LAPORAN AKHIR
PENELITIAN MANDIRI

Redesigning CHIML:

Orchestration Language for Chimera-Framework

Peneliti
Go Frendi Gunawan, M.Kom.
Jozua Ferjanus Palandi, M.Kom.
Subari, M.Kom.

SEKOLAH TINGGI INFORMATIKA & KOMPUTER INDONESIA
Januari 2019

HALAMAN PENGESAHAN

PENELITIAN MANDIRI

Judul Penelitian : **Redesigning CHIML: Orchestration Language for Chimera-Framework**

Peneliti:

m. Nama Lengkap : Go Frendi Gunawan
n. NIP/NIDN : 0728108701
o. Jabatan Fungsional : Asisten Ahli
p. Program Studi : Teknik Informatika
q. Nomor HP : 0896-8086-8343
r. Alamat surel (e-mail) : frendi@stiki.ac.id

Anggota Peneliti (1)

e. Nama Lengkap : Jozua Ferjanus Palandi
f. NIP/NIDN : 0012057201

Anggota Peneliti (2)

e. Nama Lengkap : Subari
f. NIP/NIDN : 0702027201

Mahasiswa yang terlibat : - orang

NO	Nama	NRP	Program Studi
1	-		
2	-		
3	-		

Biaya Penelitian : -

Malang, 25 Januari 2019

Mengetahui,
Kepala Program Studi

(Rakhmad Maulidi, M.Kom.)
0706018203

Ketua Peneliti,

(Go Frendi Gunawan)
0728108701

Menyetujui
Kepala LPPM,

(Subari, S.Kom., M.Kom.)
0702027201

DAFTAR ISI

Halaman

DAFTAR TABEL

No table of figures entries found.

DAFTAR GAMBAR

No table of figures entries found.

DAFTAR LAMPIRAN

RINGKASAN

Component-Based Software Engineering (CBSE) telah terbukti cukup efektif untuk menangani kompleksitas perangkat lunak. Saat ini pengembang lebih suka membangun micro-services daripada aplikasi monolitik tunggal. Beberapa pendekatan SOA (Arsitektur Berorientasi Layanan) seperti HTTP / REST API, CORBA, dan BPEL umumnya digunakan oleh pengembang. Beberapa dari solusi tersebut dibangun dengan asumsi bahwa pengembang dapat membangun layanan dari awal atau dapat membuat lapisan abstraksi untuk layanan yang sudah ada sebelumnya. Dalam kebanyakan kasus, asumsi itu benar. Namun, ada beberapa kasus ketika pengembang lebih memilih untuk menjaga arsitektur sesederhana mungkin tanpa perlu membangun lapisan abstraksi tambahan. Misalnya, ketika mereka bekerja dengan mini-embedded system. Sebelumnya, bahasa orkestrasi berbasis YAML dikembangkan untuk Chimera-Framework (Kerangka agnostik bahasa untuk komputasi yang berdiri sendiri dan terdistribusi). Namun, kami mengamati beberapa titik lemah dalam bahasa tersebut. Dalam tulisan ini, kami mempersempit bahasa orkestrasi untuk memungkinkan pengembang mengakses layanan yang sudah ada sebelumnya tanpa perlu membangun lapisan abstraksi lain.

PRAKATA

Dengan segala kerendahan hati, peneliti memanjatkan puji dan syukur kehadirat Tuhan atas selesainya laporan Hasil Penelitian dengan judul "**Redesigning CHIML: Orchestration Language for Chimera-Framework".** Penulisan laporan hasil penelitian ini dimaksudkan untuk memenuhi salah satu tri darma perguruan tinggi yaitu penelitian dosen. Penulis menyadari, selesainya penyusunan Laporan Hasil Penelitian ini tidak terlepas dari bantuan berbagai pihak.

Untuk itu penulis mengucapkan Terimakasih kepada:

1. Ketua STIKI
2. Wakil Ketua bidang Akademik
3. Kepala Program Studi Teknik Informatika STIKI Malang
4. Kepala LPPM STIKI Malang

Semoga Laporan Hasil Penelitian ini dapat dimanfaatkan dan dapat memberikan sumbangsih pemikiran untuk perkembangan pengetahuan bagi penulis maupun bagi pihak lain yang berkepentingan.

Terima kasih.

BAB 1
PENDAHULUAN

Pengembangan perangkat lunak adalah topik yang sangat menarik. Cara orang mengembangkan perangkat lunak berubah ketika paradigma baru muncul. Pada gilirannya, pengembangan perangkat lunak juga memengaruhi budaya. Ini mengubah cara orang berinteraksi satu sama lain serta bagaimana mereka berinteraksi dengan komputer.Seiring dengan semakin kompleksnya perangkat lunak, membangun dan memelihara perangkat lunak juga menjadi lebih sulit. Berbagai pendekatan telah dicoba untuk membuat proses lebih mudah.Beberapa kerangka kerja seperti Rails [1], Django [2], Laravel [3], dll berfokus pada cara membuat kode bersih, dipisahkan, dan dapat digunakan kembali. Pada gilirannya, ini membuat pengembangan dan pemeliharaan perangkat lunak lebih mudah. Kerangka kerja ini dengan cepat mendapatkan popularitas di kalangan pengembang perangkat lunak.Namun, meskipun keuntungan jelas dari kerangka kerja itu, mereka juga memiliki beberapa kelemahan. Karena kerangka kerja dibangun di atas bahasa pemrograman tertentu, integrasi ke berbagai bahasa pemrograman cukup menantang. Misalnya, untuk membangun perangkat lunak dengan menggunakan Rails, pengembang harus membuat kode di Ruby. Hal yang sama juga berlaku untuk Django, yang dibangun di atas Python, dan Laravel yang dibangun di atas PHP. Situasi ini dikenal sebagai vendor-lock-in [4]. Dan dalam beberapa kasus, ini dapat memengaruhi kemampuan pemeliharaan serta pengembangan lebih lanjut dengan cara yang buruk.Dengan demikian, pendekatan agnostik yang lebih teknologi diperlukan untuk mengatasi masalah vendor-lock-in. Baru-baru ini, SOA dan layanan mikro mulai populer. Pendekatan-pendekatan itu membuat pengembang fokus pada komponen kecil yang dikenal sebagai layanan daripada perangkat lunak monolitik yang kompleks. Suatu layanan biasanya mewakili sumber daya tunggal atau proses bisnis. Layanan ini dapat saling tidak tergantung satu sama lain atau saling menyadari. Terlepas dari keunggulan dan popularitas, SOA dan layanan mikro cenderung mengalami masalah konkurensi [5].Salah satu aspek penting dalam SOA / layanan mikro adalah bagaimana pengembang dapat menyusun layanan independen untuk bekerja bersama. Ada dua pendekatan umum untuk

menyusun layanan. Proses menyusun beberapa layanan yang independen satu sama lain bernama orkestrasi, sedangkan proses untuk menyusun beberapa layanan yang saling menyadari disebut koreografi. Orkestrasi membutuhkan satu pengontrol pusat untuk mengelola layanan [6].Orkestrasi memiliki sejarah yang sangat panjang. Implementasi orkestrasi yang paling awal adalah mekanisme Unix Pipe [7]. Mekanisme ini masih relevan dan digunakan oleh pengguna Unix / Linux.Mekanisme Unix Pipe bukan satu-satunya implementasi orkestrasi perangkat lunak. Remote Procedure Call (RPC) juga digunakan untuk proyek yang lebih besar. RPC yang umum digunakan adalah JSONRPC [8] dan XML-RPC [9]. Pada 2015, Google juga memperkenalkan GRPC [10].Selain Unix Pipe dan RPC, beberapa implementasi lainnya diperkenalkan oleh perusahaan dan konsorsium independen. OMG memperkenalkan CORBA pada tahun 1991, sementara OASIS diperkenalkanBPEL pada tahun 2001. Beberapa pengembang juga menerapkannya sendiriImplementasi HTTP / REST API. Pada 2016, Feilhauer dan Sobotka menciptakan kerangka kerja bernama DEF yang berfokus pada eksekusi paralel [11]. Pada 2017, kami juga melakukan penelitian untuk mengembangkan kerangka agnostik bahasa lain bernama Chimera-Framework [12].Dibandingkan dengan kerangka kerja monolitik seperti Rails dan Django, mekanisme orkestrasi ini lebih skalabel dan agnostik teknologi. Ini berarti bahwa pengembang dapat memilih tumpukan teknologi terbaik untuk membangun komponen / layanan mereka.Di antara mekanisme orkestrasi tersebut, Unix pipe dan Chimera-Framework adalah satu-satunya yang mendukung tetapi tidak terbatas pada protokol jaringan. Biasanya, Arsitektur Berorientasi Layanan (seperti CORBA, BPEL, atau DEF) berkorelasi dengan arsitektur terdistribusi, tetapi ini tidak selalu terjadi. Misalnya, pengembang mungkin hanya perlu meneruskan data dari R ke Python di mesin yang sama.Dibandingkan dengan mekanisme yang ada, Chimera Framework fokus pada bagaimana membiarkan upaya pengembang seminimal mungkin. Komponen orkestrasi di Chimera-Framework tidak harus sadar HTTP. Bahkan utilitas UNIX lama seperti date atau cat dapat berfungsi sebagai komponen [12]. Tetapi dalam kasus arsitektur terdistribusi diperlukan, komponen yang sama masih akan dapat digunakan.Dalam penelitian ini, kami fokus pada peningkatan mekanisme orkestrasi di Chimera-Framework. Bahasa orkestrasi bernama CHIML (Chimera

Markup Language) yang merupakan superset dari YAML [13].CHIML dirancang agar mudah dibaca, ringkas, dan intuitif, namun memiliki beberapa kelemahan:

- CHIML lambat karena kerangka kerja memanggil vm.runInNewContext untuk setiap pernyataan.
- Dukungan CHIML untuk komputasi paralel terbatas, terutama jika paralelisasi berorientasi data.
- CHIML menggunakannya protokol HTTP sendiri untuk komunikasi jaringan, yang tidak biasa.Dalam penelitian ini, kita akan menyelesaikan dua masalah ini dengan mendesain ulang CHIML.

