkan berhasil melewati pengujian dengan baik. Terdapat pengecualian pada *property* (3), namun hal ini bersifat teknis pada *library* Java yang digunakan.

Khusus untuk *property* (3), implementasi spesifikasi dengan asumsi yang lebih longgar memberikan hasil bahwa aplikasi dinyatakan valid – meskipun demikian, spesifikasi dalam eksperimen dilakukan dengan asumsi yang lebih ketat. Alasan untuk hal ini telah didiskusikan dalam pembahasan terkait implementasi spesifikasi pada bagian sebelumnya.

Tabel 5: Hasil Eksperimen terhadap Studi Kasus

Spesifikasi	**Hasil Verifikasi**	**Catatan**
Script	Valid	Implementasi beririsan dengan property (2)
Property (1)	Valid	
Property (2)	Valid	
Property (3)	Invalid	Library `java.util.Calendar` tidak memadai untuk pengujian property ini
Property (4)	Valid	
Added (1)	Valid	
Added (2)	Valid	Implikasi dari property (2) dan (4)

script pada T2:

```
java -ea -cp .;TT_0.4_silverlance.jar U2.T2.RndEngine
(MockClassName) --nullprob=-1 --excldefault
```

BAB V EVALUASI STUDI KASUS

Bab ini mendiskusikan mengenai evaluasi dan analisis dari eksperimen yang telah dilakukan terhadap studi kasus. Termasuk dalam pembahasan pada bagian ini adalah analisis mengenai kapabilitas T2 terhadap verifikasi dari studi kasus aplikasi E-Voting sebagai *component software* dan hal-hal lain yang terkait dalam proses verifikasi tersebut.

Secara umum, pembahasan pada bagian ini dibagi menjadi dua bagian utama. Di bagian pertama adalah Evaluasi Hasil Eksperimen yang menjelaskan mengenai evaluasi dan analisis terhadap eksperimen yang telah dilakukan pada tahap sebelumnya, meliputi spesifikasi yang diujikan dan hal-hal terkait yang ditemukan dalam proses verifikasi yang dilakukan. Di bagian kedua adalah Analisis Kapabilitas T2 terhadap Eksperimen Studi Kasus, yang menjelaskan mengenai analisis dari kapabilitas T2 berdasarkan hasil yang diperoleh dari eksperimen pada penelitian ini.

5.1. Evaluasi Hasil Eksperimen

Dari hasil eksperimen yang dilakukan terhadap studi kasus, dapat diperhatikan bahwa secara umum proses verifikasi dengan T2 memberikan hasil yang sesuai harapan, yaitu bahwa T2 dapat menyatakan valid atau tidaknya aplikasi studi kasus terhadap spesifikasi yang telah ditetapkan. Meskipun demikian, terdapat beberapa kendala yang perlu diperhatikan terkait proses verifikasi dengan T2 terhadap studi kasus.

Perlu diperhatikan juga bahwa terdapat penyesuaian-penyesuaian yang perlu dilakukan terhadap studi kasus untuk keperluan verifikasi dengan T2. Hal ini dilakukan sehubungan dengan karakteristik beberapa bagian dari aplikasi yang memiliki kekurangsesuaian dengan karakteristik dari T2 sebagai *verification tool* – pembahasan detail mengenai hal tersebut telah dijelaskan pada bagian sebelumnya dari laporan ini.

5.1.1. Pandangan Umum

Secara umum, terdapat beberapa catatan terkait penggunaan T2 dalam proses verifikasi terhadap studi kasus. Walaupun perlu digarisbawahi bahwa T2 mampu melakukan dan memberikan hasil pengujian terhadap hampir seluruh spesifikasi yang diujikan, namun terdapat beberapa karakteristik yang perlu diperhatikan terkait proses verifikasi studi kasus.

T2 adalah sebuah *verification tool* yang memungkinkan pengguna untuk melakukan modifikasi terhadap konfigurasi dari T2. Perlu ditekankan bahwa modifikasi terhadap konfigurasi ini dapat memberikan hasil verifikasi dengan tingkat kekakuan (*strictness*) yang berbeda-beda – hal ini mungkin terjadi dengan adanya akses terhadap konfigurasi domain pengujian dan konfigurasi *scope* dari verifikasi. Dalam konteks ini, penggunaan dan modifikasi konfigurasi dari T2 harus dilakukan dengan memperhatikan asumsi-asumsi yang ada dari aplikasi yang akan diujikan.

Hal lain yang perlu diperhatikan adalah bahwa terdapat spesifikasi yang tidak dapat diverifikasi dengan T2 pada penelitian ini, yaitu pada *property* (3). Meskipun demikian, hal ini tidak dapat langsung disimpulkan sebagai indikasi bahwa pengujian akan spesifikasi terkait waktu tidak dapat difasilitasi oleh T2 – pembahasan lebih lanjut mengenai topik ini disajikan pada bagian selanjutnya dari laporan ini, yaitu pada bagian Analisis Hasil Eksperimen.

