PERANCANGAN ATURAN TRANSFORMASI UML – SYSTEMC DALAM PERANCANGAN *EMBEDDED SYSTEM*

Maman Abdurohman[1], Kuspriyanto[2], Sarwono Sutikno[2], dan Arif Sasongko[2]

[1]Fak. Informatika, IT Telkom, Jl. Telekomunikasi Terusan Buah Batu, Bandung, 40257, Indonesia
[2]STEI, Institut Teknologi Bandung, Jl. Ganesha 10, Bandung, 40132, Indonesia

E-mail: m_abdurohman@yahoo.com

Abstrak

Pemodelan adalah salah satu proses awal dalam pengembangan suatu aplikasi atau produk. Tahap ini dilakukan untuk meminimalkan kesalahan pada produk akhir. Salah satu metode pemodelan berorientasi objek yang banyak digunakan adalah pemodelan UML (*Unified Modeling Language*). Dalam UML suatu sistem dipandang sebagai kumpulan objek yang memiliki atribut dan *method*. SystemC adalah bahasa perancangan perangkat keras yang berbasis C++. SystemC merupakan sebuah *library* yang mendefinisikan tipe-tipe komponen perangkat keras. Dalam pemodelan bersama perangkat keras dan perangkat lunak, UML dan SystemC memiliki kemampuan yang sama. Pada *paper* ini dilakukan analisis proses transformasi dari pemodelan berorientasi objek dengan UML dan implementasi dengan menggunakan SystemC. Hasil penelitian menunjukan bahwa proses transformasi UML-SystemC dapat dilakukan karena keduanya memiliki *nature* yang sama sebagai lingkungan yang dapat merancang bersama *hardware* dan *software*. Perangkat yang digunakan untuk penelitian ini adalah Rational Rose dan SystemC.

Kata Kunci: *C++, rational rose, systemC, UML*

Abstract

Modeling is one of the first process in the development of an application or product. This phase is done to minimize errors in the final product. One method in object-oriented modeling that is widely used is UML (Unified Modeling Language). In UML a system is seen as a collection of objects that have attributes and methods. SystemC is a hardware design language based on C++. SystemC is a library that defines the types of hardware components. In a joint modeling of hardware and software, UML and SystemC have similar capabilities. In this paper, researchers analyzed the transformation of object-oriented modeling with UML and the implementation by using SystemC. The results shows that the transformation process of UML-SystemC can be done because both have the same nature as the environment that can design both hardware and software. The device used for this study is the Rational Rose and SystemC.

Keywords: *C++, rational rose, systemC, UML,*

1. Pendahuluan

Proses perancangan *embedded system* dimulai dengan spesifikasi produk, pemisahan HW/SW (*Hardware/Software*), perancangan HW/SW secara rinci, integrasi dan pengujian, seperti pada gambar 1. Biasanya ada dua metrik yang digunakan untuk menentukan kesesuaian hasil rancangan yaitu biaya dan kecepatan. Kedua metrik tersebut banyak digunakan dalam proses perancangan. Metrik lainnya seperti ukuran, konsumsi listrik, fleksibilitas, skalabilitas dan lain-lain.

Spesifikasi produk menjelaskan batasan-batasan perancangan sesuai dengan keinginan perancang. Proses pemisahan HW/SW yaitu melakukan pemisahan antara bagian sistem yang akan diimplementasikan dalam bentuk HW dan SW. Perancangan rinci HW dan SW dilakukan secara paralel. Kemudian proses integrasi dan pengujian.

Perancangan level *register* dilakukan dengan menggunakan bahasa pemrograman perangkat keras, yang umum digunakan adalah Verilog, VHDL dan ESTEREL. Proses verifikasi dan pengujian dilakukan untuk memastikan kesesuaian hasil rancangan dengan spesifikasi awal [1].

Semakin terasanya kegunaan *embedded system* pada berbagai perangkat menyebabkan peningkatan permintaan produk-produk *embedded system*. Di samping variasi sistem yang diminta

pasar juga waktu pasar yang semakin sempit (*time-to-market*). Semakin cepat perubahan teknologi maka teknologi semakin cepat usang.

