BEHAVIOR ENGINEERING

Niken Listya Pratiwi – 0706272055

Abstrak

Masalah yang sering ditemui dalam *software engineering* adalah memodelkan *requirement* dengan benar. Paper ini menjelaskan secara singkat pendekatan Behavior Engineering atau yang sebelumnya disebut Genetic Software Engineering. Pendekatan tersebut memodelkan *requirement* yang informal ke dalam bentuk formal. Konsep yang ditawarkan sederhana, selain itu juga menawarkan kemampuan dalam mendeteksi cacat/masalah pada *requirement* lebih dini, kontrol terhadap kompleksitas, dan mengakomodasi adanya perubahan. Selain itu juga dijelaskan secara singkat mengenai Integrare, *environment* yang mendukung pendekatan ini.

Keywords: *behavior tree,behavior engineering, genetic software engineering, integrare*

1 Pendahuluan

Metode software engineering yang banyak dipakai sekarang masih kurang dalam mendeliver konsistensi dan cukup kompleks. Pada UML, misalnya, memiliki lebih dari satu diagram untuk merepresentasikan *requirement*. Hal ini menyebabkan peningkatan kompleksitas saat mendesain sistem. Oleh karena itu, dibutuhkan alternatif atau metode baru yang dapat menanggulagi hal tersebut sehingga dapat mengurangi kompleksitas saat pembuatan desain.

Idenya adalah merepresentasikan *requirement* dengan baik, konsisten dan tidak ambigu. Hal itu dicapai dengan memformalkan deskripsi dari *requirement* yang diinginkan. Tantangan untuk memformalisasikan *requirement* adalah sebagai berikut [5]:

- akurasi, seberapa tepat apa yang direpresentasikan dengan yang sebenarnya diinginkan
- validasi, stakeholder dapat dengan mudah memvalidasi hasil representasi tersebut apakah sesuai dengan yang diinginkan
- kompleksitas, menghindari *short term memory overflow*, dengan secaran runtut dalam menggambarkan apa yang diinginkan
- cacat/masalah pada requiremenr, membuat masalah terdeteksi lebih awal
- pemahaman, dapat melihat apa yang diinginkan secara menyeluruh
- pembagian kerja, memperlihatkan pembagian kerja dengan jelas untuk meningkatkan produktivitas tim.

Itu semua bisa dilakukan bila kita membuat sistem dari *requirement*nya. Requirement menggambarkan perilaku dari yang diinginkan sedangkan sistem menunjukkan perilakunya. Ada dua

jenis perilaku, perilaku komponen itu sendiri dan perilaku *network* (sekumpulan komponen). Perilaku komponen menggambarkan tindakan (aksi) yang dilakukan komponen tersebut. Sedangkan untuk perilaku *network* komponen memenggambarkan interaksi pada sekumpulan komponen dengan adanya perpindahan/penyaluran kontrol.

Dengan begitu dibutuhkan representasi *requirement* secara formal. Hal itu selain akan memudahkan dalam validasi dan tahap desain, juga akan memudahkan dalam pengimplementasian. Representasi formal dengan menggunakan *behavior tree* dalam pendekatan Behavior Engineering memiliki maksud tersebut.

2 Behavior Engineering dan Behavior Tree

2.1 Definisi

Behavior Engineering [5, 8] menggunakan pendekatan dengan notasi grafik yang disebut Behavior Tree untuk memformalkan dan mengintegrasikan keseluruhan *requirement* dari sistem dalam satu struktut tree.

Behavior Tree [1] adalah sebuah grafik formal mirip tree yang merepresentasikan perilaku dari sebuah atau sekumpulan entitas tersebut menyadari/memiliki keadaan/state, mengubah state, membuat keputusan, menanggapi maupun menyebabkan suatu event terjadi, dan berinteraksi dengan memindahkan kontrol.