Di sisi lain, aplikasi E-Voting yang menjadi studi kasus dari penelitian ini adalah aplikasi berbasis *Graphical User Interface* (GUI). Proses verifikasi yang dilakukan tidak meliputi verifikasi terhadap GUI dari aplikasi sebagaimana ditetapkan dalam ruang lingkup penelitian, namun terdapat beberapa aspek GUI yang perlu diperhatikan terkait verifikasi dalam penelitian ini – detail pada bagian berikutnya dari laporan ini.

5.1.2. Analisis Hasil Eksperimen

Berdasarkan hasil eksperimen yang telah dilakukan, ditemukan bahwa terdapat beberapa kendala dalam proses verifikasi. Termasuk dalam kendala yang disebutkan

ditemukan dalam proses verifikasi terhadap salah satu spesifikasi – hal ini terkait secara langsung dengan hasil verifikasi yang dilakukan dengan T2.

Secara umum, kendala yang ditemukan tidak memiliki pengaruh yang signifikan terhadap hasil dari proses verifikasi. Meskipun demikian, beberapa hal terkait kendala yang ditemukan tetap perlu diperhatikan sehubungan dengan kapabilitas dari T2 yang tidak dirancang secara khusus untuk mengeksplorasi area yang bersangkutan.

Verifikasi terhadap *Property* (3)

Sebagaimana telah disebutkan sebelumnya, terdapat kendala dalam proses verifikasi terhadap *property* (3) yang terkait waktu setelah aplikasi berjalan. *Property* ini dinyatakan sebagai berikut, dalam formulasi UNITY.

```
forall X. X ~= NOTIME : t.tick = X |--> t.tick > X
```

Telah dijelaskan sebelumnya bahwa *property* (3) memiliki karakteristik terkait waktu yang bersifat eventual. Meninjau kembali definisinya, spesifikasi tersebut menyatakan bahwa apabila diberikan *X* sebagai nilai waktu (`t.tick`) pada suatu saat, maka nilai `t.tick` akan lebih besar daripada *X* pada suatu saat setelahnya.

Meskipun demikian, Java sebagai bahasa pemrograman tidak mendukung implementasi terhadap *property* ini sesuai definisi aslinya. Sebagaimana telah disajikan dalam bagian sebelumnya, implementasi spesifikasi ini kemudian dilakukan dengan asumsi yang lebih ketat bahwa nilai waktu `t.tick` akan selalu bertambah pada *sembarang* waktu setelah *X*.

Pendekatan yang lebih longgar untuk implementasi dari spesifikasi ini telah didiskusikan pada bab IV, dan pendekatan ini memberikan hasil bahwa aplikasi dinyatakan valid dalam proses verifikasi. Meskipun demikian, implementasi dengan spesifikasi yang lebih longgar tersebut memberikan predikat yang lebih lemah dibandingkan definisi spesifikasi awal, dan dengan demikian tidak digunakan dalam eksperimen.

Dengan definisi yang lebih ketat tersebut, secara informal implementasi dari *property* (3) memiliki makna sebagai berikut: pada suatu waktu *X*, dan aplikasi tidak sedang berada pada masa NOTIME, maka nilai waktu pada suatu masa setelahnya akan memiliki nilai lebih besar daripada *X*. Hal ini dapat juga dipandang bahwa waktu dalam aplikasi selalu berjalan maju, dan tidak dapat dimundurkan kembali. Pada pekerjaan sebelumnya hal ini diimplementasikan dengan menggunakan *library* `java.util.Calendar` yang disediakan Java, dan teknik yang sama digunakan dalam penelitian ini untuk verifikasi terhadap *property* ini.

Pada tahap ini, ditemukan kendala bahwa hasil verifikasi yang dilakukan oleh T2 memberikan hasil yang tidak konsisten dengan teknik ini; terjadi *violation* terhadap spesifikasi ini dalam sesi pengujian dengan T2. Hal ini merupakan kontradiksi bahwa seharusnya nilai dari *instance* `java.util.Calendar` akan selalu bertambah untuk sembarang waktu yang diambil setelahnya.

Penyelidikan lebih lanjut memberikan penjelasan bahwa anomali ini terjadi karena keterbatasan *library* `java.util.Calendar` di mana unit waktu terkecil adalah satu *millisecond* atau 1/1000 detik. Hal ini mengakibatkan hasil verifikasi menjadi tidak konsisten, karena dalam satu sesi verifikasi dengan T2 satu langkah pengujian terhadap spesifikasi terjadi dalam rentang waktu di bawah satu *millisecond*.

Perlu diperhatikan bahwa dengan demikian verifikasi untuk spesifikasi ini menjadi hal yang tidak dapat dilakukan dengan teknik tersebut. Perlu juga digarisbawahi bahwa hal ini tidak terkait dengan kapabilitas T2 sebagai *verification tool*, namun lebih karena keterbatasan dari *library* yang dipergunakan untuk keperluan verifikasi ini.