BAB 2
TINJAUAN PUSTAKA

A. Orkestrasi dan Koreografi

Pada tahun 1975, Frank DeRemer dan Hans Kron menulis makalah tentang programming-in-large and programming-in-small. Pemrograman-dalam-besar adalah konsep untuk menulis perangkat lunak yang terdiri dari berbagai modul yang mungkin ditulis oleh orang yang berbeda [14]. Konsep pemrograman-dalam-besar sangat erat digabungkan dengan orkestrasi dan koreografi. Dalam orkestrasi, komponen dikendalikan oleh pengontrol tunggal, sedangkan dalam koreografi, komponen saling memperhatikan. Dalam beberapa kasus, orkestrasi dan koreografi dapat digunakan bersama. Pengembang mungkin membuat koreografi komponen di mana komponennya adalah orkestrasi dari komponen yang lebih kecil [6].

B. SOA dan Layanan Mikro

SOA dan layanan mikro adalah konsep arsitektur yang serupa. Filosofi inti dari kedua paradigma ini adalah untuk membagi masalah besar menjadi masalah kecil. Istilah SOA biasanya digunakan ketika orang berbicara dalam ruang lingkup yang lebih besar, sementara layanan mikro biasanya disebut dengan ruang lingkup yang lebih kecil. Ketika layanan disusun dan digunakan secara internal itu adalah layanan mikro. Tetapi ketika layanan terkena sistem eksternal, itu adalah SOA. Meskipun perbedaannya tidak jelas, orang biasanya setuju bahwa SOA mungkin mengandung layanan mikro [5].Dari sudut pandang teknis, SOA dan layanan mikro mungkin melibatkan orkestrasi atau koreografi. Dan tidak peduli yang mana yang digunakan, diperlukan protokol untuk menyampaikan pesan di antara layanan.

C. HTTP / REST API

HTTP API adalah protokol yang cukup umum. Arsitekturnya juga relatif sederhana. Ada layanan web dan klien. Klien mengirim permintaan ke layanan web dan layanan web membalas dengan tanggapan. Data respons biasanya dalam format JSON / XML. Komponen SOA biasanya

mengekspos titik akhir HTTP API. OMDB misalnya, menyediakan API untuk mencari basis data film. Di sisi lain, REST (Representation State Transfer) adalah implementasi HTTP API yang lebih ketat. Ini diperkenalkan dan didefinisikan pada tahun 2000 oleh Roy Fielding dalam tesis doktoralnya [15]. Menempatkan fokus pada bagaimana URI harus mewakili objek, sedangkan kata kerja HTTP berfungsi sebagai metode itu. Jika titik akhir sepenuhnya mengadaptasi spesifikasi REST, itu disebut layanan web RESTful.

D. JSON-RPC, XML-RPC, dan GRPC

RPC adalah singkatan dari Remote Procedure Call. Fokus RPC adalah membiarkan prosedur yang dibuat pengembang di komputer jarak jauh semudah prosedur lokal. JSON-RPC dan XML-RPC bekerja dengan cara yang sama, hanya format pertukaran data yang berbeda. Seperti namanya, JSON-RPC menggunakan JSON [8], sedangkan XML-RPC menggunakan XML [9]. Pertama, klien mengirim permintaan ke server. Permintaan berisi nama metode dan parameter. Server kemudian membalas dengan nilai pengembalian.Karena XML lebih verbose dari JSON, XML-RPC juga membutuhkan lebih banyak bandwidth dibandingkan dengan JSON-RPC. Dan karena XML dan JSON adalah format teks, data biner juga harus dikodekan dalam format teks. Misalnya, gambar mungkin perlu dikodekan dalam format basis-64.Untuk mengatasi kebutuhan untuk menyandikan data ke dalam format teks,Google membuat protokol lain bernama GRPC (Google Remote Procedure Call). GRPC mendukung beberapa bahasa pemrograman termasuk Python, Javascript, Java, dan PHP. Untuk menggunakan GRPC [10], pengembang harus membuat kerangka rintisan (mirip dengan IDL di CORBA, BPEL, atau EJB).

BAB 3
TUJUAN DAN MANFAAT PENELITIAN

Tujuan dari penelitian ini adalah :

- Merancang framework dan meningkatkan kecepatan eksekusi CHIML

Manfaat yang didapat dari penelitian ini adalah :

- Membuat CHIML mendukung paralelisasi berorientasi data
- Menerapkan protokol yang lebih umum untuk komunikasi jaringan.
- Membuat CHIML menjadi mudah dibaca, ringkas, dan intuitif.

BAB 4
METODE PENELITIAN

Penelitian ini akan dilakukan dengan metodologi sebagai berikut :

A. Improving Speed

Setelah melakukan beberapa penyelidikan, kami menemukan bahwa ada dua alasan mengapa CHIML lambat:

- CHIML memanggil vm.runInNewContext untuk setiap pernyataan.
- Node.Js perlu memeriksa semua direktori yang sesuai untuk menyelesaikan jalur modul yang diimpor.

Untuk menyelesaikan masalah pertama, kami menulis ulang interpreter CHIML dan membangun transpiler sebagai gantinya. Jadi, alih-alih menggunakan vm.runInNewContext untuk setiap pernyataan, kami menerjemahkan seluruh skrip CHIML ke dalam Javascript yang valid. Jadi, kita hanya perlu memanggil vm.runInNewContext sekali. Pendekatan ini mari kita jelajahi kemungkinan lain seperti benar-benar membangun Javascript runnable yang secara teoritis runnable dan dipisahkan dari Chimera-Framework. Adapun masalah kedua, banyak pengembang telah mencoba untuk mengatasi masalah dengan menimpa Node.Js asli guna mempercepat proses penyelesaian modul.

B. Supporting Data-Oriented Parallelization

Pada versi sebelumnya, CHIML sudah mendukung paralelisasi dengan menggunakan kata kunci paralel. Namun, mekanisme ini bekerja dengan asumsi bahwa pengembang sudah tahu berapa banyak proses paralel yang dibutuhkan. Sejak beberapa waktu, paralelisasi didasarkan pada input pengguna / data yang diambil, mekanisme ini menjadi tidak dapat digunakan dalam kasus tersebut. Untuk mengatasi masalah ini, kami memutuskan untuk mendukung peta, memfilter, dan mengurangi. Konsep-konsep ini cukup umum dalam pemrograman fungsional, dan sangat berguna untuk mengubah array. Di belakang layar, kami menerjemahkan peta, memfilter, dan mengurangi menjadi Promise.all () untuk membuatnya benar-benar sejajar.

C. More Common Network Protocol

Dalam beberapa tahun terakhir, banyak protokol baru seperti GraphQL muncul. Beberapa berpendapat bahwa GraphQL akan menjadi protokol standar baru menggantikan REST-API. Namun, hal ini tidaklah benar. GraphQL hanyalah lapisan pembungkus lain yang merangkum beberapa titik akhir REST. Lebih jauh, kami pikir REST pada dasarnya adalah titik akhir tunggal yang merangkum beberapa RPC menjadi satu sumber daya tunggal. Dengan demikian, kita dapat mengatakan bahwa blok bangunan komunikasi berbasis jaringan adalah RPC.vKarena CHIML didasarkan pada JavaScript, dan JavaScript dekat dengan JSON, kami memutuskan untuk menggunakan JSON-RPC untuk bertukar data melalui jaringan. Selain JSON-RPC, pure HTTPRequest juga didukung. Dengan menggunakan protokol-protokol itu, kami percaya Chimera akan lebih berguna karena dapat berbicara dengan berbagai sistem dengan mulus.