Permintaan pasar yang bervariasi dan dalam tempo yang relatif singkat, bagi produsen merupakan tekanan yang harus direspons. Kondisi ini diistilahkan sebagai *time-to-market pressure*. Proses perancangan dan implementasi perangkat perlu untuk ditingkatkan dengan membuat metode perancangan yang lebih cepat sehingga dari mulai ide dasar sampai implementasi *prototype* dilakukan sebelum permintaan pasar meningkat. Dalam pengembangan perangkat dibuat suatu perhitungan peluang pendapatan sehingga menghasilkan batasan biaya teknis perangkat. Biaya teknis perangkat ini akan menjadi batasan dalam pemodelan perancangan.

Abstraksi model menunjukkan tingkat ketelitian suatu model. Semakin tinggi tingkat abstraksi suatu model maka tingkat ketelitian semakin rendah. Pada perancangan level RTL satuan terkecil rancangan yaitu: komponen komputasi, sinyal data, sinyal kendali, dan sinyal *clock*.

Sinyal data, kendali dan *clock* berperan dalam komunikasi antar komponen komputasi. Pada tingkat kompleksitas tertentu perancangan dengan rincian sinyal yang akurat cukup memadai. Kelebihan model ini adalah tingkat akurasi *cycle* dan *clock* yang tinggi sehingga hasilnya lebih presisi. Pendefinisian yang rinci komponen komputasi beserta sinyal-sinyal yang terdapat di dalamnya dan *interface* yang menghubungkannya dengan dunia luar dapat dipandang suatu kelebihan.

Pada sisi lain, hal ini dapat dipandang sebagai hambatan untuk kecepatan simulasi. Kompleksitas rancangan suatu *embedded system*, memerlukan waktu simulasi dan pengujian yang lebih lama. Tingkat akurasi yang tinggi terhadap pendefinisan sampai tingkat sinyal adalah suatu hambatan karena memakan waktu. Kita bisa analogikan ini ketika kita melihat pada sudut pandang abstraksi yang lebih rendah di bawah RTL yaitu abstraksi gerbang. Contoh, rangkaian penjumlah pada level gerbang dituliskan secara rinci. Pada perancangan level RTL penjumlah ini lebih sederhana, hanya memerlukan satu baris instruksi, z<=x+y;, tanpa mengurangi fungsionalitasnya. Demikian juga dengan pomodelan level RTL yang dianggap masih rendah perlu untuk ditingkatkan.

Pemodelan yang dimaksud adalah sesuatu di atas RTL. Dengan tingkat abstraksi yang lebih tinggi ini bagian-bagian perancangan yang kurang perlu akan diabaikan. Sinyal, *interface*, *clock* adalah bagian-bagian perancangan yang tidak akan terlalu diperhatikan pada perancangan level tinggi.

2. Metodologi

Abstraksi perancangan adalah bagaimana cara memandang objek rancangan. Dalam perancangan *embedded system* dimulai dari perancang yang paling rendah yaitu level *transistor*, gerbang – *flip-flop*, *register*, dan elektronik.

Perancangan level *transistor* dan gerbang sudah lama ditinggalkan karena proses verifikasinya yang sangat lambat dan tidak memadai lagi untuk memenuhi tuntutan kecepatan perancangan. Dengan kemampatan IC lebih dari lima miliar per *chip*, perancangan berbasis gerbang – *flip-flop* tidak lagi mencukupi terutama dalam hal kecepatan proses.

Peningkatan level abstraksi perancangan telah dilakukan yaitu dengan meningkatkan level abstraksi perancangan dari level gerbang ke *Register Transfer* Level (RTL). Langkah ini sebenarnya termasuk langkah yang signifikan dan ada yang menyebutnya sebagai revolusi teknologi [2] bagi peningkatan produktivitas perancang. Perancang tidak lagi disibukan untuk membahas pada level gerbang. Perancangan level gerbang sudah dianggap mafhum dan tidak perlu lagi dibahas. Perancangan dimulai dengan mendefinisikan komponen-komponon di atas gerbang yaitu *register*.

Terdapat dua bahasa perancangan level RTL yang dikenal luas dan telah distandarkan oleh IEEE yaitu Verilog dan VHDL. Keduanya secara *de facto*, dianggap sebagai bahasa perancangan perangkat keras, *Hardware Description Language* (HDL). Pada awalnya, kedua bahasa ini adalah alat yang digunakan untuk menyimpan notasi hasil rancangan bukan bahasa pemrograman umumnya.