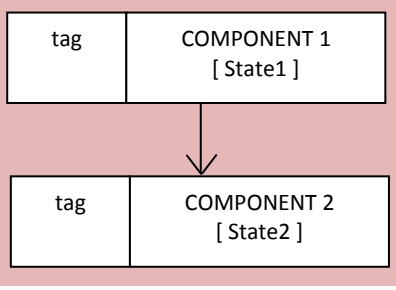
2.2 Sintaks dari Behavior Tree

Secara singkat sintaks dari behavior tree [1, 4, 5, 6] adalah sebagai berikut:

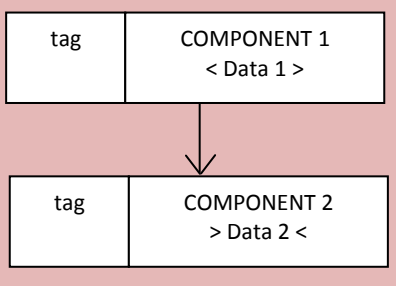
	Semantics	Contoh
State – Realization / Internal State tag | COMPONENT [State]	Komponen "COMPONENT" mencapai keadaan "State" kemudian menyalurkan kontrol ke output dari komponen	Telepon dalam keadaan on atau menyala | TELEPON [On]
Attribute – Assignment / Attribute State tag | COMPONENT [Attribute := Value]	Komponen "COMPONENT" memberikan sebuah nilai "Value" pada "Attribute", salah satu atribut dari komponen.	Telepon dengan nomor | TELEPON [No := 021123456]
Conditional – flow of control / IF State	Komponen "COMPONENT" menyalurkan control ke	Jika telepon menyala

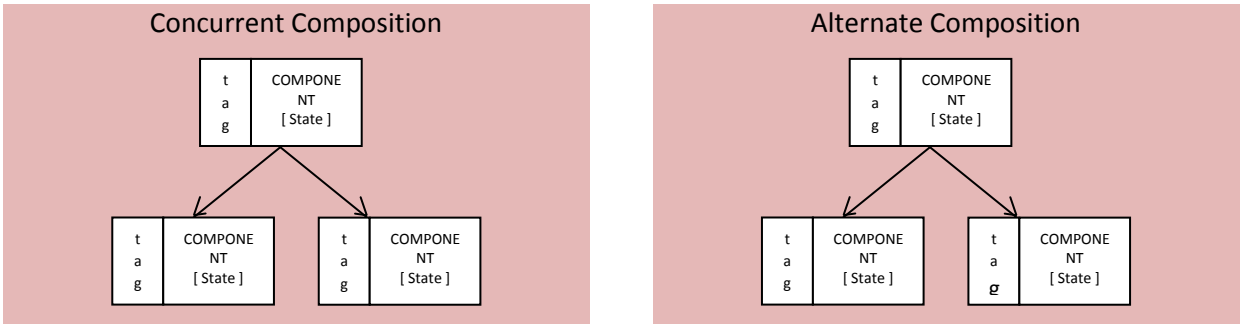
tag | COMPONENT ? Condition ?	outputnya hanya bila memenuhi kondisi “Condition”.	| TELEPON ? On ?
Event – flow of control / WHEN State tag | COMPONENT ?? Event ??	Komponen “COMPONENT” menyalurkan kontrol ketika **dan** jika “Event” terjadi setelah mencapai component-state ini	Saat telepon berdering | TELEPON [Berdering]
Guard – flow of control / WHEN State tag | COMPONENT ??? State ???	Komponen “COMPONENT” menyalurkan kontrol ketika **atau** jika “Event” terjadi setelah mencapai component-state ini	Saat telepon berdering | TELEPON [Berdering]
Data Ouput State tag | COMPONENT < Data >	Komponen COMPONENT mengeluarkan output Data ke komponen penerima yang terhubung dengan outputnya	Telepon mengirimkan pesan | TELEPON < Pesan >
Data Input State tag | COMPONENT < Data >	Komponene COMPONENT memasukkan input Data dari komponen pengirim yang terhubung dengan inputnya	Telepom menerima pesan | TELEPON > Pesan <
System State – Realization tag | SYSTEM [State]	Komponen SYSTEM mencapai keadaan State kemudian menyalurkan kontrol ke outputnya	| TELEPON [Off]

