VERIFIKASI DAN SERTIFIKASI PERANGKAT LUNAK BERBASIS KOMPONEN

Ario Santoso, Daniel Cahyadi, Richard Lokasasmita
Fakultas Ilmu Komputer, Universitas Indonesia

ABSTRAKSI

Pengembangan perangkat lunak berbasis komponen merupakan pilihan yang sangat menarik bagi para pengembang dan pemilik sistem, karena dapat menurunkan waktu dan biaya pengembangan. Namun cara pengembangan seperti ini masih menemui banyak permasalahan dalam prosesnya, khususnya dalam hal verifikasi dan sertifikasi komponen.

Artikel ini akan memberikan pengantar mengenai sebuah alat bantu yang dibangun dengan tujuan untuk dapat menerapkan sertifikasi dan verifikasi pada komponen perangkat lunak.

1. PENDAHULUAN

Saat ini, trend penggunaan komponen dalam pengembangan perangkat lunak semakin meningkat. Berbagai keuntungan ditawarkan dengan mengembangkan perangkat lunak berbasis komponen: waktu pengembangan yang lebih singkat, biaya pengembangan yang biasanya lebih murah, dan meningkatnya kegunaan dari produk berbasis komponen [3]. Komponen itu sendiri dapat didefinisikan sebagai sebuah unit komposisi yang dapat digunakan oleh pihak ketiga tanpa perlu mengetahui detail implementasinya, cukup melalui spesifikasi yang menyertai komponen tersebut.

Dengan motivasi meniru keberhasilan proses produksi alat-alat elektronika ataupun perangkat keras lainnya yang berbasis komponen, dimana hasil produksi dapat meningkat dengan pesat, proses pengembangan perangkat lunak pun mulai melakukan hal yang sama. Namun hal ini masih belum dapat terlaksana dengan baik, karena masih banyak masalah-masalah yang ditemui dalam teori maupun prakteknya. Masalah yang biasa ditemui dalam proses pengembangan perangkat lunak antara lain, cara menemukan komponen yang sesuai dengan kebutuhan, jaminan komponen berfungsi dengan baik (tersertifikasi), teknik dalam mengintegrasikan komponen-komponen yang ada menjadi satu sistem yang utuh, hingga pemeliharaan dari komponen itu sendiri.

Penelitian ini akan berfokus pada masalah standardisasi untuk sertifikasi komponen dan pengujian kebenaran dari komposisi komponen. Hingga saat ini, konsorsium yang mengembangkan CORBA (sebuah standard framework) baru menawarkan standardisasi *syntactic*. Standardisasi ini hanya mengatur bagaimana satu komponen dapat digabungkan dengan komponen lain sehingga dapat menghasilkan suatu aplikasi yang baru. Padahal dalam pengembangan aplikasi yang *reliable* kebutuhannya tidak hanya sampai pada tahap *syntactic*, melainkan dibutuhkan standardisasi *semantic* yang secara formal matematis menjamin bahwa komposisi yang dihasilkan dapat berfungsi sesuai dengan yang diharapkan dan tidak mengganggu kerja dari masing-masing komponen.

Oleh karena itu, sebuah *prototype* alat bantu pengembangan perangkat lunak berbasis komponen akan dibuat untuk dapat menerapkan standardisasi [1] dan memberikan fasillitas untuk melakukan pengujian sertifikasi *semantic* dari design yang dihasilkan. Dengan alat bantu tersebut, design sertifikasi dapat diterapkan dalam permasalahan industri Indonesia. Proses ini diharapkan dapat menyiapkan Indonesia dalam menyongsong era pengembangan perangkat lunak yang diproyeksikan kelak berbasis komponen [4].

Bagian 2 dari paper ini akan menjelaskan mengenai *state of the art* dari penelitian ini, yang akan memberikan gambaran mengenai hasil penelitian-penelitian sebelumnya. Bagian 3 akan menjelaskan rancangan *prototype* alat bantu yang akan dibuat dan pada bagian 4 menyimpulkan isi dari paper ini.

2. CVT – STATE OF THE ART

Sertifikasi perangkat lunak dapat menjamin keamanan dan kepercayaan suatu sistem perangkat lunak. Sertifikat ini akan mencakup semua informasi yang dibutuhkan untuk penilaian secara independen dari berbagai properti yang dimiliki oleh perangkat lunak tersebut [5].