Level abstraksi RTL dianggap belum menjawab tuntutan perkembangan *embedded system*. Beberapa kondisi yang layak dipertimbangkan di sini antara lain, pertama semakin mampatnya teknologi IC menuntut kemampuan untuk meletakkan miliaran *transistor* pada suatu *chip* yang kecil. Kedua, aplikasi semakin kompleks dan terus semakin kompleks serta terintegrasi dengan komunikasi, kendali, sistem *mobile* dan menyebar ke mana-mana. Ketiga, fungsionalitas perangkat keras dan perangkat lunak semakin fleksibel untuk dipertukarkan. Semua fungsi bisa diperangkatkeraskan dan diperangkatlunakkan. Sistem perancangan dituntut untuk dapat

melakukan pemisahan yang tepat sesuai dengan kebutuhan perangkat.

Pada dua dekade terakhir (awal tahun 1990 sampai sekarang) pemikiran ke arah peningkatan level abstraksi perancangan mulai banyak dibicarakan. Mestinya ada sesuatu level abstraksi di atas RTL. Ide awal pengistilahan level perancangan di atas RTL adalah level elektronik, dengan istilah *Electronic System Level* (ESL).

Definisi standar ESL masih belum standar. Secara umum pengertiannya adalah suatu sistem di atas RTL (*Register Transfer Level*), bisa berupa *hardware* maupun *software*. *Unified Modeling Language* (UML) adalah bahasa untuk spesifikasi, visualisasi, pembangunan dan dokumentasi sistem perangkat lunak. Pada perancangan UML, sistem didefinisikan sebagai sekumpulan objek yang memiliki atribut dan metode. Atribut adalah variabel-variabel yang melekat pada objek. Sedangkan metode adalah fungsi-fungsi yang dapat dilakukan oleh objek.

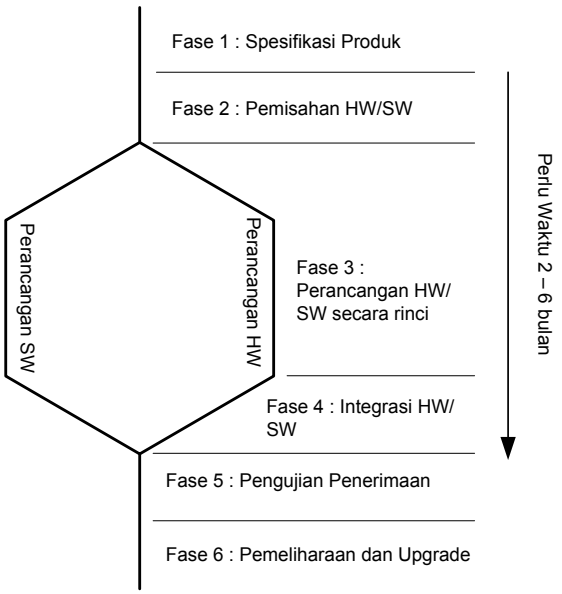
Kelas objek tidak dapat berdiri sendiri dalam penggunaannya, dilakukan perwujudan (*instansiasi*) dari objek tersebut. Beberapa tahapan dalam pemodelan UML adalah pendefinisan *use case* untuk analisis kebutuhan. Dilanjutkan dengan pendefinisian model berupa diagram yang menunjukan konsep atau objek.

Diagram interaksi adalah notasi yang menunjukan kolaborasi antar objek. Tahap selanjutnya adalah pendefinisian diagram kelas. Dalam UML terdapat 13 tipe diagram, yaitu: *sequence diagram, state machine diagram, activity diagram, package diagrams, use case diagrams, class diagrams, timing diagram, communication diagram (collaboration diagram), component diagram, object diagrams, interaction overview diagrams,* dan *composite structure diagrams.*

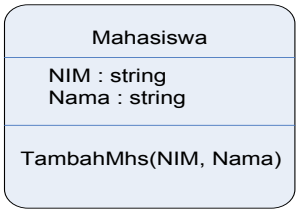
UML adalah metode pemodelan berorientasi objek. Sebelum UML telah banyak dibuat pemodelan berorientasi objek seperti : Simula-67, Objective C, C++, Eiffel, dan CLOS. Sesuai dengan namanya, *Unified*, pada dasarnya UML merupakan pemodelan yang generik. Sifatnya yang general memudahkan UML untuk digunakan pada berbagai tipe sistem yang dimodelkan.