Sequential Composition



Data-flow Composition





Tabel 1 Sintaks dari Behavior Tree

Apa yang direpresentasikan oleh notasi *behavior tree* tertuang dalam bentuk tree sederhana dari komponen-komponen. Tiga tipe komunikasi *interprocess* disampaikan disini: variabel yang dipakai bersama (*shared variable*), sinkronisasi, dan penyaluran pesan (*message passing*). *Behavior tree* juga memiliki *semantic formal* yang berdasarkan *low-level process algebra, behavior tree process algebra* (BTPA) [6]. *Behavior tree* dapat juga digunakan untuk mendukung simulasi dari model-checking dan penghasilan kode program [1,7].

2.3 Pendekatan dengan Menggunakan Behavior Tree

Pada setiap *individual requirement* dilakukan translasi ke representasi formalnya yaitu dalam bentuk *requirement behavior tree*. Setelah pentranslasian yang dilakukan selanjutnya adalah integrasi terhadap *requirement behavior tree (RBT)* tersebut menjadi bentuk *design behavior tree (DBT)*.

Diagram 1 Translasi Requirement dari Informal Menjadi Formal

Baik dalam translasi maupun integrasi akan dapat mengidentifikasi adanya cacat/masalah yang sangat susah ditemukan pada informal *requirement*. Cacat/masalah disini adalah adanya ambiguitas, alias, ketidaklengkapan, ketidakkonsistenan, ketidakakuratan, dan redundansi (perulangan) yang dapat dideteksi lebih dini [3].

Cacat/Masalah	Saat translasi	Saat Integrasi
Ambiguitas	Masalah ambiguitas ini terjadi karena kurangnya konteks dalam saat menyampaikan *requirement* yang	Seringnya ambiguitas terdeteksi dengan tidak lengkapnya informasi selama integrasi, sehingga statement

	diinginkan, hal ini akan bisa diidentifikasi saat translasi karena bentuk yang formal tidak mentoleransi adanya hal tersebut.	yang ambigu akan menyulitkan pemodelan prakondisi.
Alias	Terjadi ketika kata yang berbeda digunakan untuk mendeskripsikan entitas, keadaan/state, aksi, event yang sama. Contoh: di salah satu *requirement* dituliskan sebagai menyala dan di *requirement* lain On.	Satu komponen memiliki dua nama yang berbeda, saat pengintegrasian hal ini akan terlihat.
Tidak lengkap	Dapat diidentifikasi dengan tidak adanya perilaku alternatif	Biasanya terdeteksi dengan tidak ditemukannya kondisi sebelum dan sesudah dari komponen tersebut
Tidak akurat	Salah satu contohnya adalah pemberian nilai range atribut yang tidak sesuai	Dalam hal ini perilaku yang tidak sesuai dengan konteksnya
Tidak konsisten	Terdeteksi ketika suatu *requirement* ditemukan tidak konsisten dengan dirinya sendiri	Saat mengintegrasikan dua atau lebih *behavior tree* yang tidak konsisten menjadi satu *behavior tree* akan menyebabkan adanya kontradiksi, masalah ini dapat dicek dengan *model-checking*
Redundansi	Tidak terdapat masalah ini dalam translasi.	Bisa dikarenakan adanya redundan *requirement* atau dikarenakan adanya ketidakkonsistenan