Terminasi *Thread* Pasca Proses Verifikasi

Aplikasi E-Voting adalah sebuah aplikasi yang terdiri atas komponen-komponen JavaBeans, dengan *nature* sebagai sebuah aplikasi berbasis GUI. Hal ini memiliki implikasi bahwa masing-masing komponen saling berkomunikasi dengan utilisasi *event object* dan *event listener*, dan untuk keperluan ini diperlukan keberadaan *thread*

yang memiliki fungsi untuk menunggu dan mengolah *event* yang mungkin akan diluncurkan oleh aplikasi.

Hal ini berarti bahwa dalam aplikasi E-Voting, terdapat *thread* yang menunggu terjadinya *event* selain *thread* yang melakukan proses logika dari aplikasi. Keadaan ini pada akhirnya mengakibatkan aplikasi tidak akan selesai (*terminate*) setelah proses verifikasi dengan T2 selesai. Pada akhirnya proses verifikasi akan selesai, namun aplikasi tidak akan mengalami terminasi.

Pada aplikasi yang dibangun dengan Java secara *plain* (tanpa mengutilisasi aspek GUI dan *event*), masing-masing *class* yang diemulasi oleh T2 memiliki *constructor* yang akan melakukan terminasi setelah seluruh *statement* dieksekusi. Hal ini mengakibatkan proses terminasi tidak menjadi masalah pada aplikasi-aplikasi tersebut, namun hal ini tidak terjadi pada aplikasi E-Voting – di sini, proses dianggap tidak berhenti karena keberadaan *thread* yang telah dijelaskan sebelumnya.

Topik terkait *thread* telah diantisipasi dalam **[2]** di mana dinyatakan bahwa T2 memiliki keterbatasan terkait dengan program yang berjalan dalam lingkungan *multithreading*. Penggunaan *thread* oleh aplikasi tidak menimbulkan masalah yang signifikan terkait proses dan hasil verifikasi dalam penelitian ini, namun dalam kasus khusus seperti yang ditemukan dalam studi kasus, proses terminasi harus dilakukan secara manual oleh pengguna. Meskipun demikian, perlu diperhatikan bahwa secara umum utilisasi *thread* dalam konteks ini tidak terkait proses verifikasi yang dilakukan oleh T2.

Di sisi lain, proses terminasi thread secara manual mungkin akan menjadi faktor yang mengurangi kenyamanan pengguna, khususnya dalam melakukan *batch testing*. Termasuk dalam konteks ini adalah dalam pengujian terhadap aplikasi dengan kebutuhan khusus seperti halnya E-Voting yang menjadi studi kasus pada penelitian ini.

Graphical User Interface dalam Verifikasi

Telah ditetapkan bahwa verifikasi terhadap GUI merupakan hal yang berada di luar ruang lingkup dari penelitian ini. Meskipun demikian, terdapat beberapa aspek yang perlu diperhatikan terkait GUI dalam eksperimen yang telah dilakukan.

Dalam aplikasi E-Voting, komponen-komponen yang ada dibangun dengan fungsi untuk GUI yang didefinisikan dalam *class* yang bersesuaian. Sebagai contoh, definisi dari *textbox* dan *button* untuk keperluan pemungutan suara didefinisikan dalam komponen `Voting`. Komponen-komponen lain didefinisikan sebagai *subclass* dari JPanel (`javax.swing.JPanel`), dan dengan demikian masing-masing komponen memiliki aspek GUI tersendiri dalam pengembangannya.

Untuk proses verifikasi dengan T2, aspek hal yang dilakukan oleh *mock class* terbatas pada instansiasi komponen-komponen yang dibutuhkan, berikut *wrapper* untuk masing-masing *method* yang dibutuhkan dalam proses pengujian. Aspek GUI sama sekali diabaikan dalam *mock class*, namun dengan keadaan bahwa masing-masing komponen dibiarkan utuh dengan modifikasi minimal terkait proses verifikasi dengan T2 – penjelasan terkait topik ini telah disajikan pada Bab IV dari laporan ini.

Hal yang dapat diperhatikan dari sudut pandang ini adalah bahwa perancangan aplikasi E-Voting yang menjadi studi kasus masih belum mengimplementasikan pemisahan antara logika aplikasi (terkait proses kalkulasi yang dilakukan) dengan aspek tampilan (terkait aspek GUI pada aplikasi). Pada aplikasi E-Voting, proses logika dari aplikasi di-*embed* pada *action listener* yang seharusnya berada pada tataran tampilan aplikasi. Hal ini menimbulkan kebutuhan terhadap modifikasi aplikasi sebelum dapat diverifikasi dengan T2, mengingat T2 tidak dirancang untuk dapat melakukan verifikasi terhadap aspek GUI dari aplikasi.

Mempertimbangkan bahwa proses verifikasi dalam penelitian ini difokuskan hanya kepada aspek logika dari aplikasi, pemisahan antara aspek logika dan aspek tampilan adalah hal yang dapat dilakukan untuk meningkatkan efisiensi proses verifikasi. Dalam konteks ini, pemisahan antara kedua aspek tersebut akan menghasilkan proses verifikasi yang lebih terfokus, dengan pembatasan lingkup verifikasi hanya kepada bagian-bagian yang relevan dari aplikasi.

