Transformasi *XML* dengan Skema *Relax-NG* menjadi Komponen *GUI* pada Kustomisasi Modul *OpenERP* disertai Uji Performa-nya

Yosua Alvin Adi Soetrisno
Jurusan Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
yosuaalvin.mti13@mail.ugm.ac.id

Selo Sulistyo
Jurusan Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
selo@ugm.ac.id

Ridi Ferdiana
Jurusan Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia
ridi@ugm.ac.id

***Abstrak* - *Enterprise Resource Planning* (*ERP*) memiliki modul aplikasi yang bisa dikustomisasi. Modul aplikasi pada produk *OpenERP* dibangun dengan konsep *Model-View-Controller* (*MVC*). Komponen *view* pada konsep *MVC*, digunakan untuk merepresentasikan *graphical user interface* (*GUI*) dari aplikasi, yang pada umumnya dibentuk dari struktur e*Xtensible Markup Language* (*XML*). Struktur dokumen *XML*, yang dituliskan dalam bentuk *tree maps* akan sulit untuk dipahami. Skenario transformasi *XML* menjadi komponen *GUI* dilakukan untuk membantu pengguna mendapatkan persepsi bagaimana komponen tersebut akan ditampilkan pada aplikasi sebenarnya.**

Pada transformasi akan dilakukan *mapping* deklaratif dari elemen pada *tag-tag XML* menjadi komponen *GUI* tertentu. Transformasi *XML* ke komponen *GUI* bisa dibantu oleh *tools* penggambar diagram seperti *DIA* yang bisa membaca beberapa format *XML*. Visualisasi yang diperoleh dari hasil transformasi ini, akan mempermudah proses kustomisasi, karena visualisasi memberikan gambaran bagaimana susunan komponen pada aplikasi.

Kata kunci - *enterprise resource planning*, transformasi, *XML*, komponen *view*, *GUI*, *mapping*, *diagramming tools*.

I. PENDAHULUAN

Aplikasi *ERP* memiliki struktur data yang kompleks, sehingga untuk mengembangkannya membutuhkan waktu yang lama. Pemahaman karateristik struktur data diperlukan agar bisa membangun tampilan *GUI* dengan baik. Salah satu contoh produk *ERP* adalah *OpenERP*. *OpenERP* menggunakan struktur data yang dikembangkan berdasarkan konsep *MVC*. Konsep *MVC* digunakan, agar perubahan pada komponen *view*, tidak mengubah proses penanganan data, demikian juga sebalik-nya, dan dapat memisahkan akses dan representasi data dengan interaksi komponen[1].

Sesuai konsep MVC, komponen *view* pada *OpenERP* dibentuk dari dokumen *XML*. *XML* merupakan *markup language* yang digunakan sesuai *grammar* tertentu, yang bisa digunakan untuk bertukar informasi secara mudah, karena menggunakan konteks yang bisa dibaca, dengan prinsip pemisahan konten dari presentasinya [2].

Masalah pada penggunaan *XML* adalah, walaupun *XML* baik untuk pertukaran data, namun memerlukan proses tambahan untuk memvisualisasikan sebuah aplikasi. Visualisasi memberi gambaran bagaimana komponen disusun dalam tatanan aplikasi, sedangkan *XML* hanya merepresentasikan sekumpulan elemen penyusun visual yang terstruktur dalam bentuk tree [3].

XML yang dipakai pada *OpenERP* merupakan *XML* jenis *data-oriented*. *XML* jenis *data-oriented* memiliki beberapa *tag* yang dikenali aplikasi menurut skema tertentu. Dengan adanya skema, bentuk komponen tidak terlalu dinamis, tetapi mengikuti kaidah aplikasi *form-based*, sehingga pada proses transformasinya bisa diterapkan aturan yang jelas. *OpenERP* memakai skema *Relax-NG* untuk membedakan karakteristik masing-masing komponennya.

Karakteristik komponen dan elemen *field* yang dipakai pada *XML OpenERP* dijelaskan oleh atribut yang mengikuti *tag* dan informasi yang terletak di antara *tag* merupakan label dari sebuah komponen [4]. Komponen *view* pada *framework OpenERP* bisa dibedakan menjadi *menu*, *generic view*, *dynamic view*, *action*, *access right*, dan *workflow*. Setiap kelompok akan memiliki atribut khusus yang berbeda, sesuai hasil transformasinya.

Pada proses transformasi akan dilakukan pengenalan dan pengelompokan komponen, sesuai dengan atribut pada masing-masing *tag*, kemudian akan memanggil fungsi pada *tools diagramming DIA*, untuk memilih komponen yang sesuai untuk diletakkan pada sebuah lembar kerja. Hasil transformasi pada lembar kerja bisa disimpan menjadi *XML* yang dikenali *DIA* dengan tambahan *properties sizing* dan *positioning* sesuai letak dan ukurannya pada kanvas.

Pada penelitian ini, akan dibuat skrip transformasi yang dapat membantu proses visualisasi dari *XML* menjadi komponen *GUI* pada *OpenERP*, dibantu oleh *tools diagramming DIA*, yang menyediakan *icon* untuk masing-masing komponen.