Hasil riset dari [6,7,8,9,10] mengajukan beberapa formalisasi yang mengindikasikan standardisasi sertifikasi dari sisi *semantic*. Namun hasil riset tersebut masih kurang memperhatikan aspek-aspek pragmatis dalam pengembangan perangkat lunak berbasis komponen seperti yang disinyalir oleh [4,11]. Oleh karena itu , diajukanlah standardisasi [1,2] yang merupakan gabungan baik dari sisi pragmatis, maupun sisi teoritis. Dari sisi pragmatis, dibutuhkan standardisasi yang memudahkan pengembangan perangkat lunak dari sisi tingkat kesulitan maupun biaya/waktu, sementara dari sisi teoritis dibutuhkan standardisasi yang menjamin kebenaran dan keakuratan perangkat lunak untuk dapat diuji secara formal matematis.

Oleh karena itu, dibutuhkan adanya alat bantu yang dapat memudahkan penerapan metode formal matematis yang sulit, agar dapat diterapkan di industri perangkat lunak. Sebenarnya, alat bantu pengembangan perangkat lunak berbasis komponen

sudah banyak yang populer digunakan antara lain, Visual Studio+supernova [13] dan SUN ONE [14]. Namun alat-alat bantu tersebut belum ada yang dapat menguji kebenaran aplikasi yang dihasilkan, dan menguji apakah sertifikasi *semantic* yang dimiliki tiap-tiap komponen sudah sesuai atau tidak [11]. Padahal bila alat bantu tersebut dapat melakukan sertifikasi *semantic* secara otomatis, maka proses pengujian secara formal matematis dapat menjadi lebih cepat dan murah.

Pada saat ini ada beberapa konsep komponen perangkat lunak yang sudah populer digunakan, yaitu Javabeans/Enterprise Java Beans [Sun Microsystem] dan .NET Framework [Microsoft]. Namun konsep komponen yang ada saat ini, belum mendukung proses verifikasi komponen secara *semantic*. Agar proses tersebut dapat dilakukan, maka perlu dilakukan modifikasi dengan memberikan informasi tambahan yang dibutuhkan oleh proses verifikasi secara *semantic*. Bentuk tambahan informasi yang diberikan, dapat berupa berkas terpisah ataupun berupa modifikasi internal dari komponen itu sendiri.

3. COMPONENT VERIFICATION TOOLS

Saat ini sedang dikembangkan *tools* yang akan digunakan untuk melakukan verifikasi dan juga sertifikasi komponen, yang pada akhirnya akan sangat membantu dalam pengembangan perangkat lunak berbasiskan komponen. Program ini dituliskan menggunakan bahasa pemograman Java. Java telah dikembangkan cukup lama dan sangat matang dalam konsep *object oriented approach* yang ditawarkan. Secara umum *tools* yang dikembangkan terdiri dari empat buah bagian (*packages*), yakni:

3.1 Bean Builder

Package Bean Builder ini menyerupai dengan text editor pada umumnya. Namun yang membedakan *bean builder* adalah pembuatan komponen, yang dalam hal ini merupakan *java bean*, yang dilakukan pada *bean builder* memperhatikan aspek *semantic* dari *bean* yang dikembangkan. *Tools* yang selama ini ada telah memperhatikan aspek *syntactic* dari komponen, namun masih mengabaikan aspek *semantic* komponen. Sebagai analogi apabila pengembang hendak membuat komponen "Pintu" yang salah satu attributnya adalah jenis kayu yang digunakan, maka terjadi perbedaan informasi yang dapat disimpan antara *bean builder* dengan *tools* pada umumnya. Apabila *tools* yang dikembangkan hanya memperhatikan aspek *syntactic* dari "Pintu", maka informasi yang tersimpan dapat dituliskan sebagai berikut:

Component	: Pintu
Attributes	: {String : Jenis_Kayu, …}
Methods	: {Method A, Method B, …}

Namun informasi ini belum cukup untuk melakukan verifikasi komponen secara semantik, yang termasuk memperhatikan *constraints* yang dimiliki oleh komponen tersebut. Oleh sebab itu pada *bean builder* selain menyimpan informasi tersebut, maka komponen akan secara *self-defined* menyimpan *constraints* dari dirinya sendiri. Berikut ini adalah contoh informasi yang akan disimpan:

Component	: Pintu
Attributes	: {String : Jenis_Kayu, …}
Methods	: {Method A, Method B, …}
Constraints	: {Jenis_Kayu : {Jati, Mahoni}, …}

Untuk melakukan pengecekan terhadap *constraints* komponen, maka akan digunakan apa yang disebut sebagai *bean inspector*. *Bean inspector* akan menggunakan teknik yang disebut dengan *reflection*.