Perlu diperhatikan bahwa aspek GUI dalam sebuah aplikasi berbasis Java meliputi domain yang kompleks; termasuk di antaranya adalah *thread* dan *event handling*. T2 tidak dirancang untuk melakukan verifikasi dalam area ini, dan masih belum diketahui sejauh mana kapabilitas T2 bisa digunakan untuk mengeksplorasi area tersebut. Eksperimen dalam penelitian ini dilakukan dengan mengabaikan aspek GUI dari aplikasi E-Voting, dan sebagaimana telah ditetapkan dalam ruang lingkup penelitian, proses verifikasi terhadap aspek GUI tidak menjadi perhatian dari penelitian ini.

5.2. Analisis Kapabilitas T2 terhadap Eksperimen Studi Kasus

Pada bagian ini, pembahasan difokuskan kepada analisis terhadap kapabilitas T2 terkait eksperimen yang dilakukan terhadap studi kasus. Pembahasan pada bagian ini meliputi kesesuaian T2 sebagai sebuah *verification tool* untuk keadaan khusus yang dimiliki oleh aplikasi E-Voting sebagai studi kasus.

Di bagian pertama, disajikan pembahasan terkait analisis kapabilitas T2 dalam verifikasi studi kasus berupa *component software*. Pada bagian berikutnya, pembahasan difokuskan kepada analisis terhadap kapabilitas T2 dalam verifikasi studi kasus berupa aplikasi yang berbasis GUI.

5.2.1. Studi Kasus sebagai *Component Software*

Pada dasarnya, T2 adalah sebuah *verification tool* yang dikembangkan untuk sebuah aplikasi yang dikembangkan dengan Java secara *plain*, tanpa mempertimbangkan aspek *component software* dalam implementasi JavaBeans pada studi kasus. Meskipun demikian, implementasi dengan JavaBeans sebagai *component software* memiliki karakteristik yang pada dasarnya serupa dengan implementasi sebuah *class* standar pada Java.

Dari hasil eksperimen yang telah dilakukan, dapat diperhatikan bahwa proses verifikasi dengan T2 dapat dilakukan dengan baik untuk sebuah *component software* yang dibangun dengan JavaBeans. Terdapat beberapa penyesuaian yang harus

dilakukan untuk mengakomodasi aspek-aspek pada studi kasus sebagai sebuah *component software*, namun secara keseluruhan proses verifikasi dengan T2 dapat dilakukan dengan baik terlepas dari konteks studi kasus sebagai *component software*.

T2 melakukan proses verifikasi terhadap sebuah *class* secara independen, terhadap satu buah *class* dalam satu waktu. Dalam konteks ini, sebuah komponen dapat dipandang sebagai sebuah *class*, dan verifikasi terhadap komponen dapat dilakukan secara independen oleh T2. Di sisi lain, untuk sebuah aplikasi yang dibangun dengan komponen-komponen yang saling memiliki ketergantungan, proses verifikasi dengan T2 harus dilakukan dengan sebuah *class* yang merepresentasikan keseluruhan aplikasi dan komponen-komponen yang digunakan.

5.2.2. Aspek GUI pada Aplikasi Studi Kasus

Telah dijelaskan pada bagian sebelumnya dari laporan ini bahwa proses verifikasi dengan T2 dilakukan dengan mengabaikan aspek GUI, walaupun dengan keadaan bahwa *statement-statement* terkait aspek GUI tetap dieksekusi dalam komponen. Dalam eksperimen yang dilakukan, proses verifikasi yang dilakukan hanya memperhatikan elemen-elemen terkait logika aplikasi.

Tidak ditemukan masalah dengan proses verifikasi yang dilakukan oleh T2, dengan catatan bahwa aspek-aspek GUI yang tidak signifikan terhadap logika diabaikan dalam proses verifikasi. Utilisasi *event object* dan *event listener* tidak mempengaruhi jalannya proses verifikasi oleh T2, dan fungsi-fungsi dari aplikasi berjalan dengan baik terlepas dari diabaikannya aspek GUI pada *mock class* yang menjadi media pengujian.

Dari hasil eksperimen yang telah dilakukan, proses verifikasi dengan T2 tampak tidak memiliki masalah untuk aplikasi yang dirancang dengan GUI. Meskipun demikian, terdapat catatan bahwa dalam penelitian ini aspek GUI diabaikan dalam proses verifikasi – hal ini ternyata tidak mengganggu jalannya proses verifikasi, dan dengan demikian penggunaan T2 untuk aplikasi yang dirancang dengan GUI tetap mampu memberikan hasil yang valid.