Pada level atas, cara pandang dalam UML dapat dibagi menjadi tiga area: *structural clasification*, *dynamic behavior,* dan *model management*. Klasifikasi struktural menjelaskan suatu objek dalam sistem dan hubungan antar objek tersebut. Pembahasan di dalamnya menyangkut *class, use case*, komponen dan *node. Dynamic behavior* menjelaskan prilaku sistem dari waktu ke waktu. Termasuk dalam *dynamic behavior view* adalah *state machine view*, *activity view,* dan *interaction view.* Model *management* menjelaskan pengorganisasian model secara hierarkis.

Konsep *view* statis sangat penting dalam UML. *View* ini disebut statis karena tidak menjelaskan perilaku sistem yang dengan ukuran waktu. Unsur-unsur pokok dalam *view* statis adalah kelas dan relasi antar kelas, seperti: asosiasi, generalisasi, dan berbagai jenis ketergantungan antar kelas. *Class* adalah penggambaran konsep dari domain aplikasi atau solusi aplikasi. *View* statis digambarkan dalam *diagram class*, yang fokusnya menjelaskan tentang kelas. Gambar 2 menunjukan contoh *diagram class.*



Gambar 1. Alur proses perancangan *embedded system* secara umum.



Gambar 2. *Diagram class* mahasiswa.

Pada *diagram class* gambar 2, nama *class* : mahasiswa, atributnya NIM dan nama, tipe atributnya string dan operasi atau metodenya TambahMhs(NIM, Nama). Pada awalnya UML dirancang untuk pemodelan perangkat lunak. Dengan kemampuan yang dimilikinya dalam memodelkan, UML dapat digunakan dalam pemodelan perangkat keras. Hal ini dikarenakan UML dapat mengakomodasi pemodelan berorientasi objek yang memungkinkan untuk digunakan pada pemodelan perangkat keras.

SystemC adalah *library* C++ yang menyediakan berbagai komponen untuk digunakan pada perancangan level transaksi. Kemampuan yang dimiliki SystemC adalah pemrograman sekuensial sebagaimana C++ dan pemrograman *concurrent*. Kemampuan pemrograman *concurrent* ini memungkinkan SystemC untuk digunakan dalam memodelkan komponen perangkat keras dan perangkat lunak yang kompleks.

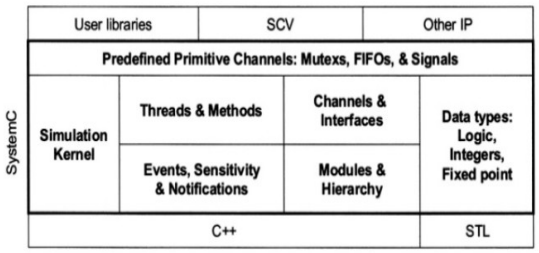
Dibandingkan dengan VHDL dan Verilog, SystemC memiliki kelebihan dalam hal pemrograman terstruktur yang tidak dimiliki oleh VHDL dan Verilog. Dalam hal ini SystemC memiliki fleksibilitas yang lebih tinggi sebagaimana halnya bahasa C++. Kelebihan lain adalah sifat alami SystemC sebagai bahasa C++, sehingga spesifikasi perangkat keras yang ditulis dalam bahasa C tidak perlu lagi diubah ke dalam bentuk lain [3].

Library SystemC digunakan untuk mendukung pemodelan level sistem. SystemC mendukung berbagai level abstraksi dan dapat digunakan untuk perancangan dan verifikasi yang cepat dan efisien. *Library* SystemC disediakan oleh Open SystemC Initiative (OSCI), sebuah organisasi nirlaba yang terbuka. OSCI didukung oleh sejumlah perusahaan, universitas dan individu yang tertarik pada pengembangan pemodelan dengan abstraksi yang lebih tinggi. *Library* SystemC dapat diperoleh secara gratis di www.systemc.org.