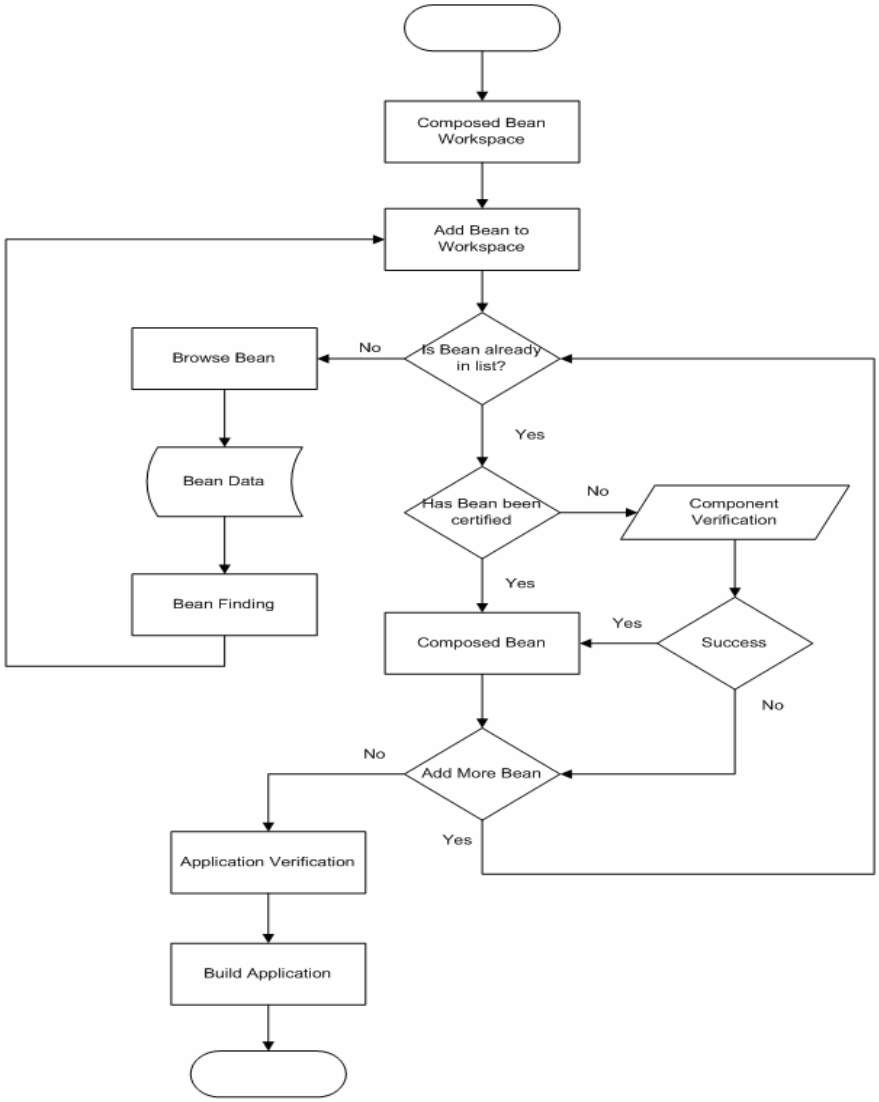
Selain itu, *bean builder* juga akan melakukan verifikasi tiap komponen yang dibuat. Untuk melakukan verifikasi tersebut akan digunakan *formal methods.* Apabila suatu komponen telah lolos tahap verifikasi maka komponen tersebut dikatakan telah tersertifikasi. Untuk lebih jelasnya mengenai mekanisme verifikasi dan sertifikasi yang dilakukan, akan dijelaskan lebih lanjut pada bagian 3.4 paper ini.