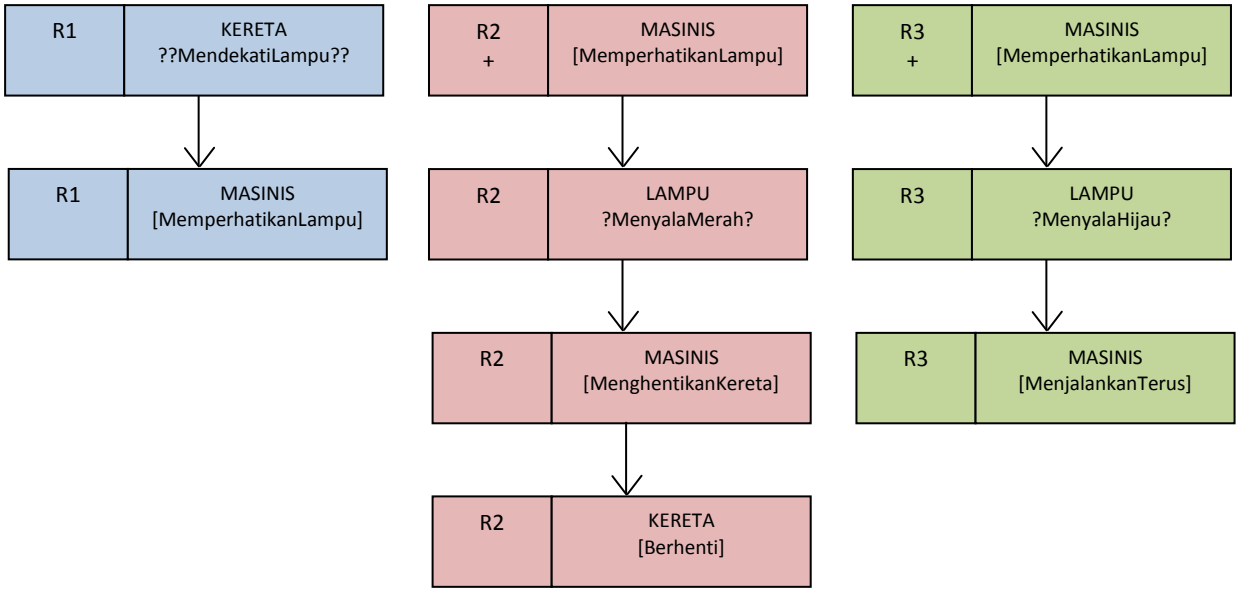
Tabel 2 Cacat yang Terdeteksi Saat Translasi dan Integrasi

Contoh proses pemodelan *requirement* RBT kemudian menjadi DBT:

R1: Ketika kereta mulai mendekati lampu, masinis memperhatikan sinyal lampu dari kejauhan

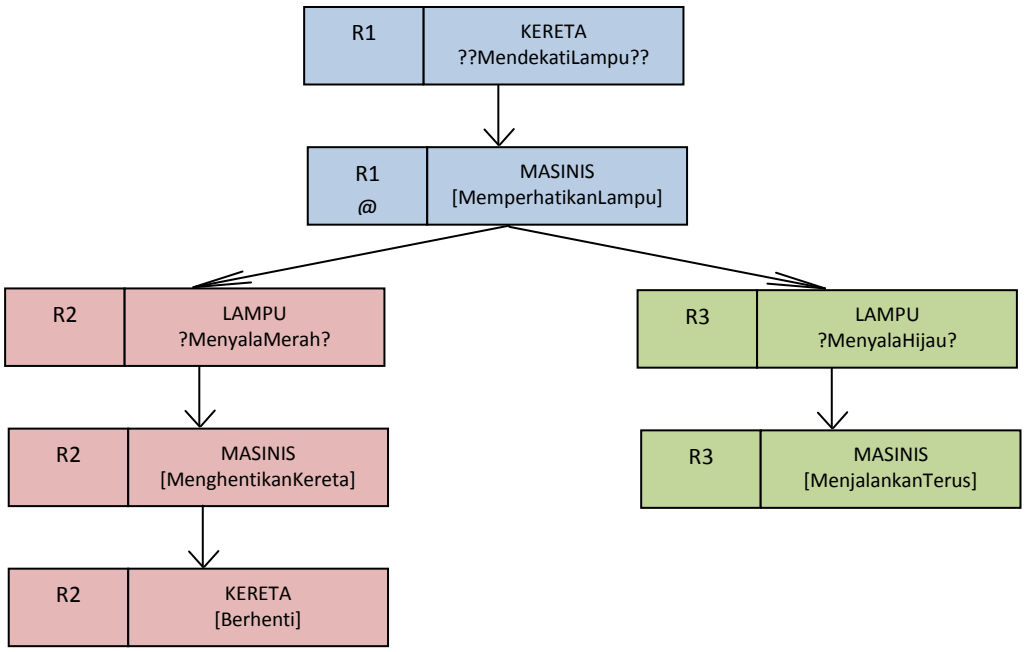
R2: Bila sinyal lampu merah yang menyala, masinis menghentikan kereta