Makalah ini disusun sebagai berikut. Bagian 2 akan menjelaskan kajian pustaka. Bagian 3 akan menjelaskan metodologi dalam melakukan transformasi. Bagian 4 akan menjelaskan analisis fitur dan performa transformasi. Bagian 5 akan menjelaskan kesimpulan dan saran pengembangan.

II. KAJIAN PUSTAKA

Kustomisasi pada produk *ERP* merupakan proses mengganti, mengembangkan, atau menambahkan fitur khusus yang belum tersedia pada modul *on-the-self* (bawaan) *ERP*. Kustomisasi pada konsep *MVC* dilakukan dengan memodifikasi komponen *model*, *view*, atau *controller* menggunakan bahasa pemrograman tertentu. Pada OpenERP, *model* akan diwakili struktur data pada *Structured Query Language* (*SQL)*, *view* akan dijelaskan pada file *XML*, serta *controller* untuk mengatur interaksi terdapat pada kode *Python* [1].

. Modifikasi tampilan dilakukan dengan melaku-kan *inheritance view* dan menambahkan deskripsi komponen pada file *XML*-nya [5].

Visualisasi *XML* menjadi tampilan grafis akan mengurangi beban kognitif dari pengguna yang ingin melakukan kustomisasi [6]. *XML OpenERP* menggunakan skema *Relax-NG* yang terletak pada file *view.rng*. *Relax-NG* dikembangkan dari *Document Type Definition* (DTD) menjadi skema yang perbedaannya terdapat pada Tabel 1 [7].

Beberapa sistem berusaha melakukan visualisasi dari dokumen *XML* seperti *Angur* [3].*Angur* menggunakan *X-path* untuk mengakses jalur dari *tree* yang dipilih. *Angur* mengubah file *XML* menjadi sebuah *graph* interaktif yang *node*-nya bisa diubah-ubah. Transformasi *XML* pada *Angur* dilakukan oleh *Java Parser* (*JAXP*). Komponen *graph* digambar oleh *Graph Manager* yang terdiri dari *library* grafis untuk menggambarkan *interface*.

TABEL 1. PERBANDINGAN *DTD* DAN SKEMA [7]

Perbedaan	***DTD***	**Skema**
Sintaks	unik singkat tidak ada *tag* perlu dipahami	standart lengkap menggunakan *tag* mudah ditemukan
Penulisan	belum ada *datatype*	membuat dan menggunakan *datatype* secara bebas
Kemunculan Batasan	menggunakan ?,*,+ untuk menjelaskan batasan 0 atau 1	hanya memerlukan atribut *minOccurs* dan *maxOccurs* untuk memberikan batasan
Enumerasi	deklarasi ringkas ketika elemen tidak disertakan	tipe enumerasi disertakan pada konten elemen

Visualisasi bentuk tree secara interaktif bisa dilakukan juga dengan menampilkan *Document Object Model* (*DOM*) [8] pada grafik *tree* yang bisa dijelajahi. Pendekatan ini menjadi ide untuk menggunakan konsep *Unified Modelling Language* (*UML*) dalam memodelkan *meta-data* dari *XML*. Konsepnya adalah dengan menambahkan atribut dari kelas dan sub elemen-nya sebagai kumpulan elemen *list* [4].

Visualisasi yang lain ada pada *User Interface* (UI) *Creation Pattern DSL* to *GUI transformation*[9]. *Domain Extraction Algorithm* (*DEAl*) yang dipakai bisa melakukan ekstraksi fitur dari *Domain Specific Language* (DSL) domain tertentu untuk kemudian diterjemahkan ke domain *GUI*. *DEAl* menggunakan algoritma *traversal* [9].

. Konten domain dari *UI* diekstrak sebagai *graph* yang memiliki relasi dan *properties*. *Tools* ini melakukan tiga langkah yaitu menjelajahi *GUI* dari aplikasi *Java*, mengekstrak informasi menjadi domain *form*, dan membuat tata bahasa, model, dan *parser* sesuai informasi domain [10].*GraphML* juga melakukan transformasi *XML* dengan menambahkan data baru menggunakan atribut *key/data*[11]. *GraphML* menggunakan *parseinfo*[11]. untuk mengimplementasikan *parser* dari dokumen *GraphML*, sehingga *meta-data* tambahan bisa disertakan pada atribut *XML* dari elemen tertentu.

Pada penelitian ini, sesuai dengan kajian pustaka, akan melakukan metode *parsing* yang sederhana untuk memperoleh *value* dari atribut yang spesifik dari masing-masing komponen dan menggunakan *graphic library custom shape*, yang dibentuk dari komponen *OpenERP*. *Custom shape* akan dibuat pada *tools DIA*. Konsep *parseinfo* [11]. akan diterapkan ke *tools DIA*, sehingga komponen bisa diberi atribut tambahan yang dibutuhkan *tools*, selain atribut bawaan, seperti atribut ukuran dan posisi.