D. Readable Language

Ada beberapa metrik untuk mengukur keterbacaan. Pada tahun 1975, Kincaid memperkenalkan formula untuk mengukur keterbacaan teks [16]. Meskipun rumus ini biasa digunakan untuk mengukur keterbacaan teks, beberapa orang berpendapat bahwa bahasa pemrograman harus memiliki berbagai jenis metrik.Pada 2010, Buse dan timnya memperkenalkan metrik baru untuk mengukur keterbacaan bahasa pemrograman [17]. Metrik pada dasarnya adalah versi modifikasi dari formula Kincaid.Kami mencoba menggunakan metrik ini untuk membandingkan CHIML dengan Python dan Javascript.

E. Semantic Design

Tinjauan umum semantik CHIML disajikan pada Listing 1 sebagai antarmuka skrip.

Listing 1. CHIML Semantic Design // Structure of a 'command' interface CommandObject{

```
  ins : string | string [ ] , out : string , vars : {[ key : string ] : any } , if : strin
  g = " true " ,

  map : string , filter : string , reduce : string , into : string , accumulator :
  string ,

  do : command | command [ ] , parallel : command [ ] ,

   while : string ="false",
};
```

Untuk mendukung keduanya, pemrograman-dalam-besar, dan pemrograman-dalam-kecil, kami menempatkan kontrol cabang dan loop di semantik (jika dan sementara). Peta, filter, dan pengurangan saling eksklusif. Pengembang hanya dapat menggunakan salah satunya, bersamaan dengan kunci. Akumulator hanya digunakan untuk mengurangi operasi.Karena kami fokus pada bagaimana membuat bahasa lebih kompak, kami juga mendukung beberapa pekerjaan pendek sehingga satu proses dapat ditulis dalam satu baris.Misalnya, proses pada Listing 2 juga dapat ditulis seperti pada Listing 3Selain itu, karena pengembang mungkin menggunakan untuk menulis tugas sebelah kiri dalam bahasa pemrograman utama (yaitu: hasil = fungsi (param1, param2)), kami pikir panah terbalik di CHIML dapat membuatnya lebih intuitif. Dengan demikian, Listing 3 juga dapat ditulis seperti Listing 4.

Listing 2. CHIML Program Example ins : month year
out : calendar do : cal

Listing 3. CHIML Program Example (Short)
(month , year) –> cal –> calendar

Listing 4. CHIML Program Example (Short-Reversed) calendar <– cal <– (month , year)

Sifat CHIML yang dapat digunakan untuk pemrograman in-large dan programming-in-small memungkinkan pengembang untuk dengan cepat membuat prototipe. Misalnya, untuk mendapatkan konten halaman-1 ke halaman-10 dari sebuah situs web, pengembang dapat melakukan iterasi seperti yang ditampilkan pada Listing 5.

Listing 5. CHIML Iteration ins : url out : html do :
```
– i <–– 1
– while : i < 11 do :
        – ( url + '? page =' + i ) –> curl –> htmlFragment
      – ( html + htmlFragment ) ––> html
      – i <–– ( i +1)
```

Kami juga mengambil analisis lebih lanjut dan menyimpulkan bahwa untuk kasus serupa, lebih baik melakukan tugas secara paralel, sehingga permintaan pertama tidak memblokir permintaan selanjutnya. Ini dapat dicapai dengan sempurna dengan menggunakan mekanisme peta seperti yang ditunjukkan pada Listing 6:

Listing 6. CHIML Map Feature ins : url out : html vars : indexes : [1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10]
do :
```
– map : indexes into : responses ins : index do :
          – ( url + '? page =' + index ) –> curl –> response

  – | ( responses , ' ') –> {( array ) => array . join ("") }
      –> html
```

Peta lebih efisien karena di bawah hood, setiap iterasi akan dilakukan secara non-blocking.

BAB 5
HASIL DAN LUARAN YANG DICAPAI

Untuk setiap masalah, kami membangun 3 solusi. Solusinya menggunakan protokol HTTP API dan ditulis dalam tiga bahasa pemrograman (Python, JavaScript, dan CHIML). Beberapa teknologi seperti CORBA, BPEL, dan EJB tidak membantu menyelesaikan masalah kami karena teknologi tersebut memerlukan IDL, pengaturan broker, dan prasyarat lainnya. Seperti HTTP API, SOAP dan RPC hanyalah protokol. Dengan data encoder / decoder yang benar dan titik akhir yang sesuai, kami juga dapat membangun solusi yang sama untuk teknologi ini.Untuk membuat kode lebih pendek, kami menggunakan fungsi peta dan panah untuk JavaScript, dan menggunakan ekspresi lambda untuk Python.Solusi dari masalah g disajikan pada Listing 8, 9, dan 10:

Listing 8. Solusi JavaScript untuk masalah-g:

```
const rpn = require ( ' request −promise−native ' ) async function fetchGenres ( ) {
    const body = await rpn ( ' http : / / l o c a l h o s t :3000/ genres ' )
    const genres = JSON . parse ( body ) . map ( ( genre ) => genre . name )
    console . log (JSON . s t r i n g i f y ( genres ) )
} fetchGenres ( )
```

Listing 9. Solusi Python untuk masalah-g:

TABLE I. COMPARISON

b real	b sys	b user	fkgl	fre	loc	rbr	size	solution/case
0.509	0.028	0.503	6.571	64.934	30	0.000	978	js, gba
0.114	0.011	0.096	8.492	57.214	15	0.002	662	py, gba
0.827	0.054	0.829	5.795	73.210	17	0.018	576	chie, gba
0.799	0.054	0.798	7.606	64.371	334	0.000	21955	chic, gba
0.486	0.034	0.477	6.895	63.440	16	0.007	540	js, gb
0.104	0.009	0.092	7.871	59.853	12	0.006	471	py, gb
0.802	0.057	0.798	5.219	76.690	13	0.081	396	chie, gb
0.801	0.049	0.806	7.639	64.218	254	0.000	15048	chic, gb
0.476	0.028	0.474	7.551	59.499	7	0.307	250	js, g
0.096	0.014	0.081	7.422	60.549	6	0.160	217	py, g
0.798	0.051	0.807	5.187	74.796	6	0.147	167	chie, g
0.768	0.053	0.770	7.715	63.576	133	0.000	6498	chic, g

```
response = request . urlopen ( ' http : / / l o c a l h o s t :3000/ genres ' ) . read ( )
genres = json . loads ( response ) r e s u l t = l i s t (map( lambda genre : genre [ 'name ' ] , genres ) )
print ( json . dumps ( r e s u l t ) )
```

Listing 10. Solusi CHIML untuk masalah-g:

```
– ( ' http : / / l o c a l h o s t :3000/ genres ' ) –> [ sys .
        httpRequest ] –> genres
– map : genres into : genres
    do :          ( genre ) –> ( x ) => x . name –> genre
```

BAB 6
KESIMPULAN DAN SARAN

CHIML berfungsi dengan baik sebagai bahasa orkestrasi. Namun, untuk mengendalikan struktur, keberadaan komponen perantara dapat membantu meningkatkan kinerja. Ciri terbaik CHIML adalah dukungannya untuk pemrograman-dalam-besar dan pemrograman-dalam-kecil. Meskipun struktur kontrolnya masih menderita untuk kecepatan dan kinerja, ia berfungsi dengan baik sebagai alat prototyping. Dalam hal fleksibilitas, CHIML dapat dibandingkan dengan Python atau JavaScript. Dalam kebanyakan kasus, bahkan memiliki ukuran dan LOC terkecil dari keduanya, Python dan JavaScript. Ini berarti bahwa pengembang dapat memulai solusi orkestrasi di CHIML, dan secara bertahap melakukan optimasi sesuai kebutuhan.Terlepas dari keuntungannya, interpreter CHIML masih harus di refactored dan dioptimalkan agar dapat digunakan untuk kasus-kasus penggunaan dunia nyata.

DAFTAR PUSTAKA

[1] D. H. Hansson, "Ruby on rails — a web-application framework that includes everything needed to create database-backed web applications according to the model-view-controller (mvc) pattern." https://rubyonrails.org, accessed: 2018-04-30.

[2] D. S. Foundation, "The web framework for perfectionists with deadlines — django," https://djangoproject.org, accessed: 2018-04-30.

[3] T. Otwell, "Laravel - the php framework for web artisans," https://laravel.com, accessed: 2018-04-30.

[4] B. K. Jr., "What is software vendor lock-in? (and how to avoid it)," https://www.makeuseof.com/tag/software-vendor-lock-avoid, accessed: 2018-04-30.

[5] M. Little, "Soa versus microservices?" https://www.infoq.com/news/2015/12/soa-v-microservices, accessed: 2018-04-30.

[6] J. Spacey, "Orchestration vs choreography," https://simplicable.com/new/orchestration-vs-choreography, accessed: 2018-04-30.