BAB VI ANALISIS TERHADAP T2 FRAMEWORK

Bagian ini memaparkan mengenai analisis terhadap kapabilitas dari T2 Framework, berdasarkan hasil eksperimen yang telah disajikan dalam pembahasan pada bagian sebelumnya dari laporan ini. Termasuk dalam pembahasan pada bagian ini adalah analisis terhadap penggunaan T2 secara umum dalam fungsinya sebagai sebuah *verification tool*.

Pembahasan pada bagian ini difokuskan kepada karakteristik dari T2 sebagai sebuah *verification tool*; hal ini membedakan dengan topik bahasan pada Bab V, di mana pada bagian tersebut proses analisis difokuskan kepada eksperimen terhadap studi kasus. Pembahasan pada bagian ini menyajikan analisis terhadap kapabilitas dari T2 secara umum, khususnya dari segi fungsional dan pendekatan yang dilakukan oleh T2 dalam melakukan verifikasi.

6.1. Penggunaan T2 Framework

T2 adalah sebuah *verification tool* yang dikembangkan untuk keperluan verifikasi terhadap aplikasi berbasis Java. Secara khusus, T2 dikembangkan untuk memfasilitasi pengujian dengan pendekatan yang lebih formal dan simbolik dalam proses verifikasi. Meskipun demikian, penerapan spesifikasi dalam T2 tidak bisa benar-benar dilakukan dalam notasi yang bersifat formal; masing-masing spesifikasi harus terlebih dahulu diterjemahkan ke dalam Java.

Pendekatan yang ditawarkan oleh T2 meliputi kapabilitas untuk mengeksplorasi teknik dan *library* yang disediakan oleh Java untuk keperluan definisi spesifikasi. Dengan demikian, proses verifikasi dalam T2 memiliki karakteristik yang bersifat formal (dengan catatan bahwa spesifikasi harus diterjemahkan dalam Java), deklaratif (menyatakan keadaan-keadaan yang harus dipenuhi) dan *powerful* (dengan eksploitasi terhadap *library* dari Java).

Secara umum, T2 didesain untuk verifikasi aplikasi yang dikembangkan dengan Java. Sehubungan dengan konteks aplikasi yang diujikan, perlu diperhatikan bahwa mungkin terdapat karakteristik-karakteristik yang harus diakomodasi dalam pengujian. Berikut adalah

analisis penggunaan T2 dalam masing-masing konteks aplikasi, dengan analisis yang didasarkan pada hasil dari eksperimen yang telah dilakukan.

6.1.1. *Standard Class* dalam Java

T2 dirancang untuk keperluan verifikasi aplikasi yang dibangun dengan Java, secara khusus aplikasi yang dibangun sebagai sebuah *standard class* dalam Java. Di sini, sebuah *standard class* didefinisikan sebagai aplikasi yang tidak mengutilisasi aspek-aspek mutakhir dari Java; termasuk dalam klasifikasi ini adalah utilisasi *thread*, *Graphical User Interface*, dan implementasi *component software* dengan JavaBeans.

Untuk aplikasi berupa *standard class* dalam Java, T2 memiliki karakteristik yang *powerful* dengan pendekatan formal, tanpa penyesuaian-penyesuaian yang perlu dilakukan seperti halnya pada konteks aplikasi lain. Beberapa fasilitas seperti *method wrapper* yang disediakan oleh T2 dirancang untuk mendukung keperluan ini tanpa penyesuaian lebih lanjut – hal ini tidak terjadi pada proses verifikasi untuk konteks aplikasi yang lain yang membutuhkan beberapa penyesuaian.

Secara umum, T2 bekerja dengan baik untuk konteks aplikasi pada *standard class* dalam Java. Pada eksperimen awal yang dilakukan dalam penelitian ini, T2 berhasil menjalankan fungsi verifikasi tanpa kendala, dan memberikan hasil sebagaimana telah dijelaskan pada bagian sebelumnya dari laporan ini – secara khusus, pada Bab III yang mendiskusikan mengenai eksperimen awal.

6.1.2. *Component Software*

Pada dasarnya, aplikasi berbasis JavaBeans merupakan bentuk khusus dari aplikasi dengan *standard class* pada Java. Memperhatikan karakteristik yang pada dasarnya serupa, proses verifikasi dengan T2 merupakan hal yang dapat dilakukan untuk keperluan ini.

Dalam konteks *component software*, sebuah *class* mungkin menjadi sebuah komponen, atau sebuah aplikasi yang dibangun dari komponen-komponen lain. Di

sini, T2 dapat memfasilitasi pengujian untuk kedua keadaan tersebut. Sebagai sebuah komponen, proses pengujian dapat dilakukan secara independen seperti halnya pada *standard class*. Hal ini dapat difasilitasi mengingat karakteristik dasar sebuah komponen JavaBeans adalah juga sebagai sebuah *class* pada Java.

Di sisi lain, untuk sebuah aplikasi yang dibangun dari komponen-komponen lain dibutuhkan pendekatan dengan beberapa penyesuaian. Perlu diperhatikan bahwa T2 tidak dapat mengkombinasikan pengecekan terhadap masing-masing komponen sebagai *class* yang berbeda. Pada keadaan ini, diperlukan pendekatan khusus untuk mengkombinasikan *instance* dari masing-masing komponen ke dalam suatu *class* tersendiri.