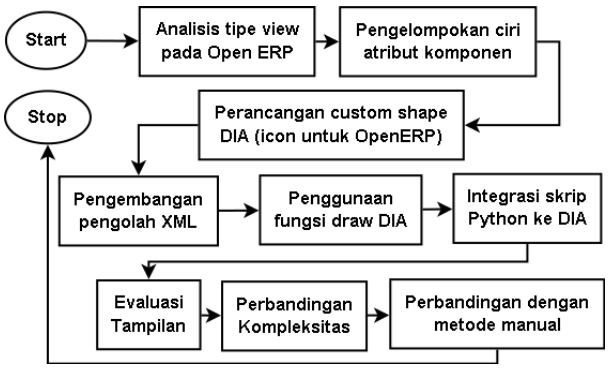
III. METODOLOGI

A. Alat dan Bahan Penelitian

Alat yang digunakan adalah *tools diagramming DIA*. Bahan yang digunakan adalah data *XML* yang akan ditransformasi dan diuji diperoleh dari paket *OpenERP* versi 6.1 yang tersedia di http://nightly.odoo.com/old/openerp-6.1/6.1.20140804/. Data ini terdiri dari 443 file *XML view* yang kemudian dibagi sesuai dengan tipe *view*-nya menjadi 1014 file *XML*.

B. Kerangka Kerja Pengembangan Tools Tranformasi

Pada penelitian ini, digunakan kerangka kerja rekayasa perangkat lunak yang dijelaskan pada *flowchart* Gambar 1. Tahap awal adalah analisis, yang bertujuan untuk memahami ciri dan atribut komponen yang digunakan di *OpenERP*, sehingga nantinya bisa diidentifikasi oleh skrip transformasi. Selain itu jenis tampilan yang digunakan seperti *form view* dan *grid view* juga perlu dianalisis, agar ketika dirancang, bisa sesuai dengan pola pada aplikasinya. Tahap ini juga akan menganalisis fungsi yang bisa dilakukan *tools DIA* untuk membuat objek.



Gambar 1. *Flowchart* Garis Besar Penelitian

Pada tahap perancangan, setelah komponen dari *XML* bisa dibedakan dan dikelompokkan sesuai ciri atributnya, maka akan dibuat komponen grafis yang bisa mewakili komponen yang dijelaskan pada XML. Perancangan tersebut dilakukan dengan membuat *custom shape* pada *tools DIA*. *Custom shape* dibuat dengan memberi *link icon* serta menentukan ukurannya, ketika ditampilkan ke layar.

Pada tahap pengembangan, akan dibuat skrip yang menggunakan *library* pengolahan XML yang mempermudah pencarian dan pemisahan *tag*, sehingga *tag* bisa dikelompokkan sesuai ciri dan atributnya. Fungsi pengolahan *XML* ini akan dipermudah dengan proses pemisahan jenis *view* yang bisa dilakukan dengan mengakses *data view* langsung dari *database*. Setelah berhasil dipisahkan, informasi yang ada pada atribut digunakan untuk memilih komponen grafis yang sesuai dan memberi label pada masing-masing komponen. Fungsi-fungsi tersebut akan disatukan menjadi sebuah kelas *display*. Kelas ini akan membuat objek berupa komponen grafis setiap ditemukan komponen yang berlainan.

Pada tahap implementasi maka skrip akan diintegrasikan ke *tools DIA* dan dicoba untuk melakukan proses transformasi dari beberapa *file XML* yang tersedia pada *OpenERP*.

Pada tahap evaluasi dan pengujian, maka akan dilakukan perbandingan tampilan pada aplikasi yang sebenarnya (bisa berupa *desktop application* atau *web application*) dengan tampilan yang ada di *tools DIA*. Apabila tampilan yang ada pada aplikasi sesuai dengan yang ditampilkan di *tools DIA* maka proses transformasi telah berhasil.

Perbandingan kompleksitas dilakukan dengan mencari variasi *tag*, jumlah total *tag*, jumlah atribut dari *tag*, kedalaman *node* dari *tree XML*, dan juga ukuran file *XML*. Variasi dari komponen kompleksitas tersebut akan dibandingkan terhadap performa waktu untuk mengetahui komponen kompleksitas apa yang paling berpengaruh terhadap waktu. Selain itu juga dilakukan perbandingan beberapa *library* pengolah *XML*, untuk mengetahui *library* yang paling efisien mengolah *XML*.

Perbandingan antara penggunaan *tools DIA* dengan proses transformasi dan *rendering* penggabungan komponen grafis juga dilakukan. Hal ini dilakukan untuk mengetahui apakah penggunaan pemrograman *Python* berbasis kelas (*OOP*), efektif untuk melakukan transformasi dibanding dengan proses transformasi prosedural.

C. Perancangan Custom Shape pada DIA

DIA akan memperlakukan setiap *shape* sebagai objek yang memiliki *properties* tertentu. *DIA* memiliki kemampuan untuk menambahkan *custom shape* dengan deskripsi *XML* sederhana. *Interface* dari *OpenERP* bisa diwakili oleh beberapa *icon .png*, yang mewakili struktur *header*, *toolbar*, *field*, *relation field*, *button*, dan *list* seperti pada Gambar 2. *Custom shape* disimpan dalam format *.shape* dengan penambahan atribut *Scalable Vector Graphic* (*SVG*) yang sering dipakai pada *tools* pengolah gambar.