Pendefinisan SystemC berada di atas bahasa C++. Terdapat beberapa bahasa inti dan tipe data yang terdapat dalam SystemC dan semuanya dibangun di atas bahasa C++. Bahasa inti terdiri dari *Module/Process*, *Port/Interface, Event, Channel* dan *Event-driven simulation kernel* (gambar 3). Tipe data yang telah didefinisikan terdiri dari *4-valued logic types* (01XZ), *bit/logic vector*, *arbitrary precision integer*, *fixed point* dan tipe-tipe lain yang didefinisikan langsung oleh pengguna berdasarkan bahasa C++.

Pada lapis di atasnya terdapat *elementary channel* seperti *signal, timer, mutex, semaphore* dan FIFO yang menyediakan mekanisme komunikasi antar objek secara *concurrent*.

Module adalah kelas C++ yang membungkus objek perangkat keras atau perangkat lunak. *Module* ini didefinisikan dalam SystemC sebagai kelas sc_*module*. *Module* ini setara dengan *entity/architecture* pada Verilog atau VHDL, yang mewakili suatu blok dasar komponen. Suatu *module* dengan *module* lain berkomunikasi melalui *channel* dan *port*. Dalam *module* terdapat sejumlah proses *concurrent* sebagai implementasi perilaku komponen tersebut.

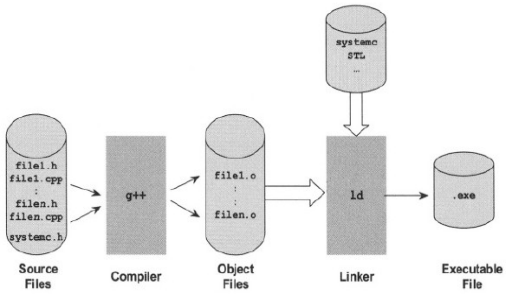


Gambar 3. Arsitektur SystemC [4].

Port adalah objek yang terdapat dalam *module*, yang berfungsi menghubungkan *module* dengan dunia luar. *Port-port* yang telah didefinisikan dalam SystemC adalah sc_in<>, sc_out<>, sc_inout<>, sc_fifo_in<>, sc_fifo_out<> dll.

Terdapat dua jenis proses yaitu : SC_*METHOD* dan SC_THREAD. Keduanya mirip, hanya berbeda dalam masalah pewaktuan. SC_*METHOD* tidak dapat dihentikan sementara pada saat eksekusi sedangkan SC_THREAD dapat dihentikan sementara pada saat eksekusi. Proses kompilasi SystemC dapat dilihat pada gambar 4.

Channel adalah media komunikasi SystemC. Sifatnya lebih umum dibandingkan sinyal. Beberapa *channel* dalam SystemC seperti sc_signal, sc_fifo, sc_semaphore, dan lain-lain. Prinsip komposisi sama dengan perancangan hierarki. Pada sistem yang kompleks, komposisi sangat diperlukan. *Channel* menghubungkan komponen-komponen perancangan. Pada saat ini konsep *channel* sedang dikembangkan pada proses TLM (*Transaction Level Modeling*).



Gambar 5. Proses kompilasi SystemC [4].

Aturan-aturan yang dibangun untuk transformasi dari pemodelan UML ke dalam pemodelan SystemC yaitu, pertama setiap *class* dalam UML dibuat *module* dalam SystemC (SC_*MODULE*). Kedua, setiap atribut *class* dalam UML dibuat *port* dalam SystemC (SC_IN, SC_OUT, SC_INOUT). Ketiga, setiap nama atribut *class* dalam UML dibuat nama *port* dalam SystemC. Keempat, setiap tipe atribut *class* dalam UML dibuat tipe data dalam SystemC. Kelima, setiap *method* dalam UML dibuat *method* dalam SystemC (SC_*METHOD*). Keenam, setiap *class* sc_channel dalam UML dibuat *channel* dalam SystemC (SC_CHANNEL). Ketujuh, setiap *class virtual* dalam UML dibuat sebuah *interface* dalam SystemC.

Di samping aturan-aturan di atas terdapat aturan fungsional SystemC lainnya seperti melengkapi lingkungan pemrograman dalam SystemC dengan empat buah aturan, seperti yang akan dijelaskan berikut:

Pertama, memasukkan systemc.h sebagai salah satu *file* dalam definisi awal : include. (gambar 5)

```
#include "systemc.h"
```

Gambar 5. definisi awal : include.

Kedua, Adanya proses utama dalam SystemC yang melingkupi keseluruhan sistem yaitu modul sc_main (gambar 6)

```
int sc_main(int argc, char ** argv)
```

Gambar 6. Modul sc_main.