Dalam verifikasi terhadap *component software*, spesifikasi dapat didefinisikan baik dari internal komponen maupun dari *class* utama. Hal ini memberikan pengecekan yang kuat dalam proses verifikasi terhadap *component software*. Dalam kasus ini, misalkan komponen *K* diinstansiasi oleh *class C* dalam *main class* dengan implementasi JavaBeans, dan kemudian T2 melakukan pengujian terhadap *C*. Dalam kasus ini, T2 melakukan pengecekan tidak hanya terhadap spesifikasi pada *C*, namun juga pada *K* dan elemen-elemen di bawahnya. Dikombinasikan dengan penerapan spesifikasi pada *K* dan *C*, hal ini menghasilkan pengecekan yang kuat dengan verifikasi dengan T2.

Di sini perlu ditekankan bahwa untuk sebuah aplikasi dengan komponen-komponen yang saling berkomunikasi, dibutuhkan *method-method* yang berperan sebagai *wrapper* untuk masing-masing *method* yang menjadi anggota dari komponen. Pendekatan ini membutuhkan pengetahuan yang minimal terhadap komponen, dalam hal ini terkait nama serta parameter dan *return type* dari *method* yang akan diselidiki.

Secara keseluruhan, verifikasi dengan T2 merupakan hal yang mungkin dilakukan untuk konteks aplikasi *component software* berbasis Java. Dengan tetap mempertimbangkan perbedaan karakteristik yang mungkin timbul antara *standard class* dan *component software* dengan JavaBeans, T2 tetap dapat melakukan verifikasi dengan baik – dengan catatan mengenai penyesuaian-penyesuaian yang telah didiskusikan sebelumnya.

6.1.3. Aplikasi *Multithreading*

Telah dinyatakan dalam [2] bahwa T2 tidak dapat melakukan verifikasi terhadap *thread* yang berjalan secara *concurrent*. Meskipun demikian, hal ini tidak berarti bahwa T2 sama sekali tidak dapat digunakan pada aplikasi yang melakukan utilisasi terhadap *thread*.

Berdasarkan hasil eksperimen terhadap studi kasus, diperoleh hasil bahwa T2 dapat menjalankan fungsi verifikasi dalam aplikasi dengan utilisasi *thread*. Meskipun demikian, perlu diperhatikan bahwa pada studi kasus penerapan *multithreading* dilakukan untuk keperluan *event handling* pada aplikasi. Di sini, proses yang melibatkan logika dari aplikasi berjalan pada *statement-statement* yang dieksekusi secara sekuensial pada satu *thread* sehingga verifikasi dengan T2 tidak mengalami hambatan dalam melakukan pekerjaannya.

Hal yang perlu digarisbawahi adalah bahwa T2 tidak dapat melakukan verifikasi terhadap integritas dari jalannya aplikasi yang diakses oleh lebih dari satu *thread* yang berjalan secara *concurrent* – hal ini tidak terjadi pada studi kasus. Sebagai catatan, perlu diperhatikan bahwa utilisasi *thread* pada studi kasus mengakibatkan T2 tidak dapat melakukan terminasi aplikasi pasca proses verifikasi. Hal ini tidak menimbulkan kendala dari segi verifikasi, namun dengan demikian terminasi pasca verifikasi harus dilakukan secara manual oleh pengguna.

Dari hasil eksperimen yang telah dilakukan, dapat diperhatikan bahwa ketidakcocokan T2 dalam melakukan verifikasi terhadap aplikasi dengan *concurrent thread* tidak mengakibatkan T2 otomatis tidak cocok untuk verifikasi pada konteks aplikasi tersebut. Meskipun demikian, perlu diperhatikan bahwa integritas dari proses verifikasi oleh T2 dapat dijamin hanya untuk eksekusi *statement* pada satu *thread* tertentu – keadaan inilah yang terjadi dan ditemukan pada studi kasus.

6.1.4. *Graphical User Interface*

Pada dasarnya, T2 tidak dirancang untuk melakukan verifikasi terhadap aspek-aspek terkait GUI dari program yang diujikan. Termasuk dalam aspek-aspek terkait GUI adalah proses *event handling* terhadap aksi yang dilakukan oleh pengguna terhadap *interface* dari aplikasi (misal: klik suatu *button*), dan hal ini berada di luar ruang lingkup dari penelitian.

Meskipun dengan keadaan bahwa T2 tidak dirancang untuk melakukan verifikasi terhadap aspek GUI, hal ini tidak berarti bahwa T2 tidak cocok untuk melakukan verifikasi pada konteks aplikasi berbasis GUI; memperhatikan eksperimen terhadap studi kasus, proses verifikasi dapat dilakukan dan memberikan hasil dengan beberapa penyesuaian. Dalam penelitian ini penyesuaian dilakukan dengan mengabaikan aspek GUI pada aplikasi, dan dengan demikian proses verifikasi difokuskan kepada logika dari aplikasi yang diujikan. Pada tahap ini, perlu ditekankan bahwa *statement-statement* terkait aspek GUI dari aplikasi akan tetap dieksekusi – namun tidak ikut diverifikasi oleh T2.