Gambar 2. Komponen *Interface OpenERP*

D. Pengembangan Pengolah XML dengan Library LXML

LXML merupakan *library* dari *Python* yang memiliki *binding* dengan *library C*, *libxml2*, dan *libxslt*. *Library* ini unik karena mengkombinasikan kecepatan dan fitur *traversal XML* yang sederhana dengan *Python Application Programming Interface* (*API*). *API* ini dikenal dengan *ElementTree API* [12].

Library ini menggunakan komponen *Element* yang mewakili sebuah *tree*. *Element* bisa membawa objek dengan fleksibel, dan dirancang untuk menyimpan struktur data hirarki di memori. *Element* merupakan persilangan antara *list* dan *dictionary*.

Fungsi *LXML* yang digunakan pada penelitian ini adalah *fromstring()*, *attrib.get()*, dan *tag()*. Fungsi *fromstring()* digunakan untuk membuat *Element* dari string. Dengan fungsi ini maka akan diperoleh *Element* baru yang merepresentasikan seluruh bagian *XML*. Fungsi *fromstring()* ini bisa diterapkan pada *dictionary*, sehingga bisa mengambil bagian tertentu dari keseluruhan *XML*.

Setiap *instance* dari kelas *Element* memiliki beberapa atribut yang merupakan sebuah *dictionary*. *Dictionary* memiliki *keys* dan *value*, misalnya untuk atribut {nama : "Yosua"} maka memiliki *keys* = nama dan *value* = Yosua. Fungsi *get()* pada sebuah *instance* dari elemen digunakan untuk memperoleh *value* dari *keys* tertentu. Kelebihan dari metode ini adalah bisa menyediakan nilai *default* bila elemen tidak memiliki *keys* yang ingin diperoleh. Fungsi *tag()* digunakan untuk menemukan *tag* tertentu pada dokumen XML. Misalnya untuk menemukan *tag* nama maka bisa digunakan fungsi *if etree.tag = "nama"*.

E. Implementasi Python Script to Generate View

DIA bisa menggunakan skrip *Python* sebagai *plug-in*, oleh karena itu skrip transformasi akan didaftarkan sebagai salah satu *sub-menu* di *tools DIA*. Proses yang dilakukan setelah *sub-menu* dijalankan adalah mengakses *database OpenERP* melalui proses *Remote Procedure Call* (*XML-RPC)*. Pembacaan *view* dari database lebih cepat, karena

pada database tipe *view* sudah dibedakan. Prosedur *TreeView* akan membantu pengguna dalam menelurusi seluruh *view OpenERP* yang disajikan dalam tampilan *tree*.

Masing-masing tipe *view* akan memiliki *child element* dan atribut yang bisa dipisahkan berdasarkan *tag*-nya. Pemisahan *tag* ini akan digunakan untuk memberikan *style* dan juga melakukan pengaturan tampilan dari masing-masing komponen. Proses pengaturan tadi akan diatur dalam sebuah kelas *display* yang menggunakan beberapa metode *draw* untuk membuat objek di *tools DIA*.

IV. ANALISIS DAN HASIL

F. Proses Pemilihan View dan Pemisahan Tag

XML view pada *database* akan dikelompokkan berdasarkan tipe *view*-nya. Pengelompokan ini akan menyederhanakan proses pencarian atribut pada data *XML*. Pada proses manual, file *XML* harus dipisahkan menjadi bagian *tree*, *form*, atau *search*, setiap ada *tag*, <*field name="arch" type="xml"*>. Tipe *tree* akan mengarah kepada beberapa tipe *search* yang ada.

Variabel *view* akan menyimpan *string XML* yang sudah dipisah, setelah itu akan dilakukan *query many2one* untuk mendapatkan komponen yang berelasi dalam sebuah tampilan. Proses ini secara jelas bisa dilihat pada *flowchart* Gambar 3.

G. Class display pada Transformasi

Fungsi lengkap dari inisiasi kelas bisa dilihat pada Gambar 4. Perbedaan lengkap dari metode yang ada pada kelas *display* bisa dilihat pada Tabel 2. Fungsi *draw_element* akan memberikan kete-rangan *default value* dari semua *field*. Fungsi *draw_element* juga menggambarkan *notebook_path* yang menandakan ada beberapa *pages* yang ada dalam satu tampilan layar kerja.

Fungsi *draw* akan menambahkan komponen yang pasti ada pada aplikasi seperti *head_logo* juga memanggil *process* (*form*, *tree*, atau *search*) untuk menggambar lengkap komponen yang ada pada sebuah *form*, *tree*, atau *search view*. Fungsi *draw* juga menambahkan elemen *toolbar* di sebelah kiri.