Ketiga, instansiasi modul dalam program utama (gambar 7).

```
Tipemodul nama_modul("teks modul");
```

Gambar 7. Instansiasi modul dalam program utama.

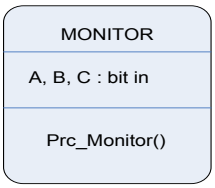
Keempat, memasukkan perintah menjalankan simulasi (gambar 8).

```
Start ();
return 0;
```

Gambar 8. Perintah menjalankan simulasi.

3. Hasil dan Pembahasan

Berikut ini perancangan sederhana dari UML yang akan ditransformasi ke dalam SystemC. Sebuah *class* dalam UML dengan nama monitor yang memiliki atribut A, B, C, Clk:



Gambar 9. Contoh kelas Monitor.

Berdasarkan aturan satu maka Monitor menjadi satu *module* dalam SystemC yang ditunjukkan pada gambar 10 di bawah ini:

```
SC_MODULE(monitor)
{
SC_CTOR(monitor) // konstruktor
{
}
}
```

Gambar 10. Monitor menjadi satu *module* dalam SystemC.

Berdasarkan aturan dua, tiga, dan empat maka atribut A, B dan C menjadi *port* dalam SystemC dengan tipe bit in yang ditunjukkan pada gambar 11 di bawah ini:

```
sc_in<sc_bit> A, B, C;
```

Gambar 11. A, B dan C menjadi *port* dalam SystemC dengan tipe bit in.

Berdasarkan aturan lima maka metode Prc_Monitor menjadi satu sc_*method* dalam SystemC (Gambar 12):

```
SC_METHOD (Prc_monitor)
```

Gambar 12. Metode Prc_Monitor menjadi satu sc_*method* dalam SystemC.

Hasil transformasi dari kelas Monitor dalam UML secara keseluruhan dapat dilihat di gambar 13:

```
SC_MODULE(monitor)
{
sc_in<sc_bit> A, B;
sc_in<sc_bit> C;
void prc_monitor()
{
}
SC_CTOR(monitor)
{
SC_METHOD
(prc_monitor);
sensitive << A<<B <<C;
}
};
```

Gambar 13. Hasil transformasi dari kelas Monitor dalam UML secara keseluruhan.

Berdasarkan aturan tambahan, maka ditambahkan beberapa baris program untuk menjalankan systemC yang dapat di lihat pada gambar 14:

```
#include "systemc.h"
int sc_main(int argc, char ** argv)
monitor M("monitor");
sc_start();
return 0;
```

Gambar 14. Tambahan beberapa baris program untuk menjalankan systemC.

Sehingga hasil akhir transformasi dari UML ke dalam systemC adalah sebagai berikut:

```
#include "systemc.h"
SC_MODULE(monitor)
{
sc_in<sc_bit> A, B;
sc_in<sc_bit> C;
void prc_monitor()
{
}
SC_CTOR(monitor)
{
SC_METHOD (prc_monitor);
sensitive << A<<B <<C;
}
};
int sc_main(int argc, char ** argv)
{
monitor M("monitor");
 sc_start();
 return 0;
 }
```

Gambar 15. Hasil akhir transformasi dari UML ke dalam systemC.

Setelah proses kompilasi dengan menggunakan IDE Evan yang mengakomodasi *library* SystemC 1.0 maka diperoleh hasil sebagai berikut:

```
Compile
Compiling...
main.cpp
main.cpp:27:3: warning: no newline at end of file
Linking...

test2.exe - 0 error(s), 1 warning(s)
```

Gambar 16. Hasil kompilasi dengan IDE Evan.

Hasil kompilasi di atas menunjukkan bahwa proses transformasi *class* dari UML ke dalam SystemC berhasil dilakukan. Dalam implementasi bahasa pemrograman, khususnya C++, biasanya modul program dibagi menjadi dua bagian yaitu definisi modul dan isi dari modul.

Dalam rangka memudahkan pengelolaan modul-modul dalam pemrograman, definisi modul biasanya diletakkan dalam *file header*. Pada contoh kasus yang dibahas pada *paper* ini, transformasi yang dilakukan akan menghasilkan *file* definisi modul. *File* ini lebih tepatnya sebagai *file header*.