[7] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell, "Mass-produced software components," in *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, 1968, pp. 88–98.

[8] Trac, "Json-rpc," http://json-rpc.org, accessed: 2018-04-30.

[9] I. UserLand Software, "Xml-rpc.com," http://xmlrpc.scripting.com, accessed: 2018-04-30.

[10] I. Google, "grpc.io," https://grpc.io, accessed: 2018-04-30.

[11] T. Feilhauer and M. Sobotka, "Def-a programming language agnostic framework and execution environment for the parallel execution of library routines," *Journal of Cloud Computing*, vol. 5, no. 1, p. 20, 2016.

[12] G. F. Gunawan, J. F. Palandi, and M. Amien, "Chimera - simple language agnostic framework for stand alone and distributed computing," in *Computer Applications and Information Processing Technology (CAIPT), 2017 4th International Conference on*. IEEE, 2017, pp. 144– 153.

[13] C. C. Evans, "The official yaml website," http://yaml.org/, accessed: 2018-04-30.

[14] F. DeRemer and H. Kron, "Programming-in-the large versus programming-in-the-small," *SIGPLAN Not.*, vol. 10, no. 6, pp. 114–121, Apr. 1975. [Online]. Available: http://doi.acm.org/10.1145/390016.808431

[15] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," in *Proceedings of the 22Nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 407–416. [Online]. Available: http://doi.acm.org/10.1145/337180.337228

[16] F. R. R. R. C. B. Kincaid, J.P., "Derivation of new readability formulas (automated readability index, fog count, and flesch reading ease formula) for navy enlisted personnel," *Research Branch Report 875*, 1975.

[17] R. P. L. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, July 2010.

LAMPIRAN-LAMPIRAN

Lampiran 1. Instrumen Penelitian

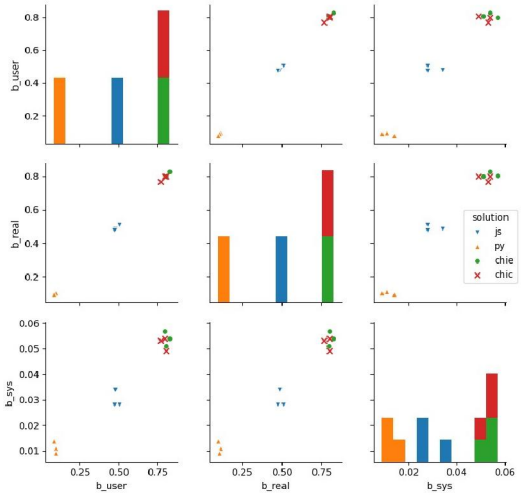

Fig. 1. Performance Comparison

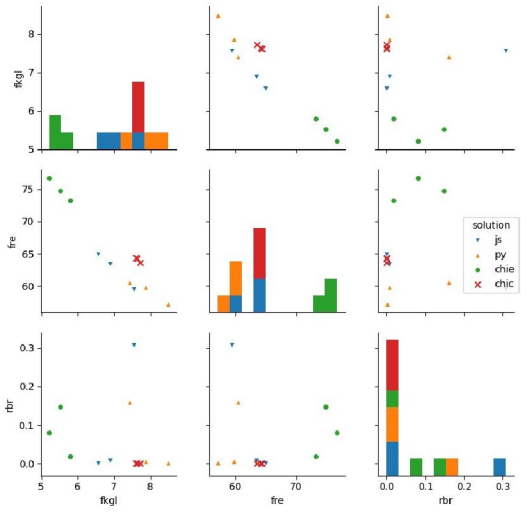

Fig. 2. Readability Comparison

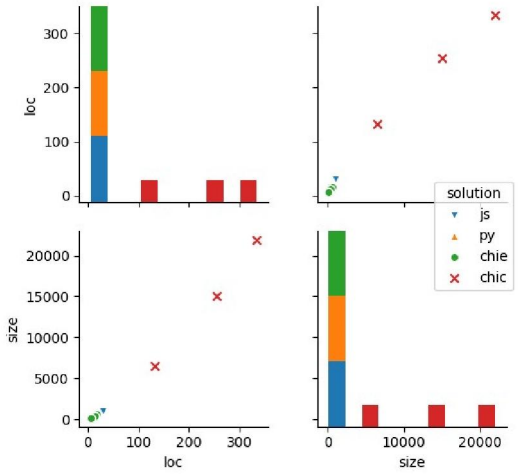

Fig. 3. Size Comparison

Lampiran 2. Personalia Tenaga Pelaksana Beserta Kualifikasinya

No	Nama/NIDN	Instansi Asal	Bidang Ilmu	Alokasi Waktu (Jam/ Minggu)	Uraian Tugas
1	Go Frendi Gunawan, M.Kom. /0728108701	STIKI MALANG	Desktop Programming	10	▪ Mengkoordinasi penyusunan aplikasi ▪ Mengkoordinasi persiapan instrumen penelitian dan perlengkapan penunjang lainnya ▪ Mengkoordinasi penyusunan laporan penelitian dan artikel ilmiah seminar ▪ Penanggungjawab
2	Jozua Ferjanus Palandi, M.Kom., /0012057201	STIKI MALANG	Teknik Informatika	10	▪ Membantu ketua dalam menyusun aplikasi ▪ Membantu ketua menyiapkan instrumen penelitian dan perlengkapan penunjang lainnya ▪ Membantu ketua menyusun laporan penelitian dan artikel ilmiah
3	Subari, M.Kom. /0702027201	STIKI MALANG	Mobile Programming	10	▪ Membantu ketua dalam menyusun aplikasi ▪ Membantu ketua menyiapkan instrumen penelitian dan perlengkapan penunjang lainnya ▪ Membantu ketua menyusun laporan penelitian dan anggaran (RAB)

Lampiran 3. Artikel Ilmiah

Redesigning CHIML: Orchestration Language for Chimera-Framework

Go Frendi Gunawan
STIKI Malang
Malang, Indonesia
Email: frendi@stiki.ac.id

Jozua Ferjanus Palandi
STIKI Malang
Malang, Indonesia
Email: jozuafp@stiki.ac.id

Subari
STIKI Malang
Malang, Indonesia
Email: subari@stiki.ac.id

***Abstract*—Component Based Software Engineering (CBSE) has been proven to be quite effective to deal with software complexity. Nowadays developers prefer to build micro-services rather than single monolithic application. Several SOA (Service Oriented Architecture) approaches like HTTP/REST API, CORBA, and BPEL are commonly used by developers. Some of those solutions are built under assumptions that the developers are either building the services from scratch or able to create abstraction layer for the pre-existing services. In most cases the assumptions are true. However there are cases when developers prefer to keep the architecture as simple as possible without any need to build additional abstraction layers. For example, when they work with mini-embeded system.**

Previously, a YAML based orchestration language was developed for Chimera-Framework (A language agnostic framework for stand-alone and distributed computing). In this paper, we refine the orchestration language in order to let developers accessing pre-existing services without any need to build another abstraction layer.

***Keywords*—Chimera, CBSE, Orchestration Language**

I. Introduction

Software development is a very interesting topic. The way people developing softwares is changing as new paradigms emerged. In turns, software development also affecting the culture. It change how people interact to each others as well as how they interact with computers.

As software become more and more complex, building and maintaining softwares is also become harder. Various approaches have been attempted in order to make the processes easier.

Several frameworks like Rails [1], Django [2], Laravel [3], etc are focusing on how to make codes clean, separated, and reusable. In turn, this make software development and maintenance easier. These frameworks quickly gained popularity among software developers.

However, despite on the clear advantage of those frameworks, they also have several disadvantages. Since the frameworks were built on top of specific programming language, integration to different programming languages is quite challenging. For example, in order to build software by using Rails, developers should code in Ruby. The same is also true for Django, which is built on top of Python, and Laravel which is built on top of PHP. This situation is known as vendor-lock-in [4]. And in some cases it can affect maintainability as well as further development in a bad way.

Thus, a more technology agnostic approach is needed in order to overcome vendor-lock-in problem. Recently, SOA and micro-services are gaining popularity. Those approaches let developers to focus on small components known as services rather than complex monolithic software. A service usually represent single resource or business process. The services can be independent to each other or aware to each others. Despite of the advantages and popularity, SOA and micro-services are prone to concurrency problem [5].

One important aspect in SOA/micro-services is how a developer can compose independent services to work together. There are two common approach to compose services. The process to compose several services that are independent to each others is named orchestration, while the process to compose several services that aware to each others is named choreography. Orchestration require one central controller in order to manage the services [6].

Orchestration has a very long history. The earliest implementation of orchestration was Unix Pipe mechanism [7]. This mechanism is still relevant and used by Unix/Linux users.

Unix Pipe mechanism is not the only implementation of software orchestration. Remote Procedure Call (RPC) is also used for bigger projects. The commonly used RPCs are JSON-RPC [8] and XML-RPC [9]. On 2015, Google also introduce GRPC [10].