R3: Bila sinyal lampu hijau yang menyala, masinis menjalankan terus keretanya



Gambar 1 Translasi R1, R2, R2 ke dalam Bentuk Requirement Behavior Tree

Hasil *design behavior tree*-nya adalah sebagai berikut:



Gambar 2 Design Behavior Tree

Setelah DBT terbentuk, selanjutnya adalah mentransformasikannya ke bentuk *component architecture*-nya yang disebut *component interaction network* (CIN) yang menggambarkan hubungan antar komponen dalam sistem dan memberikan sebuah tampilan dari arsitektur komponen. Dari DBT itu pula diproyeksikan bentuk *component behavior trees* (CBTs) dan *component interface*

diagrams (CIDs) yang merupakan tampilan-tampilan dari komponen secara individual. CIN, CBTs, CIDs ini bersama-sama merepresentasikan *component level design* dari system. Penghasilan proyeksi tersebut dapat automatis dilakukan dengan menggunakan "Integrare" [7].

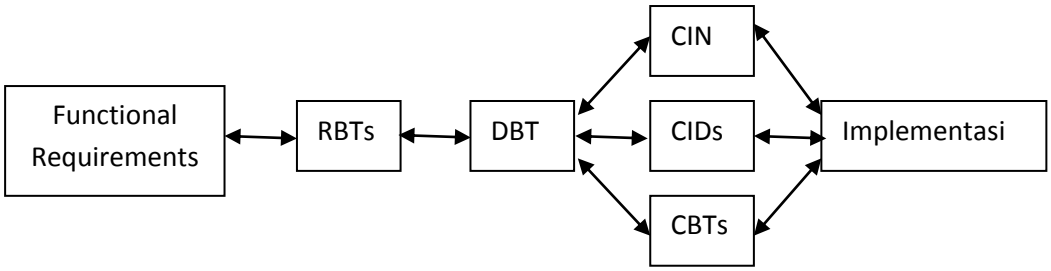


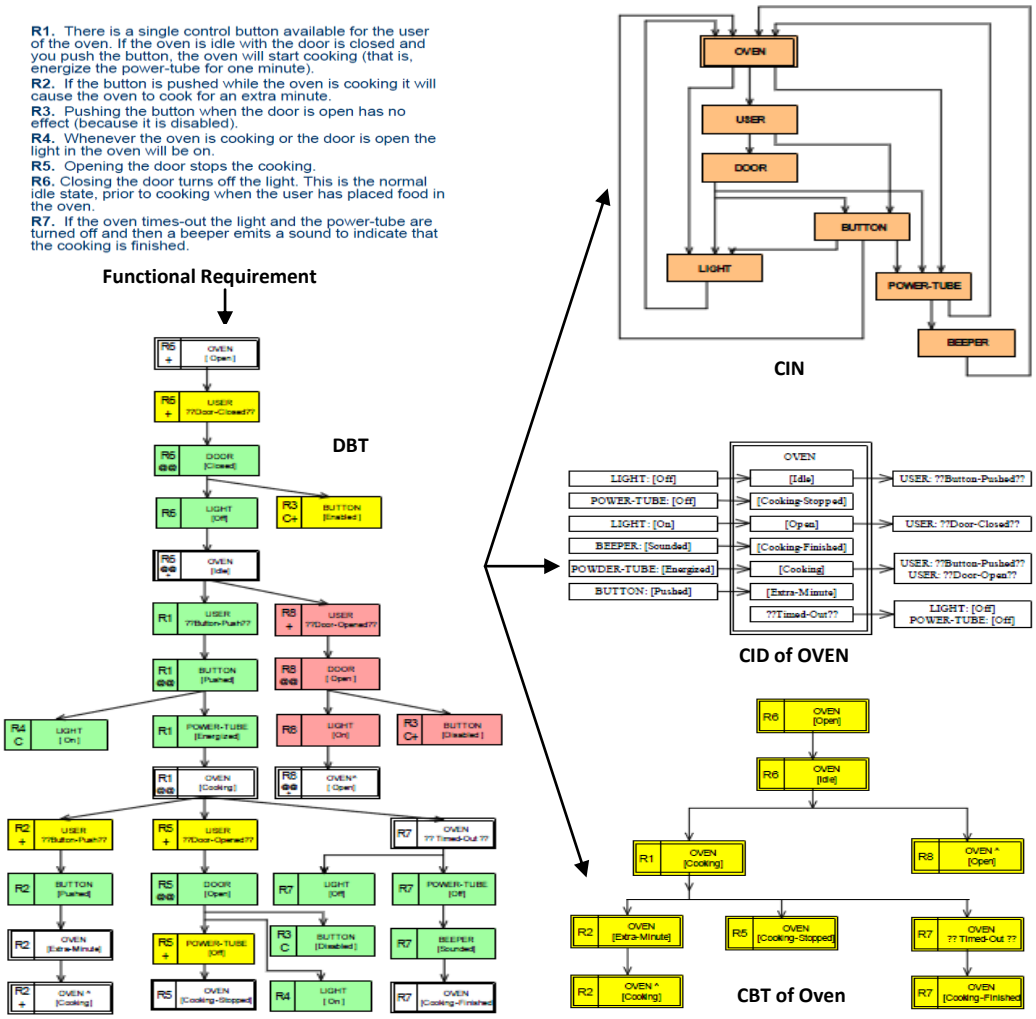
Diagram 2

Integrare merupakan *environment* yang mendukung Behavior Engineering yang menggunakan pemodelan dengan *behavior tree*. Integrare ini mengintegrasikan beberapa *tools* yang dapat digunakan pada fase-fase dalam proses desain, seperti *requirement