Pada contoh kasus transformasi kelas monitor terdapat *method* prc_monitor(). *Method* ini didefinisikan dalam *header*. Pendefinisian lebih rinci dari *method* prc_monitor() ditulis dalam modul yang berbeda. Dalam pemrograman C++, rincian program biasanya ditulis dalam *file* .cpp.

Dalam rangka mengakomodasi keperluan ini maka diperlukan aturan tambahan yang mengatur isi dari *method* yang ada pada setiap *class*. Aturan ini disebut aturan transformasi *method* yaitu:
Pertama, untuk setiap implementasi sebuah *method* dibuatkan sebuah *file*.cpp

```
Prc_monitor.cpp
```

Gambar 17. *File*.cpp.

Kedua, pada awal *file* .cpp disisipkan *file header* yaitu nama kelas nya.

```
# include “monitor.h”
```

Gambar 18. Nama kelas *file header.*

Ketiga, definisi fungsi:

```
void prc_monitor()
{
}
```

Gambar 19. Definisi fungsi.

Pada pemrograman systemC, di samping *header* dalam *file*.h dan rincian modul dalam *file*

.cpp, sebagai *top level* yang menghubungkan seluruh modul adalah modul utama yang di dalamnya terdapat definisi sc_main().

Pada level paling atas akan terdapat tiga buah komponen pemrograman yaitu:
Pertama, definisi modul : kumpulan *file header* (.h)
Kedua, definisi rinci masing-masing modul : kumpulan *file* .cpp
Ketiga, Definisi *top level* : *file* main.cpp

Pada modul *top level* definisi awal adalah menyisipkan *header* systemC (systemc.h) dan semua modul ke dalam definisi *include*. Setelah pendefinisian *top level*, modul utama diletakkan pada modul *top level* dengan mendefinisikan instruksi berikut:

```
int sc_main(int argc, char ** argv)
```

Gambar 20. Instruksi modul utama diletakkan pada modul *top level.*

Pada contoh di atas transformasi kelas monitor pada gambar 9 akan menghasilkan, pertama *file header* monitor.h yang mendefinisikan:

```
SC_MODULE(monitor)
     {
     sc_in<sc_bit> A, B;

     sc_in<sc_bit> C;

      void prc_monitor()
     {
     }
      SC_CTOR(monitor)
     {

     SC_METHOD (prc_monitor);
  sensitive << A<<B <<C;
    }
    };
```

Gambar 21. Definisi *file header* monitor.h.

Kedua, *file* implementasi modul dalam prc_monitor.cpp yang berisi:

```
#include "monitor.h"
  Void (prc_monitor)
  {
  }
```

Gambar 21. Isi *file* implementasi modul dalam prc_monitor.cpp.

Ketiga, definisi *top level* : main.cpp yaitu:

```
#include "systemc.h"
#include "monitor.h"
int sc_main(int argc, char ** argv)
{
        monitor M("monitor");
        sc_start();
      return 0;
}
```

Gambar 23. Definisi *top level* : main.cpp.

4. Kesimpulan

Hasil penelitian menunjukkan bahwa transformasi dari UML ke dalam SystemC memungkinkan dilakukan dengan menggunakan aturan-aturan (*rules*) transformasi. Dengan otomatisasi proses transformasi dari UML ke dalam SystemC memungkinkan untuk mempercepat proses perancangan *embedded system*.

Acknowledgement

Maman Abdurohman mengucapkan terima kasih kepada Fakultas Informatika – IT Telkom dan Fakultas STEI ITB atas dukungan finansial dan perangkat penelitian sehingga penelitian ini dapat diselesaikan.

Referensi

[1] H.D. Patel, "Ingredients for Successful System Level Automation & Design Methodology," Doctoral Dissertation, Virginia Polytechnic Institute and State University, United States, 2007.

[2] J. Cornet, "Separation of Functional and Non-Functional Aspects in Transactional Level Models of Systems-on-Chip," Doctoral Dissertation, Institut Polytechnique De Grenoble, France, 2008.

[3] A.S. Berger, *Embedded System Design: An Introduction to Processes, Tools, and Techniques*, CMP Books, USA, 2002.

[4] D.C. Black & J. Donovan, *SystemC : From the Ground Up*, Kluwer Akademic Publisher, USA, 2004.