Beside Unix Pipe and RPC Several other implementations were introduced by independent companies and consortiums. OMG introduced CORBA on 1991, while OASIS intoduced BPEL on 2001. Some developers also implement their own HTTP/REST API implementation. On 2016, Feilhauer and Sobotka creating a framework named DEF which is focusing on parallel execution [11]. On 2017, we also conduct a research to develop another language agnostic framework named Chimera-Framework [12].

Compared to the monolithic frameworks like Rails and Django, these orchestration mechanism are more scalable and technology agnostic. This mean that the developers can choose the best technology stack to build their components/services.

Among those orchestration mechanisms, Unix pipe and Chimera-Framework are the only ones that are agnostic to the architecture and messaging protocol. To implement CORBA or BPEL, the developers has to specify IDL or WSDL. In DEF, the similar mechanism named DEF-WS also has to be created. These specifications also assuming that components

are HTTP aware. Consequently, the developers need to consider this behavior before creating a new service/component. It doesn't mean that pre-existing components cannot be used for orchestration. However the components are either need refactoring or additional broker. This make the deployment process more complex.

Chimera-Framework in the other hand focusing on how to make the effort minimal. The orchestration components in Chimera-Framework don't have to be HTTP aware. Even an old UNIX utility like *date* or *cat* can serve as components [12].

In this research we are focusing in improving the orchestration mechanism in Chimera-Framework. The orchestration language is named CHIML (Chimera Markup Language) which is a superset of YAML [13]. CHIML is designed to be readable, compact, and intuitive.

II. Research Question

In order to have a clear direction in our research, we are focusing in these two questions:

- How to make a readable, compact, and intuitive orchestration language for Chimera-Framework.
- How the orchestration language compared to other possible solutions.

III. Literature Survey

A. Orchestration and Choreography

In 1975, Frank DeRemer and Hans Kron wrote a paper about programming-in-large and programming-in-small. Programming-in-large is a concept to write a software that consists of various modules that are possibly written by different people [14].

The concept of programming-in-large is tightly coupled to orchestration and choreography. In orchestration, the components are controlled by a single controller, while in choreography, the components are aware of each others. In some cases, orchestration and choreography can be used together. Developer might create a choreography of components where the components are orchestration of smaller components [6].

B. SOA and Micro-service

SOA and micro-service are similar architecture concept. The core philosophy of these two paradigms are to split big problem into small problems. The term SOA is usually being used when people talk in a bigger scope, while micro-service is usually refer to smaller scope. When services are composed and used internally it is micro-service. But when the services exposed to external system, it is SOA. Although the distinction can be unclear, people usually agree that SOA might contains of micro-services [5].

From the technical point of view, SOA and micro-service might involve either orchestration or choreography. And no matter which one is being used, a protocol for passing message among services is required.

C. HTTP/REST API

HTTP API is a quite common protocol. The architecture is also relatively simple. There are web services and a clients. The client send a request to the web service, and the web service reply with a response. The response data is usually in JSON/XML format. SOA components are commonly exposing HTTP API endpoint. OMDB for example, provide an API to search for movie database.

On the other hand, REST (Representation State Transfer) is a more strict implementation of HTTP API. It was introduced and defined in 2000 by Roy Fielding in his doctoral thesis [15]. Fielding focusing on how a URI should represent an object, while HTTP verb serve as it's method. If the endpoint is fully adapting REST specification, it is called *RESTful web service*.

D. SOAP

SOAP (Simple Object Access Protocol) is a standard XML format for sending and receiving message [16]. In SOAP, every message has to be wrapped in an envelope element. Inside an evelope, developer can write header and body. Header should contains application specific information, while body should contains the message itself.

SOAP containing many standards that are not specified in HTTP/REST API. However, this also mean that SOAP is more verbose and eat more bandwidth than HTTP/REST API.

Nowadays, SOAP is still used and supported by many system.

E. CORBA, BPEL, and EJB

CORBA (Common Object Request Broker) is a specification created by OMG (Object Management Group). It was published in 1991, and it's last version was released on November 2012. CORBA supporting a lot of programming languages including C and Java [17].

BPEL (Business Process Execution Language) is another specification published by OASIS. I was published in 2003. BPEL is usually used by enterprise. Unlike CORBA, BPEL is more language agnostic. That means that the components can be written in any language.

EJB (Enterprise Java Beans) is another specification for microservice. compared to CORBA and BPEL, EJB is less language agnostic. in EJB, the orchestration component has to be written in Java.

CORBA, BPEL, and EJB are more strict compared to HTTP API. The server and the client should be agree on the data format. This agreement is usually written in IDL (Interface Definition Language). Since the server and the client has already agree on the data format, server response can be shorter than HTTP/REST API.

We conclude that although CORBA, BPEL, and EJB are more difficult to set up compared to HTTP/REST API and SOAP, it might benefit the system for the long run.

F. JSON-RPC, XML-RPC, and GRPC

RPC stands for Remote Procedure Call. The focus of RPC is to let developer invoke procedures in remote computer as easy as they are a local procedure. JSON-RPC and XML-RPC works in a same manner, only the data exchange format is different. As the names implied, JSON-RPC use JSON [8], while XML-RPC use XML [9]. First the client send a request to the server. The request contains the name of the method and the parameters. The server then replied with the return value.

As XML is more verbose than JSON, XML-RPC is also need more bandwidth compared to JSON-RPC. And as XML and JSON are text format, binary data should also be encoded in text format. For example, an image might need to be encoded in base-64 format.

To overcome the need to encode data into text format, Google create another protocol named GRPC (Google Remote Procedure Call). GRPC supporting several programming languages including Python, Javascript, Java, and PHP. In order to use GRPC [10], developers have to create stub skeleton (similar to IDL in CORBA, BPEL, or EJB).

IV. CHIML

CHIML (Chimera Markup Language) is a language used in Chimera-Framework. CHIML is an orchestration language. In term of language agnosticism and simplicity, Chimera-Framework is comparable to HTTP API. However, unlike HTTP API, Chimera-Framework doesn't even require HTTP protocol. Any valid command line executable can serve as component. In this research, our goal is to make CHIML as readable as possible.

A. Design

We build *CHIML* as a superset of *YAML*. So, any valid *YAML* is also a valid *CHIML*. And as *YAML* itself is a superset of *JSON*, any valid *JSON* is also a valid *CHIML*

Unlike *YAML*, in *CHIML* is the developers are allowed to write string after block delimiter (| and >) without changing the line.

The *CHIML script* should contain a single *[program]*.

B. Semantic (Backus Naur Form)

The brief overview of CHIML semantic is presented at Listing 1

Listing 1. CHIML Semantic

```
<program> ::=
 <completeVars>
 <completeVerbose>
 <command>
 <completeCatch>
 <completeThrow>

<command> ::=
  | <completeCommand>
  | <shortCommand>

<completeCommand> ::=
  | <completeIns>
    <completeOut>
    <completeIf>
    "do: "<process><newLine>
    <completeWhile>

  | <completeIns>
    <completeOut>
    <completeIf>
    "parallel: "<process><newLine>
    <completeWhile>

  | <completeIns>
    <completeOut>
    <completeIf>
    "do: "<commandList>
    <completeWhile>

  | <completeIns>
    <completeOut>
    <completeIf>
    "parallel: "<commandList>
    <completeWhile>

  | "map: "<variableName>
    "into: "<variableName>
    <completeCommand>

  | "filter: "<variableName>
    "into: "<variableName>
    <completeCommand>

<shortCommand> ::=
  | "|("<ins>") -> " <process> " -> " <out><newLine>
  | "|("<ins>") -> " <process> "<newLine>
  | "|"<process> " -> " <out><newLine>
  | "|("<ins>") --> " <out><newLine>
  | "|"<out> " <-- ("<ins>")"<newLine>

<commandList> ::=
  | "- "<command>
  | <commandList><commandList>

<completeCatch> ::=
  | ""
  | "catch: "<condition><newLine>

<completeThrow> ::=
  | ""
  | "throw: "<string><newLine>

<completeVars> ::=
  | ""
  | "vars: "<variableList><newLine>

<completeVerbose> ::=
  | ""
  | "verbose: "<verbosity><newLine>

<completeIns> ::=
  | ""
  | "ins: "<ins><newLine>

<completeOut> ::=
  | ""
  | "out: "<out><newLine>

<completeIf> ::=
  | ""
  | "if: "<condition><newLine>

<completeWhile> ::=
  | ""
  | "While: "<condition><newLine>

<ins> ::= <variableList>

<out> ::= <variableName>
```

```
<process> ::=
  | <cliCommand>
  | <jsArrowFunction>
  | "{"<jsNormalFunction>"}"
  | "["<jsFunctionWithCallback>"]"
  | "<"<jsPromise>">"

<variableList> ::=
  | <variableName>
  | <variableName>","<variableList>
```

Since we are focusing on how to make the language more compact, we also support several short-hands so that a single process can be written in a single line.