Fungsi *process* (*form*, *tree*, *search*) akan memisahkan *tag-tag* menggunakan fungsi *attrib.get('string')*, dan setelah itu akan dideteksi apakah *tag* dari elemen berupa *newline*, *field*, *button*, *search*, *newline*, *separator*, *filter*, dsb. Fungsi yang dilakukan method *process* secara rinci adalah sebagai berikut:

- Elemen yang mempunyai *tag field*, atribut "*name*"-nya akan digunakan untuk memberi keterangan komponennya.
- *Mandatory field* akan menentukan warna dari elemen *field* tersebut.
- Jika dalam *field* ada relasi *many2one*, maka atribut "*name*"-nya, akan dicek apakah mempunyai *widget* khusus yang harus ditampilkan.
- Masing-masing komponen akan disesuaikan posisinya agar tidak bertumpukan dengan komponen yang lain.

H. HasilVisualisasi View pada Tools DIA

Hasil visualisasi komponen di *tools DIA* bisa dilihat pada Gambar 5. Pada proses transformasi ini, akan dicoba *view* yang kompleks dengan menggunakan contoh aplikasi *Meetings* yang tersedia pada modul *Sales* bagian *CRM*. Aplikasi *Meetings* mempunyai dua *notebook-path* yaitu *Meeting* dan *Invitation Detail* yang terletak pada kotak merah Gambar 5.

Ketika melakukan relasi *many2one* ke *Invitation Detail*, maka akan diperoleh *widget* aplikasi *Invitation*, dan disatukan menjadi satu layar dalam *tools DIA*. Hal ini akan mempermudah *view* dari masing-masing *page* karena disajikan dalam satu layar menurut kaidah *Single Document Interface* (*SDI*) [13].

I. Perbandingan Fitur Transformasi DIA

Pada penelitian *YATE* [14] dijelaskan karakteristik dari *transformation engine* yang diperkenalkan pada [15][16]. *YATE* menggunakan *UsiXML* untuk mengubah *XML* menjadi objek *Java*. *Rules class* akan menjelaskan semua aturan transformasi, namun akan mengalami penurunan performa, karena aturan yang berbeda bisa diterapkan ke objek yang sama dan setiap aturan harus mencari objek ini terlebih dahulu [14] Pada penelitian ini, aturan yang ada pada skrip transformasi baru diterapkan ketika sebuah objek dipilih dari *database*.

Penggunaan *library LXML* dan juga *library* lain seperti *ElementTree*, akan membuat iterasi ke file *XML* menjadi deklaratif. Algoritma di balik *library LXML* adalah *tree traversal*, yang menganggap file *XML* sebagai *Element* yang memiliki *child* dan juga atribut. Berbeda halnya dengan proses menggambar di *DIA*, yang memerlukan suatu yang imperatif, yaitu menjelaskan peletakan posisi.

Kompleksitas *mapping* dipengaruhi oleh karakter dari komponen yang akan ditranslasikan. Komponen yang tidak bisa dikelompokkan menjadi komponen statis dan juga dinamis akan mempersulit penentuan parameternya. *OpenERP* membatasi penggunaan *mapping* pada *view* dengan menyediakan komponen standart yang biasa digunakan pada *form*, sehingga mudah dikelompokkan komponennya. Perbandingan fitur skrip transformasi dengan *transformation engine* yang telah ada sebelumnya dijelaskan pada Tabel 3.

J. Perbandingan Kompleksitas XML OpenERP

Pada penelitian ini komponen *XML* memiliki variasi tingkat kompleksitas masing-masing. Variasi tersebut terdiri dari jumlah *tag*, jumlah variasi *tag*, jumlah atribut yang digunakan pada *tag*, kedalaman *node*, dan juga ukuran file. File yang memiliki jumlah *tag* banyak, belum tentu memiliki jumlah variasi *tag* yang banyak, atau jumlah atribut dari masing-masing tag yang banyak. File yang berukuran besar bisa saja terdiri dari informasi atribut yang panjang namun tidak memiliki kompleksitas variasi dari *node* yang ada. Dari 1014 file *XML* yang ada, akan coba dianalisis untuk 10 file terbesar dengan urutan berdasarkan ukuran file-nya. Susunan kompleksitas akan disajikan pada Tabel 4.

K. Perbandingan Kompleksitas terhadap Performa Waktu

Pada penelitian ini dilakukan 100 iterasi untuk menemukan *tag field* dengan atribut *name=''name''*. Di sini

digunakan berbagai macam *library* yang berguna untuk pencarian *string* pada struktur *tree*. Penggunaan *library LXML* dijelaskan pada [17] *Library* yang lain yaitu *expat*, sering digunakan pada proses pengolahan *bitstream* paralel, yang bisa memproses data *multi-line* secara bersamaan [18] Oleh karena itu hasil tes menggunakan *library expat* menunjukkan performa yang baik.

Library regex menggunakan *regular expression* yang berfungsi untuk mencari *string* berdasarkan pola kecocokan karakter dengan dibantu oleh *wildcard* dan juga kombinasi *list* [19]. *Regex* pada Gambar 6 dengan warna hijau memiliki hasil tercepat untuk melakukan *parsing*. *Regex* bisa diterapkan dengan mudah untuk mencari *tag*, namun tidak sebanding dengan kemudahan yang ditawarkan dalam menggunakan fasilitas *tree traversal*, untuk menemukan berbagai macam atribut. Perbandingan *time performance* terletak pada Tabel 5.