engineering*, simulasi, spesifikasi formal, dan juga *model-checking*. Selain itu, environment ini mendukung pengerjaan kolaborasi yang memperbolehkan beberapa *user* untuk melakukan bagian pekerjaannya masing-masing seperti melakukan translasi *requirement*, memperbaiki *tree*, dan sebagainya.

Mulai dari tahap translasi *functional requirement* hingga pemroyeksian desain diagram dapat menggunakan *tools* yang tersedia pada Integrare. Misal translasi *requirement* ke RBT dibantu dengan tool *Requirement Translation Assistant* (RTA). Validasi DBT oleh klien dapat dibantu dengan *simulation tool*. Model-Checking juga dapat dilakukan pada DBT ini untuk memastikan pemenuhan *safety* maupun *performance*.

Perkembangan lebih lanjut mengenai Integrare sampai saat ini telah sampai pada tahap *source-code generation*. Integrare memungkinkan untuk secara otomatis menghasilkan *formal specification languages* seperti SAL dan BTSL. Dengan begitu memungkinkan juga untuk mentranslasikannya ke bahasa seperti Java atau C++. [7]

Berikut merupakan CIN, CIDs, CBT yang dapat di-*generate* secara langsung dari DBT-nya. Contoh studi kasus yang diberikan adalah mengenai OVEN yang untuk detail lebih lanjut dapat dilihat pada paper [2].