Hal ini dapat dijelaskan sehubungan dengan karakteristik dari T2 dalam melakukan verifikasi. T2 melakukan pengujian dengan membandingkan antara spesifikasi dari aplikasi dengan keadaan dari aplikasi pada saat tertentu. Dengan demikian, aspek-aspek dari GUI yang tidak dinyatakan dalam spesifikasi tidak akan diperiksa oleh T2. Hal ini tidak berarti bahwa T2 sama sekali meninggalkan eksekusi dari *statement* terkait aspek GUI – T2 hanya tidak memperhatikan bagian tersebut untuk diverifikasi.

Dengan demikian, T2 dapat digunakan untuk melakukan verifikasi terhadap aplikasi dengan utilisasi terhadap aspek GUI. Perlu ditekankan bahwa pada tahap ini proses verifikasi yang dilakukan oleh T2 harus didefinisikan untuk dilakukan hanya terhadap proses logika dari aplikasi. Proses verifikasi terhadap elemen-elemen terkait aspek GUI merupakan hal yang memungkinkan untuk dilakukan, namun perlu ditekankan bahwa saat ini kapabilitas T2 untuk melakukan eksplorasi di area tersebut masih belum diketahui.

6.2. Penerapan dengan Metode Formal

T2 dirancang untuk melakukan verifikasi dengan pendekatan formal, termasuk di antaranya adalah dukungan untuk spesifikasi *Hoare triple* dan *class invariant*. Pada bagian ini, pembahasan akan difokuskan kepada analisis dari kapabilitas T2 dalam melakukan penerapan metode formal dalam proses verifikasi.

Pada pembahasan terkait penerapan metode formal, terdapat beberapa pertimbangan yang perlu dianalisis terkait pendekatan formal dalam verifikasi oleh T2. Terkait dalam pembahasan ini adalah proses implementasi dari spesifikasi formal untuk keperluan verifikasi, dan pandangan-pandangan lain dalam konteks definisi spesifikasi terkait proses verifikasi dengan T2.

6.2.1. Spesifikasi Formal dalam T2

Telah dinyatakan sebelumnya bahwa T2 memiliki dukungan terhadap definisi spesifikasi dalam pendekatan formal, khususnya dengan definisi *class invariant* dan *Hoare triple* untuk *method specifications*. Untuk keperluan ini, diperlukan analisis terhadap implementasi dari pendekatan yang sejatinya dilakukan secara formal tersebut.

Dalam pengujian dengan T2, definisi dari spesifikasi harus diterjemahkan ke dalam Java untuk dapat diverifikasi lebih lanjut. Pada keadaan ini, perlu dijamin bahwa spesifikasi yang ditetapkan harus diterjemahkan secara tepat ke dalam aplikasi, sebelum kemudian di-*compile* dan diverifikasi oleh T2. Untuk keperluan ini, diperlukan suatu penjaminan bahwa spesifikasi yang ditetapkan harus bisa diterjemahkan secara tepat ke dalam definisi *syntax* yang dipahami oleh Java.

Secara umum, proses penerjemahan dari spesifikasi dalam Java tidak memiliki masalah untuk spesifikasi terkait nilai kebenaran dari suatu predikat. Java memiliki perangkat pendukung operasi logika dengan *boolean*, termasuk dengan penggunaan operator-operator logika untuk keperluan pengecekan terhadap nilai kebenaran dari suatu predikat. Dari sisi ini, proses penerjemahan terhadap operasi logika sederhana

(misal: konjungsi dan disjungsi dapat diwakilkan dengan & dan |) merupakan hal yang dapat dilakukan dengan dukungan *native* dari Java.

Meskipun demikian, perlu diperhatikan bahwa tidak semua operator logika dapat diterjemahkan secara persis dengan *keyword* tertentu pada Java. Sebagai contoh, predikat dengan operator seperti $\forall$ dan $\exists$ akan harus diterjemahkan dengan melakukan enumerasi terhadap variabel-variabel terkait. Dalam implementasi *in-code specification*, penerjemahan untuk spesifikasi yang lebih kompleks mungkin akan harus dilakukan dalam rangkaian *statement* yang lebih panjang. Dengan demikian, proses penerjemahan dari spesifikasi yang bersifat formal ke dalam spesifikasi yang sesuai dengan definisi *syntax* pada Java merupakan tahap yang perlu diperhatikan secara seksama untuk menjamin kesesuaian dari spesifikasi yang akan diujikan terhadap aplikasi.

6.2.2. *Temporal Properties*

Salah satu spesifikasi formal yang didukung oleh T2 adalah spesifikasi terkait *temporal properties*, atau predikat yang terkait waktu dalam jalannya aplikasi. Hal ini, selain *class invariant* dan *Hoare triple* dalam *method specifications*, merupakan pendekatan formal yang didukung dalam proses verifikasi oleh T2.