Pada file *sale.orders.tree* terjadi peningkatan waktu pencarian, karena banyaknya jumlah *tag* dipadu dengan jumlah atribut yang banyak juga. Kelebihan *regex* adalah ketika terjadi banyak atribut, tidak terjadi lonjakan waktu pencarian. Selain *regex* penggunaan *library expat_hack* juga perlu dipertimbangkan dalam melakukan *parsing tree*. Pada *library expat_hack* terjadi peningkatan kualitas dari *expat* dengan membuat *dynamic link*, sehingga *expat* melakukan *parsing* ke *dictionary* yang berisi *tag* dan *value*-nya[20].

L. Perbandingan Performa Waktu Proses Transformasi Manual dengan Transformasi DIA Menggunakan Skrip

Pada penelitian ini, akan dilakukan uji performa juga terhadap proses manual yang menggunakan teknologi *parsing* dan juga *rendering* penggabungan komponen. Proses *rendering* disini akan menggunakan program *merge* (C.Fydo, 2006), yang digunakan untuk menggabungkan beberapa file *.png* menjadi sebuah *.png*. Proses *merge* dilakukan dengan menggabungkan *icon* komponen *OpenERP*, yang digambar pada *DIA*. Pada Gambar 7, tranformasi dengan *tools DIA* akan dinamai *proposed* berwarna biru. Transformasi ini ternyata bisa membantu meningkatkan performa waktu proses *rendering* manual, karena pada *tools*, informasi lengkap dari *icon* komponen tidak diperlukan.

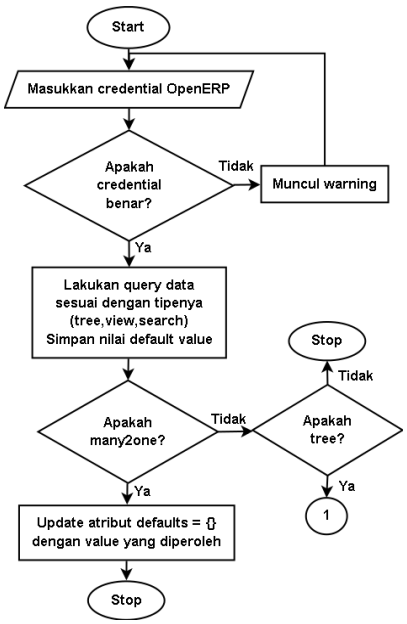
V. KESIMPULAN DAN PENGEMBANGAN

Transformasi dari bentuk *XML* menjadi komponen GUI pada *OpenERP* bisa dilakukan. Transformasi ini nantinya akan membantu pengembangan *model driven* berbasis *form*. Transformasi dari file *XML* ini dibantu oleh library *LXML* yang mempermudah pencarian atribut berdasarkan struktur *tree*. *Tools DIA* menyediakan fasilitas kustomisasi untuk menyediakan *shape* tambahan mempermudah pembuatan *icon* yang mewakili komponen pada *OpenERP*.

Pengujian kompleksitas memberi masukan untuk pengembangan transformasi dengan *regex* maupun *expat-hack* karena memiliki performa yang baik untuk file dengan jumlah atribut yang banyak.

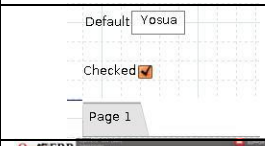
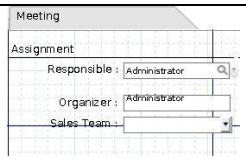
Pengujian *parsing* dan *rendering* secara terpisah memberi gambaran bagaimana *tools DIA* membantu dalam penyederhanaan definisi komponen.

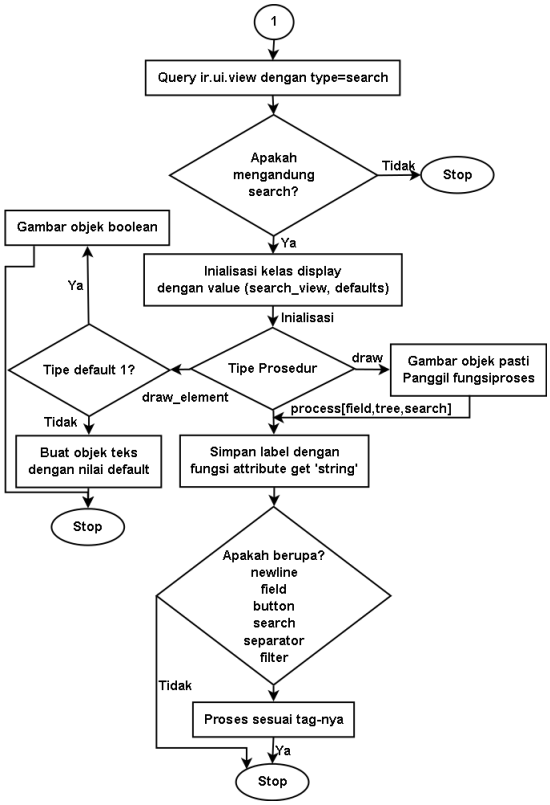
Tantangan ke depan adalah bagaimana bisa menggabungkan penggunaan berbagai *library* untuk teknik *parsing* yang lebih baik dan juga pengembangan *tools diagramming* yang memungkinkan *extend properties* ke domain tertentu, sehingga bisa membantu pengembangan *model driven* tanpa melakukan pemrograman.