Gambar 3

5 Perbandingan

Beberapa perbedaan dan kelebihan dari pendekatan Behavior Engineering menggunakan *behavior tree* ini dibandingkan dengan metode lain adalah:

- *Design behavior tree* mendeskripsikan seluruh perilaku dari sistem. Selain itu, ia juga mengandung seluruh informasi yang dibutuhkan untuk membangun diagram yang lain. Dengan adanya hal itu dapat memungkinkan terjadinya proses pengupdatean otomatis terhadap diagram lain bila terjadi perubahan pada *requirement* hanya dengan mengupdate *design behavior tree*-nya. Hal ini tidak memungkinkan terjadi pada UML. [2]
- Desainer dapat lebih fokus pada detail di setiap *individual requirement* tanpa perlu mengkhawatirkan detail pada *requirement* yang lain. Requirement tersebut dapat diproses dalam satu waktu baik pentranslasian maupun pengintegrasian yang tentunya akan mengurangi kompleksitas dalam pembuatan desain. Hal ini tidak bisa ditemukan pada UML maupun metode lainnya. [4]

- Dibandingkan dengan *Use-case* dan representasi dari scenario, integrasi jauh lebih baik dalam menemukan cacat/masalah pada *functional requirement.* [2]
- Jika dibandingkan dengan metode lain, notasi dari behavior tree merupakan bentuk formal yang dapat dengan mudah dimengerti oleh non-experts sekalipun. [7]

Kesimpulan

Studi lebih lanjut mengenai penerapan pendekatan Behavior Engineering beserta *environment*-nya perlu dilakukan. Hal ini dikarenakan kemungkinan besar keduanya dapat menjadi solusi dalam masalah yang banyak terjadi dalam *software engineering*. Di kemudiannya diharapkan keduanya dapat menggantikan metode dan *tools* lain yang diterapkan saat ini.

Referensi

[1] Dromey, R.G., "*From Requirements Change to Design Change: Formalizing the Key Steps",* 2003. Website: www.behaviorengineering.org/publications/dromey/K1-Dromey.pdf. Diakses: 19 November 2010

[2] Wen, L., Dromey, R.G., "*From Requirements Change to Design Change: A Formal Path",* 2004. Website: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1347509. Diakses: 12 November 2010

[3] Powell, D., "*Requirements Evaluation Using Behavior Trees - Findings from Industry*", 2007. Website: http://www.behaviorengineering.org/docs/ASWEC07_Industry_Powell.pdf. Diakses: 12 November 2010

[4] Dromey, R.G., *"Formalizing The Transition from Requirements to Design",* 2007. Website: http://www.behaviorengineering.org/publications/dromey/Dromey-Chapter-Final-20051.pdf. Diakses: 19 November 2010

[5] Dromey, R.G., "*Engineering Large-Scale Software-Intensive Systems",* 2007. Website: http://www.behaviorengineering.org/publications/ . Diakses: 19 November 2010

[6] Colvin, R., Hayes, I.J., "*A Semantic for Behavior Tree",* 2007. Website: http://www.accs.uq.edu.au/documents/TechnicalReports/ACCS_TR_07_01.pdf. Diakses: 4 November 2010

[7] Wen, L., Colvin, R., Lin, K., Seagrott, J., Yatapanage, N., Dromey, R.G., *"Integrare, a Collaborative Environment for Behavior-Oriented Design"*, 2007. Website: http://www98.griffith.edu.au/dspace/bitstream/10072/18625/1/43991_1.pdf. Diakses: 4 November 2010

[8] Zafar, S., Colvin, R., Winter, K., Yatapanage, N., Dromey, R.G., *"Early Validation and Verification of a Distributed Role-Based Access Control Model",* 2007. Website: http://ieeexplore.ieee.org/iel5/4425817/4425818/04425884.pdf?arnumber=4425884. Diakses: 12 November 2010