For example, the process in Listing 2 can also be written as in Listing 3

Furthermore, since developers might used to write left hand assignment in mainstream programming languages (i.e: *result = function(param1, param2)*), we think reverse arrow in CHIML can make it more intuitive. Thus, Listing 3 can also be written as 4.

Listing 2. CHIML Program Example
```
ins:
  month
  year
out: calendar
do: cal
```

Listing 3. CHIML Program Example (Short)
```
|(month, year) -> cal -> calendar
```

Listing 4. CHIML Program Example (Short-Reversed)
```
|calendar <- cal <- (month, year)
```

Despite of it's nature as orchestration language, CHIML also support several control structures like *if*, *else*, and *while*. It is useful for prototyping since the developers doesn't have to deploy intermediary components for control structure.

For example, to get the content of page-1 to page-10 of a website, the developer can do the iteration as displayed in Listing 5

Listing 5. CHIML Iteration
```
ins: url
out: html
do:
  - i <-- 1
  - while: i < 11
    do:
      - |(url + '?page=' + i) -> curl ->
        htmlFragment
      - |(html + htmlFragment) --> html
      - |i <-- (i+1)
```

We also take further analysis and conclude that for similar cases, it is better to do the task in parallel, so that the first request doesn't block the later requests. Since most modern programming language supporting *map* and *filter*, we also take a similar approach. For the previous example, we can build a more efficient solution as shown in Listing 6:

Listing 6. CHIML Map Feature
```
ins: url
out: html
vars:
  indexes: [1,2,3,4,5,6,7,8,9,10]
do:
  - ins: indexes
    out: responses
    map: index
    into: response
    do: |(url + '?page=' + index) -> curl ->
        response

  - |(responses, '') -> {\$.join} -> html
```

Map is more efficient since under the hood, each iteration will be done in a non-blocking manner.

C. Default Variables

There are some default variables in every CHIML script:

- *$*: Object, providing prebuilt utilities like $.httpRequestBody and $.join.
- *_chain_cwd*: String, current working directory of CHIML script physical location.
- *_process_cwd*: String, current working directory of CHIML script invocation location.
- *_error*: Boolean, error status.
- *_error_message*: String, error message.
- *_verbose*: Integer, verbosity level, default to *0*.
- *_ans*: Default output variable.
- *_runChain (chain, ...ins, callback)*: function to run other chain.
- *_maps (list, chain, callback)*: function to map an array into a new array. Under the hood, this will process each element in parallel.
- *_filter (list, chain, callback)*: function to filter an array into a new array. Under the hood, this will process each element in parallel.

D. Implementation

In order to parse and execute CHIML script, we have build a parser to evaluate the script and execute it on the fly. The javascript statements inside CHIML script are evaluated by using builtin Node.Js *vm* module. The global control flow is handled by using *neo-async*, a faster version of *async* module.

V. Experiment

A. Problem

For the test-case, we provide three HTTP API endpoint exposing three tables, *genres*, *books*, and *authors*. These HTTP end points are assumed as a black-box system. The only way to interact with those APIs is by sending HTTP GET request.

To select all the data from each end-point, the user can send GET request to *http://localhost:3000/[table-name]*. The user can also filter the data by adding *field-name* and *value* as GET parameter. For example, if the user want to select all

books written that have genreId = 5, the user should access this url: *http://localhost:3000/books?genreId=5*.

The response from those API will be in JSON format containing array of object. Each element of the array is correlated to table's row.

The table structure is as follow Listing 7:

Listing 7. Table Structure

```
TABLE: genres
FIELDS:
  - id
  - name

TABLE: books
FIELDS:
  - id
  - title
  - genreId
  - authorId

TABLE: author
FIELDS:
  - id
  - name
```

Using these API end points, we provide three problems. The first problem is named *g* problem. In order to solve *g* problem, we should transform HTTP API endpoint's response into array of genre's name (i.e: *["fiction","history","science"]*). In *g* problem, only one endpoint is involved.

The second problem is named *gb* problem. In *gb* problem we should fetch book titles for each genres and present it in the following manner: *["name":"fiction","books":["Rise of the Rebels","A New Dawn"],"name":"history","books":["John Adams","1776"],...]*. This problem is more difficult than the first problem, since we have to deal with two endpoints (genres and books).

The last problem named *gba* . It is very similar to *gb* problem, but in this case we should also show the author's name of every book. The expected result is as follow: *["name":"fiction","books":["title":"Rise of the Rebels","author":"Michael Kogge","title":"A New Dawn","author":"John Jackson Miller"],...]*

In order to measure the readability of the solutions, we use Flesch Kincaid readability test. while to measure the performance of the solutions, we use UNIX *time* utility.

B. Solutions

For each problems, we build 3 solutions. The solutions are using HTTP API protocol and written in three programming languages (Python, JavaScript, and CHIML). Some technologies like CORBA, BPEL, and EJB are not helpful to solve our problems since those technologies require IDL, broker setup, and other prerequisites. Like HTTP API, SOAP and RPCs are just protocols. Given the correct data encoder/decoder and suitable end point, we can also build the same solutions for these technologies.

In order to make the code shorter, we use *map* and *arrow function* for JavaScript, and using *lambda expression* for Python.

TABLE I. COMPARISON

b_real	b_sys	b_user	fkgl	fre	loc	problem	size	solution
0.246	0.022	0.236	8.390	42.035	30	gba	978	js
0.113	0.010	0.098	18.128	-28.851	15	gba	662	py
0.317	0.025	0.304	7.103	50.217	17	gba	554	chiml
0.230	0.018	0.226	11.044	22.670	16	gb	540	js
0.101	0.013	0.087	15.674	-11.228	12	gb	471	py
0.310	0.026	0.294	4.516	68.987	13	gb	381	chiml
0.221	0.021	0.214	19.039	-35.587	7	g	250	js
0.095	0.010	0.084	17.319	-23.463	6	g	217	py
0.293	0.020	0.287	4.613	68.522	6	g	163	chiml

The solutions of problem *g* are presented in Listing 8, 9, and 10:

Listing 8. JavaScript Solution for problem-g

```
const rpn = require('request-promise-native')
async function fetchGenres () {
  const body = await rpn('http://localhost:3000/
      genres')
  const genres = JSON.parse(body).map((genre) =>
      genre.name)
  console.log(JSON.stringify(genres))
}
fetchGenres()
```

Listing 9. Python Solution for problem-g

```
import json
from urllib import request
response = request.urlopen('http://localhost:3000/
    genres').read()
genres = json.loads(response)
result = list(map(lambda genre: genre['name'],
    genres))
print(json.dumps(result))
```

Listing 10. CHIML Solution for problem-g

```
out: genres
do:
  - |('http://localhost:3000/genres') -> [\$.
      httpRequestBody] -> genres
  - map: genres
    into: genres
    do: |(genre) -> (x) => x.name -> genre
```

VI. RESULT AND DISCUSSION

After building the solutions, we compare the performance by using UNIX *time* command. This benchmark providing 3 numbers, *sys*, *user*, and *real*. *Sys* is the amount of CPU time spent within kernel. *User* is the amount of CPU time spent outside the kernel but within the process. And *real* is the clock time from start to finish the call. *Real* also include time spent for IO process. We named these performance *b_sys*, *b_real*, and *b_user* respectively.

Beside comparing the performance, we also calculate the file size in byte (*size*), number of lines in code (*loc*), Flesch reading ease (*fre*), and Flesh-Kincaid grade level (*fkgl*).

For *b_sys*, *b_real*, and *b_user*, smaller number means better performance. For *size* and *loc*, smaller number is also means better. For *fre*, greater numbers means that the program is more readable. However, for *fkgl*, smaller number means smaller years of education required. This means that *fkgl* and *fre* should has negative correlation.