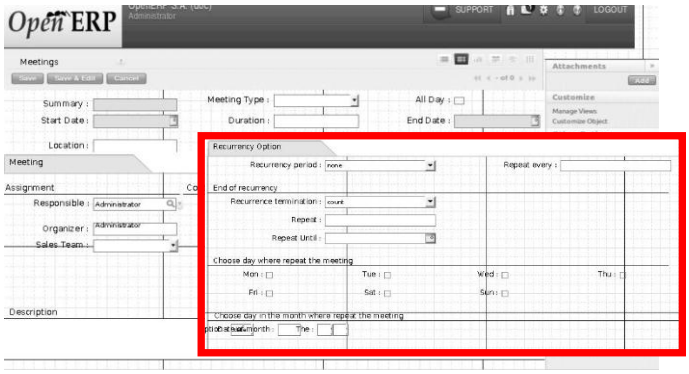
Gambar 3. *Flowchart* pemisahan *tag* dilanjutkan ke Gambar 4

TABEL 2. PERBANDINGAN *METHOD* PADA *SCRIPT IMPORT DIA*

Method	**Komponen**	**Gambar**
draw_element	default value notebook_path	
draw	head_logo title right_toolbar pemanggilan proses	
process	newline, field button, search, newline, separator, filter	



Gambar 4. *Flowchart* inialisasi kelas *display*



Gambar 5. View *Aplikasi Meetings* pada *tools DIA*

TABEL 3. PERBANDINGAN FITUR *TRANSFORMATION ENGINE* (MODIFIKASI [14])

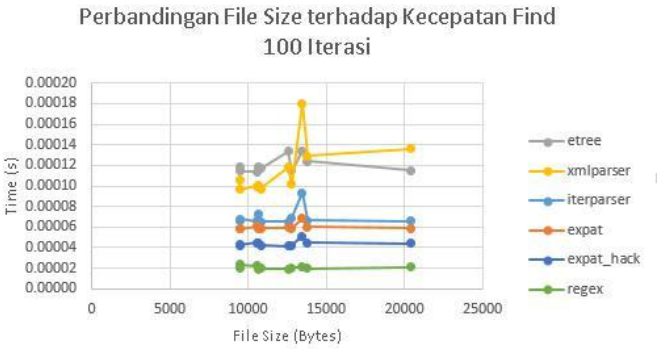
Feature	ATL	GT	TXL	4DML	XSLT	GAC	UIML	RDL	YATE	DIA + Script
Declarative	+	+	+	+	+	-	+	-	-	+
Imperative	+	-	-	-	+	+	-	+	+	+
Model Transformation	+	+	(+)	(+)	(+)	(+)	(+)	(+)	(+)	+
XML Transformation	-	-	(+)	(+)	+	+	-	+	+	(+) contribution
Code Transformation	-	-	+	(+)	-	-	-	-	-	+
Complex Mapping	+	+	+	+	+	+	-	+	+	-
Extensible	+	-	-	-	-	-	-	+	+	+
Parameterizable	-	-	-	-	-	+	-	+	+	extend, not yet

TABEL 4. PERBANDINGAN KOMPLEKSITAS FILE XML

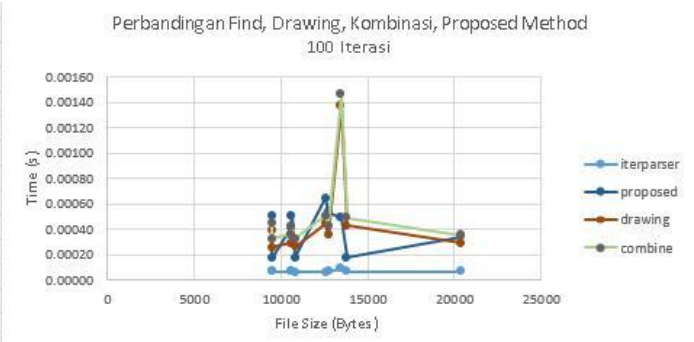
File XML	Tot. Tag	Var. Tag	Tot. Attrib.	Node Depth	File Size (B)
survey_form	144	10	276	15	20395
Surveys	106	9	183	10	13805
sale.order.tree	115	9	285	11	13441
survey_question_tree	98	9	167	8	12763
Survey-Pages	96	8	161	8	12578
Bill-of-Materials-Struc	99	10	263	6	10804
Meeting-Categories	107	10	248	9	10628
mrp.repair.form	119	9	217	9	10556
res.alarm.tree	91	10	220	9	9474
account.voucher.payr	79	8	205	6	9471