3.2 Bean Composer

Package Bean Composer dapat dikatakan bagian inti dari *tools* yang dikembangkan. Untuk dapat mengembangkan perangkat lunak berbasiskan komponen, maka diperlukan *tools* untuk menggabungkan komponen-komponen yang telah ada untuk menjadi satu aplikasi. *Bean Composer* merupakan bagian *tools* ini yang memenuhi fungsi tersebut.

Untuk dapat menggabungkan satu komponen dengan komponen lainnya pada *Bean Composer* terdapat syarat yang harus dipenuhi. Setiap komponen yang hendak digabungkan tersebut harus telah menjalani proses sertifikasi (telah memenuhi suatu *standard* tertentu).

Apabila terdapat komponen yang belum memiliki status tersertifikasi hendak digabungkan dengan komponen lain, maka *Component Verification Tools* (CVT) ini akan "memaksa" agar komponen tersebut melalui proses verifikasi terlebih dahulu. Sehingga hanya komponen yang tersertifikasi yang menyusun aplikasi yang dikembangkan.



Gambar 1. Flow Chart *Bean Composer*

Perangkat lunak yang baik merupakan perangkat lunak yang telah melalui tahap *testing*, dan *formal method* menyediakan metode untuk melakukan *testing* tersebut. Oleh sebab itu, proses verifikasi tidak hanya diperuntukkan bagi komponen penyusun aplikasi, melainkan termasuk untuk aplikasi itu sendiri.

Alur yang terjadi pada saat menyusun aplikasi dengan *bean composer* dapat dilihat pada gambar 1.

Dengan merujuk pada *flow diagram* yang terdapat pada gambar 1, dapat dilihat bahwa saat pengguna sistem hendak membuat suatu aplikasi berbasiskan komponen, maka pengguna perlu terlebih dahulu mempersiapkan komponen-komponen yang akan digunakan. Saat pengguna hendak memasukkan komponen (*bean*) ke dalam *workspace*, maka komponen haruslah komponen yang tersertifikasi. Apabila terdapat komponen yang belum tersertifikasi, maka komponen tersebut perlu diverifikasi terlebih dahulu sebelum digunakan. Terdapat kemungkinan komponen tidak berhasil diverifikasi. Apabila hal ini terjadi, maka komponen tersebut tidak dapat digunakan, sehingga untuk dapat menggunakan komponen itu, pengguna harus melakukan perubahan terhadap komponen.

Setelah aplikasi telah terbentuk, maka sistem akan melakukan verifikasi aplikasi secara keseluruhan. Hal ini perlu dilakukan karena dalam pengembangan perangkat lunak berbasiskan komponen perlu melakukan verifikasi tidak hanya terhadap komponen penyusunnya, melainkan juga terhadap hasil komposisi dari komponen-komponen tersebut [12].

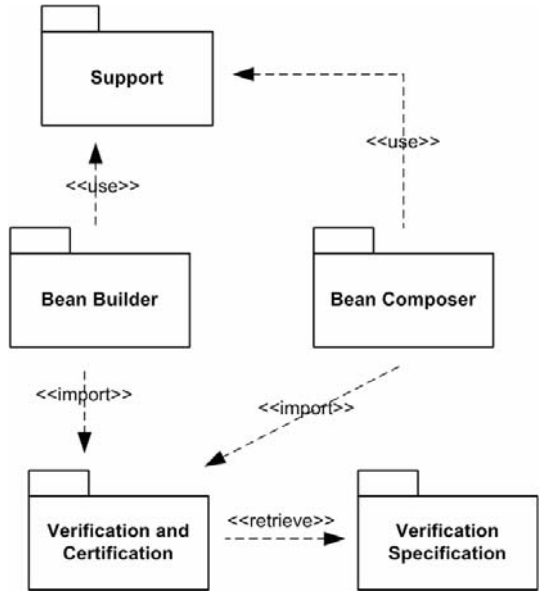
3.3 Support

Package Support berisikan fungsi-fungsi pendukung dari *bean builder* dan juga *bean composer*. Secara umum fungsi-fungsi dari *support* menangani masalah *event* yang terjadi dalam CVT *tools* dan *listener* yang menangkap *event* tersebut. Namun disamping fungsi tersebut, terdapat juga fungsi-fungsi untuk tampilan dan fungsi terhadap pembukaan berkas (*file*).