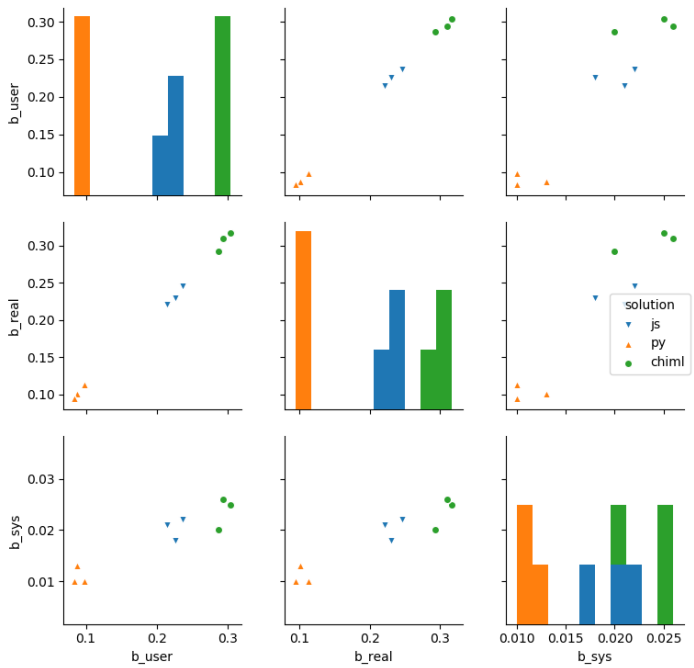

Fig. 1. Performance Comparison

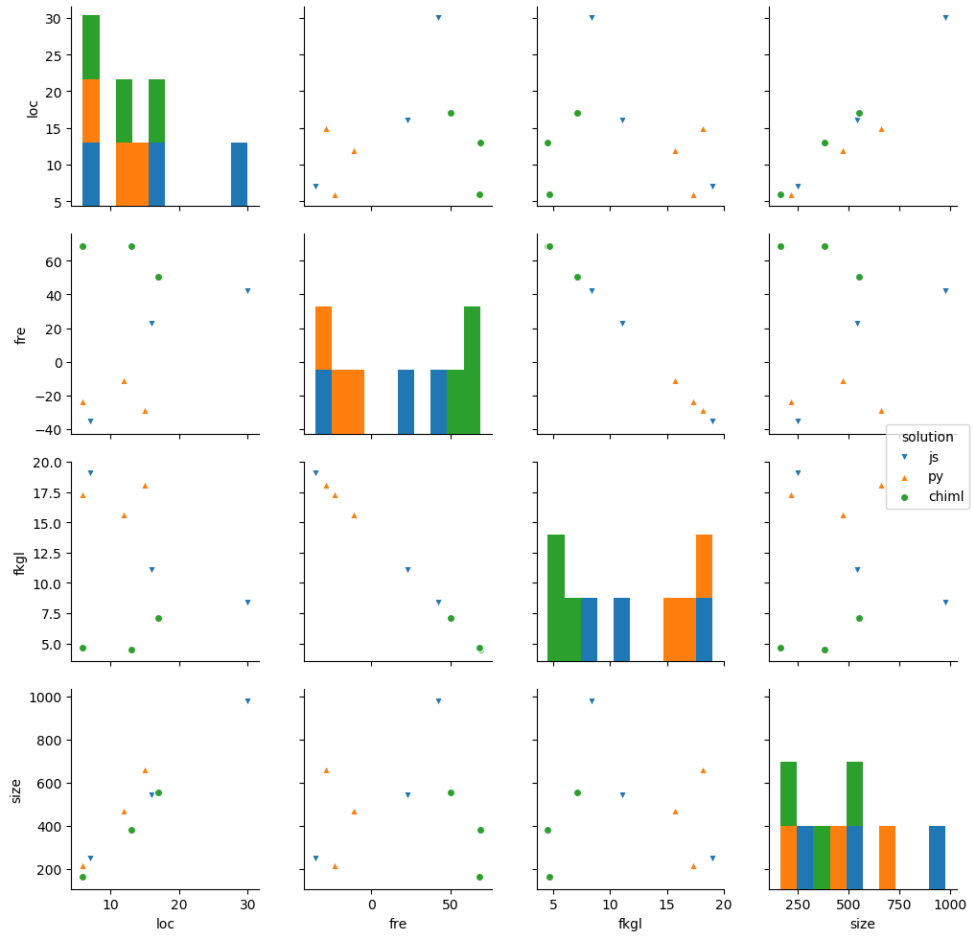

Fig. 2. Readability Comparison

The result is presented in the table I.

As shown in figure 1, we can conclude that Python solution is unexpectedly outperforming Javascript and CHIML. This is interesting, since in our solutions we didn't use multithread strategy. Python was expected to be slower because it has no non-blocking mechanism by default. However, the experiment show different result.

Also, *b_user* and *b_real* is strongly corelated to each other. This is make sense since the majority CPU time is spent in *user* mode.

In term of performance, CHIML solution is the slowest. This is also make sense, since under the hood, CHIML will be translated into JavaScript.

In terms of *size*, CHIML outperforming Python and JavaScript. The same is also true for *fkgl* and *fre*. Thus, we can conclude that according to Flesch-Kincaid readability test, CHIML is more readable from Python and JavaScript.

CHIML is unexpectedly slow when performing map or filter. It seems to be related to it's internal mechanism in translating map/filter. In CHIML, map and filter is translated into starting another subchain-execution.

We also realize that starting another external process (like UNIX tool) from CHIML is sometime take less time than running JavaScript expression inside CHIML (i.e: we use JavaScript to evaluate control structure). We are still trying to analyze this trait, but there are some possibilities that we have to consider. The first possibility is related to Chimera-Framework mechanism in evaluating JavaScript expression. Since internally we use *script.runInNewContext(context)*, the interpreter will always copy global variables and creating new context everytime. When the content of global variables increases, this can drag down the performance. The second possibility is related to Node.Js single thread nature. Eventough internally Node.Js can use more than one processor in a single thread, this will not work for our local code. In some cases, especially when the code doesn't involve IO process, everything will be executed sequentially.

So, for the sake of improving performance, we should refine CHIML parser and interpreter. Either by switching to a more concurrent language like *golang*, or by optimizing the current parser and interpreter. One of the possible solutions is using *child_process.fork* that allow us to run the sub-process in other thread.

VII. Conclusion

CHIML serve well as orchestration language. However for control structure, the existance of intermediary components can help to boost performance. The best trait of CHIML is it's support for programming-in-large and programming-in-small. Eventough the control structure is still suffering for speed and performance, it serves well as prototyping tool. In term of flexibility, CHIML is comparable to Python or JavaScript. In most cases, it even has smallest size and LOC than both, Python and JavaScript. This mean that the developer can start orchestration solution in CHIML, and gradually do optimization as needed.

Despite of it's advantage, CHIML interpreter has to be refactored and optimized in order to make it usable for real-world use-cases.

References

[1] D. H. Hansson, "Ruby on rails — a web-application framework that includes everything needed to create database-backed web applications according to the model-view-controller (mvc) pattern." https://rubyonrails.org, accessed: 2018-04-30.

[2] D. S. Foundation, "The web framework for perfectionists with deadlines — django," https://djangoproject.org, accessed: 2018-04-30.

[3] T. Otwell, "Laravel - the php framework for web artisans," https://laravel.com, accessed: 2018-04-30.

[4] B. K. Jr., "What is software vendor lock-in? (and how to avoid it)," https://www.makeuseof.com/tag/software-vendor-lock-avoid, accessed: 2018-04-30.

[5] M. Little, "Soa versus microservices?" https://www.infoq.com/news/2015/12/soa-v-microservices, accessed: 2018-04-30.

[6] J. Spacey, "Orchestration vs choreography," https://simplicable.com/new/orchestration-vs-choreography, accessed: 2018-04-30.

[7] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell, "Mass-produced software components," in *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, 1968, pp. 88–98.

[8] Trac, "Json-rpc," http://json-rpc.org, accessed: 2018-04-30.

[9] I. UserLand Software, "Xml-rpc.com," http://xmlrpc.scripting.com, accessed: 2018-04-30.

[10] I. Google, "grpc.io," https://grpc.io, accessed: 2018-04-30.

[11] T. Feilhauer and M. Sobotka, "Def-a programming language agnostic framework and execution environment for the parallel execution of library routines," *Journal of Cloud Computing*, vol. 5, no. 1, p. 20, 2016.

[12] G. F. Gunawan, J. F. Palandi, and M. Amien, "Chimera - simple language agnostic framework for stand alone and distributed computing," in *Computer Applications and Information Processing Technology (CAIPT), 2017 4th International Conference on*. IEEE, 2017, pp. 144–153.

[13] C. C. Evans, "The official yaml website," http://yaml.org/, accessed: 2018-04-30.

[14] F. DeRemer and H. Kron, "Programming-in-the large versus programming-in-the-small," *SIGPLAN Not.*, vol. 10, no. 6, pp. 114–121, Apr. 1975. [Online]. Available: http://doi.acm.org/10.1145/390016.808431

[15] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," in *Proceedings of the 22Nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 407–416. [Online]. Available: http://doi.acm.org/10.1145/337180.337228

[16] W. X. P. W. Group, "Soap version 1.2," https://www.w3.org/TR/soap12, accessed: 2018-04-30.

[17] O. M. Group, "Corba," http://www.corba.org/, accessed: 2018-04-30.