TABEL 5. PERBANDINGAN *TIMING PERFORMANCE* XML

File XML	regex	etree	xmlparser	iterparser	expat	e_hack
survey_form	0.000021	0.000115	0.000136	0.000066	0.000059	0.000044
Surveys	0.000020	0.000124	0.000129	0.000066	0.000060	0.000045
sale.order.tree	0.000021	0.000134	0.000180	0.000093	0.000068	0.000051
survey_question_tree	0.000021	0.000114	0.000102	0.000069	0.000059	0.000042
Survey-Pages	0.000019	0.000133	0.000118	0.000065	0.000059	0.000042
Bill-of-Materials-Struct	0.000020	0.000117	0.000097	0.000065	0.000059	0.000042
Meeting-Categories	0.000020	0.000119	0.000100	0.000072	0.000059	0.000044
mrp.repair.form	0.000023	0.000114	0.000099	0.000066	0.000061	0.000045
res.alarm.tree	0.000024	0.000114	0.000097	0.000068	0.000058	0.000042
account.voucher.paym	0.000020	0.000118	0.000106	0.000067	0.000058	0.000043



Gambar 6. Perbandingan *file size* terhadap kecepatan *find*



Gambar 7. Perbandingan *parsing* dan *render* manual dan transformasi

DAFTAR PUSTAKA

[1] S. Tiny, *Open Object Developer Book*. OpenERP, 2009.

[2] S. Kadry and J. Claver, "XML parser GUI using .NET Technology," *Int. Conf. Future Comput. Support. Educ.*, vol. 2, pp. 554–560, August 22-23.

[3] WSEAS International Conference on Telecommunications and Informatics, V. Niola, and WSEAS (Organization), Eds., *New aspects of telecommunications and informatics: 9th WSEAS International Conference on Telecommunications and Informatics (TELE-INFO '10) : Catania, Italy, May 29-31, 2010*. Sofia, Bulgaria: EUROPMENT Press, 2010.

[4] P. Chmelar, R. Hernych, and D. Kubicek, "Interactive Visualization of Data-Oriented XML Documents," in *Advances in Computer and Information Sciences and Engineering*, T. Sobh, Ed. Springer Netherlands, 2008, pp. 390–393.

[5] M. A. Rothenberger and M. Srite, "An Investigation of Customization in ERP System Implementations," *IEEE Trans. Eng. Manag.*, vol. 56, no. 4, pp. 663–676, Nov. 2009.

[6] E. Pietriga, J.-Y. Vion-Dury, and V. Quint, "VXT: a visual approach to XML transformations," in *Proceedings of the 2001 ACM Symposium on Document engineering*, 2001, pp. 1–10.

[7] S. Holzner, *Inside XML*. Indianapolis, Ind.: New Riders, 2001.

[8] M. C. Kei, "The Structured-Element Object Model for XML," The Chinese University of Hong Kong, 2002.

[9] M. Bačíková and J. Porubän, "Analyzing stereotypes of creating graphical user interfaces," *Cent. Eur. J. Comput. Sci.*, vol. 2, no. 3, pp. 300–315, 2012.

[10] M. Bačíková and J. Porubän, "UI Creation Patterns (using iTasks for DSL→ GUI transformation)," in *Informatics'2013: proceedings of the 13'th International Conference on Informatics*, 2013, pp. 5–7.

[11] U. Brandes and C. Pich, "GraphML Transformation," in *Proceedings of the 12th International Conference on Graph Drawing*, Berlin, Heidelberg, 2004, pp. 89–99.

[12] J. Shipman, "Python XML processing with lxml." New Mexico Tech Computer Center, 24-Aug-2013.

[13] L. Chen and G. Banflavi, "Methodologies and Architecturefor the Implementation of a Web Application," 2012.

[14] J. M. G. Calleros, A. Stanciulescu, J. Vanderdonckt, J.-P. Delacre, and M. Winckler, "A Comparative Analysis of Transformation En-gines for User Interface Development."

[15] R. Schaefer, "A survey on transformation tools for model based user interface development," in *Human-Computer Interaction. Interaction Design and Usability*, Springer, 2007, pp. 1178–1187.

[16] J.-L. Pérez-Medina, S. Dupuy-Chessa, and others, "A survey of model driven engineering tools for user interface design," in *Task Models and Diagrams for User Interface Design*, Springer, 2007, pp. 84–97.

[17] A. Veres and Szentkirályi, "Extending Python Web Services," BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS, 2001.

[18] Q. Zhang, "EMBEDDING PARALLEL BIT STREAM TECHNOLOGY INTO EXPAT," Vancouver Island University, 2007.

[19] M. Oikawa, R. Ierusalimschy, and A. Moura, "Converting regexes to Parsing Expression Grammars," in *Proceedings of the 14th Brazilian Symposium on Programming Languages, SBLP*, 2010, vol. 10.

[20] *Python XML Parsing Benchmark*. .

[21] C. Fydo Hopp, *Image Merge - For Creating Spritesheets - v0.3*. 2006.