Perlu ditekankan bahwa dalam verifikasi oleh T2 *temporal properties* sama sekali tidak terkait dengan *thread* yang berjalan secara *concurrent* – hal tersebut berada di luar kemampuan T2 sebagai *verification tool*. Di sini, *temporal properties* dapat didefinisikan secara sederhana sebagai predikat yang berubah terhadap waktu.

Implementasi spesifikasi berupa *temporal properties* untuk verifikasi dengan T2 merupakan hal yang membutuhkan perhatian khusus; dalam kasus ini, spesifikasi tidak dapat dinyatakan secara langsung dengan definisi *syntax* yang dipahami oleh Java. Spesifikasi untuk *temporal properties* dinyatakan dalam pendekatan berupa emulasi keadaan aplikasi sebagai beberapa *state* yang menggambarkan periode waktu dalam aplikasi. Pendekatan ini dilakukan dengan pembuatan *automaton* berupa

serangkaian *statement* yang dirancang khusus untuk verifikasi, dan dengan demikian membutuhkan pertimbangan tersendiri dalam implementasinya.

Hal ini dapat diperhatikan pada implementasi dari spesifikasi terkait waktu pemungutan suara pada studi kasus. Sebagai contoh, berikut adalah definisi dari *property* (1) pada spesifikasi aplikasi E-Voting. Spesifikasi tersebut menyatakan bahwa jumlah suara yang masuk setelah penutupan sesi pemungutan suara akan selalu lebih besar atau sama dengan jumlah *vote* yang masuk lewat *instance* dari komponen `Voting`.

```
wstable t.tick > LIMIT /\ V supseteq v.votes
```

Untuk implementasi *property* ini, diperlukan pendekatan berupa definisi *state* untuk merepresentasikan keadaan ketika sesi pemungutan suara dibuka dan ketika sesi pemungutan suara ditutup. Definisi *state* ini dilakukan khusus untuk keperluan verifikasi dengan T2, dan sama sekali tidak terkait dengan definisi proses logika dari aplikasi.

Berdasarkan analisis terhadap hasil eksperimen dengan studi kasus yang telah didiskusikan sebelumnya pada Bab V, diperoleh bahwa proses implementasi spesifikasi terkait *temporal properties* merupakan hal yang membutuhkan perhatian secara seksama. Pada tahap ini, implementasi spesifikasi terkait *temporal properties* harus benar-benar memperhatikan kesesuaian antara spesifikasi formal dan penerjemahannya dalam *in-code specification*. Di sisi lain, perlu ditekankan bahwa penerjemahan *temporal properties* memiliki banyak variabel yang perlu diperhitungkan dalam implementasinya: Proses ini meliputi definisi *state* dan definisi predikat-predikat yang dibutuhkan pada masing-masing *state*, berikut pertimbangan terhadap proses transisi antar *state* yang juga harus didefinisikan pada tahap implementasi dari spesifikasi.

6.3. Penerapan dalam Rekayasa Perangkat Lunak

Di luar konteks verifikasi dengan pendekatan formal, T2 adalah sebuah *testing tool* yang dibangun untuk membantu proses pengujian pada praktek rekayasa perangkat lunak. Pada sudut pandang yang lebih pragmatis, T2 adalah bagian dari suatu proses rekayasa perangkat lunak, khususnya pada tahap pengujian atau *testing* dari suatu aplikasi. Dari perspektif ini, terdapat beberapa hal yang perlu diperhatikan terkait karakteristik dari T2 dalam konteks rekayasa perangkat lunak.

Secara umum, T2 adalah sebuah *unit testing tool*, dengan unit yang diuji berada pada lingkup sebuah *class*. Hal ini sesuai dengan proses verifikasi yang dilakukan bahwa T2 melakukan pengujian terhadap sebuah *class* secara independen, masing-masing dengan spesifikasi yang dituliskan dalam *class* yang bersesuaian. Meskipun demikian, kapabilitas T2 dalam proses pengujian tidak terbatas pada *standard class* yang dibangun dengan Java, namun juga meliputi komponen-komponen yang dikembangkan dalam *component-based software engineering*.

Di sisi lain, perlu diperhatikan bahwa definisi spesifikasi dan proses verifikasi oleh T2 memiliki *maintenance cost* yang rendah; sehubungan dengan sifatnya yang menggunakan *in-code specification*, proses pemeliharaan dari spesifikasi dapat dilaksanakan secara efisien. Hal ini dapat dicapai dengan keadan bahwa spesifikasi tersebut harus di-*compile* bersama-sama dengan aplikasi sebagai prasyarat dari proses verifikasi. Secara umum, hal ini juga berarti bahwa dalam kasus terjadi perubahan desain aplikasi, sinkronisasi antara spesifikasi dan implementasi dapat dilakukan dengan lebih efisien mengingat definisi dari keduanya dituliskan pada *sourcecode* yang sama.