Lampiran 4. Laporan Penggunaan Anggaran 100%

A. Honor

No	Item	Vol	Satuan	Honor	Total
1	Honor Ketua	10	bulan	20,000	200,000
2	Honor Anggota 1	10	bulan	20,000	200,000
3	Honor Anggota 2	10	bulan	20,000	200,000
					600,000

B. Bahan Habis Pakai

No	Item	Vol	Satuan	Harga	Total
1	Kuota Internet	10	bulan	156,000	1,560,000
2	Fotocopy	1	paket	50,000	50,000
3	Tinta Printer	4	botol	50,000	200,000
4	Kertas A4	3	rim	48,000	144,000
					1.954,000

C. Perjalanan

1	Transportasi Belanja	3	orang	100,000	300,000
					300,000

D. Lain-lain

1	Akomodasi meeting	3	kali	100,000	300,000
					300,000

Total Pengeluaran: Rp. 3.154.000,-

Lampiran 5. Isian Data Kinerja Penelitian

DATA PENELITIAN		
Judul Penelitian	**Redesigning CHIML: Orchestration Language for Chimera-Framework**	
Jenis Penelitian	☐ Penelitian Dasar ✓ Penelitian terapan ☐ Pengembangan Eksperimental	
Bidang Penelitian	☐ Natural Science	☐ Mathematical Sciences ☐ Physical Sciences ☐ Chemical Sciences ☐ Earth Sciences ☐ Biological Sciences ☐ Information, Computing, and Communication Sciences ☐ Other Natural Sciences
	✓ Engineering Technology	☐ Industrial Biotechnology and Food Sciences ☐ Aerospace Engineering ☐ Manufacturing Engineering ☐ Automotive Engineering ☐ Mechanical and Industrial Engineering ☐ Chemical Engineering ☐ Resources Engineering ☐ Civil Engineering ☐ Electrical and Electronic Engineering ☐ Geomatics Engineering ☐ Environmental Engineering ☐ Maritime Engineering ☐ Metallurgy ☐ Materials Engineering ☐ Biomedical Engineering ☐ Computer Hardware ☐ Communications Technologies ☐ Interdisciplinary Engineering ✓ Other Engineering and Technology
	☐ Agricultural and Environmental Sciences	☐ Agricultural and Veterinary Sciences ☐ Environmental Sciences ☐ Architecture Urban Environment and Building ☐ Other Agricultural and Environmental Sciences
	☐ Medical Sciences	☐ Medical Sciences ☐ Public Health and Health Services ☐ Other Medical and Health Sciences
	☐ Social Sciences	☐ Education ☐ Economics ☐ Commerce, Management, Tourism and Services ☐ Policy and Political Sciences ☐ Studies in Human Society ☐ Behavioral and Cognitive Sciences ☐ Law, Justice, and Law Enforcement ☐ Journalism, Librarianship and Curatorial Studies ☐ Other Social Sciences
	☐ Humanities	☐ The Arts

		☐ Language and Culture ☐ History and Archeology ☐ Philosophy and Religion ☐ Other Humanities
Tujuan Sosial Ekonomi	☐ Defense	☐ Military and Politics ☐ Military Technology ☐ Military Doctrine, Education, and Training ☐ Military Capabilities ☐ Police and Internal Security
	☐ Plant Production and Plant Primary Products	☐ Field crops ☐ Plantation crops ☐ Horticultural crops ☐ Forestry ☐ Primary products from plants ☐ By-products utilization ☐ Herbs, Spices and Medicinal Plants ☐ Other plant production and plant primary products not elsewhere classified
	☐ Animal Production and Animal Primary Products	☐ Livestock ☐ Pasture, browse and folder crops ☐ Fisheries products ☐ Primary & by-products from animals ☐ Other animal production and animal primary products not elsewhere classified
	☐ Mineral Resources	☐ Exploration ☐ Primary mining and extraction processes ☐ First stage treatment of ores and minerals ☐ Prevention and Treatment of Pollution ☐ Other mineral resources (excluding energy) not elsewhere classified
	☐ Energy Resources	☐ Exploration ☐ Mining and extraction ☐ Preparation and supply of energy source materials ☐ Non-conventional energy resources ☐ Nuclear Energy ☐ Other energy resources not elsewhere classified
	☐ Energy Supply	☐ Energy transformation ☐ Renewable energy ☐ Energy distribution ☐ Energy Conservation and efficiency ☐ Energy issues ☐ Other energy supply not elsewhere classified
	☐ Manufacturing	☐ Processed food products and beverages ☐ Fiber processing and textiles, footwear and leather products ☐ Wood, wood products and paper ☐ Human pharmaceutical products ☐ Veterinary pharmaceutical products ☐ Agricultural chemicals ☐ Industrial chemicals and related products ☐ Basic metal products (including smelting)

		☐ Industrial mineral products ☐ Fabricated metal products ☐ Transport equipment ☐ Computer hardware and electronic equipment ☐ Communication equipment ☐ Instrumentation ☐ Machinery and equipment ☐ Latex product industry ☐ Standard supporting technologies ☐ Materials performance and processes/analysis ☐ Milling and process materials ☐ Synthesis and design of fine and specialty chemicals ☐ Consumer Products ☐ Other manufactured products not elsewhere classified
	☐ Construction	☐ Planning ☐ Design ☐ Construction processes ☐ Building management and services ☐ Other construction not elsewhere classified
	☐ Transport	☐ Ground transport ☐ Water transport ☐ Air & space transport ☐ Other transport not elsewhere classified
	✓ Information and Communication Services	✓ Computer software and services ☐ Information services (including library) ☐ Communication services ☐ Geoinformation Services ☐ Other information and communication not elsewhere classified
	☐ Commercial Services	☐ Electricity, gas and water services and utilities ☐ Waste management and recycling ☐ Wholesale and retail trade ☐ Finance, property and business services ☐ Tourism ☐ Other commercial services not elsewhere classified
	☐ Economic Framework	☐ Macroeconomics issues ☐ Microeconomics issues ☐ International trade issues ☐ Management and productivity issues ☐ Measurement standards and calibration services ☐ Commercialization ☐ Socio-economic development ☐ Economic development and environment ☐ Human resource management ☐ Other economic issues not elsewhere classified
	☐ Natural resources	☐ Soil resources ☐ Water resources ☐ Biodiversity ☐ Bioactive product ☐ Industrial raw materials ☐ Mineral resource

		☐ Other natural resources not elsewhere classified
	☐ Health	☐ Clinical (organs, diseases and conditions) ☐ Public health ☐ Health and support services ☐ Other health not elsewhere classified
	☐ Education and training	☐ Early childhood and primary education ☐ Secondary education ☐ Tertiary education ☐ Technical and further education
		☐ Special education ☐ Computer base teaching and learning ☐ Education policy ☐ Teaching ☐ Educational administration ☐ Other education and training not elsewhere classified
	☐ Social development and Community services	☐ Community services ☐ Public services ☐ Art, sport and recreation ☐ International relations ☐ Ethical issues ☐ Nation building ☐ Urban issues ☐ Other social development and community services not elsewhere classified
	☐ Environmental Knowledge	☐ Climate and atmosphere ☐ Ocean ☐ Water ☐ Land ☐ Nature conservation ☐ Social environment ☐ River and Lake ☐ Other environmental knowledge not elsewhere classified
	☐ Environmental aspects of development	☐ Plant production and plant primary products (including forestry) ☐ Animal production and animal primary products (including fishing) ☐ Mineral resources (excluding energy) ☐ Energy resources ☐ Energy supply ☐ Manufacturing ☐ Construction ☐ Transport ☐ Information and communication services ☐ Commercial services ☐ Environmental economic framework ☐ Other environmental of development not elsewhere classified
		☐ Environmental management ☐ Waste management and recycling ☐ Climate and Weather

	☐ Environmental management &other aspects	☐ Atmosphere (Excl. Climate and Weather) ☐ Marine and Coastal Environment ☐ Fresh water and Estuarine Environment ☐ Urban and Industrial Environment ☐ Forest and Wooded Lands ☐ Mining Environment ☐ Other environmental aspects not elsewhere classified
	☐ Advancement of Natural sciences, technology, and engineering	☐ Mathematical science ☐ Physical sciences ☐ Chemical sciences ☐ Earth sciences ☐ Information, computer and communication technologies ☐ Applied sciences and technologies ☐ Engineering sciences ☐ Biological sciences ☐ Agricultural sciences ☐ Medical and health sciences ☐ Multimedia ☐ Other Natural sciences, technology, and engineering not elsewhere classified
	☐ Advancement of Social sciences and humanities	☐ Social sciences ☐ Humanities ☐ Cyber law ☐ Other Social sciences and humanities not elsewhere classified
Sumber Dana		☐ Dalam negeri ☐ Luar negeri/Asing
Institusi Sumber Dana		☐ Pemerintah ☐ Swasta/industri ☐ Lembaga multilateral ☐ Lembaga nirlaba ☐ Internal perguruan tinggi ✓ Pribadi peneliti ☐ Sumber dana lain
Jumlah Dana		Rp. 3.154.000,-
Personil Dosen		NIDN : 0728108701 Nama Dosen : Go Frendi Gunawan Program Studi : Teknik Informatika
Personil Non Dosen		Nama : - Institusi : -