3.4 Gambaran CVT Secara Keseluruhan

Masing-masing *packages* yang telah disebutkan sebelumnya saling mendukung satu sama lain dalam menjalankan fungsinya. Untuk mendapatkan gambaran bagaimana keterhubungan yang terjadi, dapat dilihat pada arsitektur dari *CVT tools* pada gambar 2.

Merujuk gambar 2 dapat dilihat bahwa *bean builder* dan *bean composer* menggunakan fungsi-fungsi umum yang telah didefinisikan pada *support*. Dengan demikian akan mengurangi redundansi dari fungsi. Untuk melakukan verifikasi komponen baik yang dilakukan pada *bean builder* ataupun *bean composer*, maka akan digunakan fungsi yang terdapat pada *verification and certification*.



Gambar 2. CVT Architecture

4. PENUTUP

Pada paper ini, pendekatan untuk melakukan verifikasi dan sertifikasi dengan bantuan *tools* telah dilakukan. Pendekatan ini dilakukan dengan pembuatan *CVT Tools* yang merupakan gabungan antara *bean builder, bean composer,* dan kelak akan digabungkan pula dengan *certification and verification tools*. Untuk proses verifikasi dan sertifikasi yang dilakukan, digunakan dengan pendekatan *formal methods*.

Hingga saat ini pengembangan *tools* masih terus dilakukan dan belum mencapai versi akhir. Direncanakan riset ini akan memakan waktu 3 tahun sejak Februari 2007. Sejalan dengan riset yang dilakukan, *tools* ini akan terus mengalami peningkatan fungsionalitas yang bersesuaian.

5. AKTIFITAS SELANJUTNYA

Berangkat dari pengerjaan *tools* yang telah dikerjakan hingga saat ini, untuk langkah selanjutnya kami akan melakukan pengembangan modul sertifikasi dan verifikasi berdasarkan paper [2].

Ucapan Terima Kasih

Penelitian ini didanai oleh Menristek Republik Indonesia dengan nomor kontrak 106A/DRPM-UI/N1.4/2007 untuk pelaksanaan program insentif riset terapan.

PUSTAKA

[1] A. Azurat and I.S.W.B. Prasetya. A survey on embedding programming logics in a theorem prover. *Technical Report CS-UU-2002-007*, Institute of Information and Computing Science, University of Utrecht, 2002

[2] A.Azurat and I.S.W.B. Prasetya. Development agreement for decomoposition of progress property in component software. *Draft*.

[3] Ivica Crnkovic, Magnus Larsson. Component-based Software Engineering State of Art Report. *Technical report*, Malardalen University, Idt, 2000.

[4] Clemens Szyperski, Dominik Gruntz and Stephan Murer. *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley/ACM Press, 2002. ISBN 0-201-74572-0

[5] Ewen Denney, Bernd Fischer. Software Certification and Software Certificate Management Systems, In *Proceedings of 2005 ASE Workshop on Software Certificate Management*. Long Beach, CA, pp. 1-5, Nov. 2005.

[6] M. Broy. Multi-view modelling of software systems. In Hung Dang Van and Zhiming Liu, editors, *Proceedings of the Workshop on Formal Aspects of Component Software (FACS), 2003*. Also as UNU/IIST Report no.284.

[7] He Jifeng, Liu Zhiming, and Li Xiaoshan. Component Calculus. *Technical Report 285*, UNU-IIST, September 2003.

[8] He Jifeng, Liu Zhiming, and Li Xiaoshan. Contract-oriented component software development. *Technical Report 276*, UNU-IIST, P.O.Box 3058, Macau, 2003.

[9] Zhiming Liu, Jifeng He, and Xiaoshan Li. Contract Oriented Development of Component, In Jean-Jaques LA'evy, Ernst W. Mary, and John C. Mitchell, editors. Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, *TCI 3rd International Conference on Theoretical Computer Science (TCS2004)*, 22-27 August 2004, Toulouse, France. Kluwer, 2004. Pages 349-366.

[10] Juliana Kuster Filipe. A Logic-based Formalization for Component. *Journal Object Technology*, 1(3):231-248, 2002.

[11] Piotr Makowski and Anders P. Ravn. *Component based Development – Where's The Place for Formalization?* UNU/IIST, 2003.

[12] A.Azurat. Towards Reliable Component Software: Light-weight Formalism. *Submitted to 8th Int'l QIR*, 2005.

[13] www.**microsoft**.com

[14] www.**sun**.com