Chimera - Simple Agnostic Language Framework using Node.js and Command Line Arguments for Stand Alone and Distributed Computing

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

Mukhlis Amien
STIKI Malang
Malang, Indonesia
Email: amien@stiki.ac.id

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

***Abstract*—Component Based Software Engineering (CBSE) is a branch of software engineering that emphasizes the separation of concerns with respect to the wide-ranging functionality available throughout a given software system. The main advantage of CBSE is separation of components. A single component will only focus on a single task or related collection of tasks. Allowing software developer to reuse the component for other use-cases. By using this approach, software developer doesn't need to deal with spaghetti code. Several approaches has been developed in order to achieve ideal CBSE. The earliest implementation was UNIX pipe and redirect, while the newer approach including CORBA, XML-RPC, and REST. Our framework, Chimera, was built on top of Node.js. Chimera allows developer to build pipe flow in a chain (a YAML formatted file) as well as defining global variables. Compared to UNIX named and unnamed pipe, this format is easier and more flexible. On the other hand, unlike XML-RPC, REST, and CORBA, chimera doesn't enforce users to use special protocol such as HTTP (except for distributed computing scenario). Nor it require the components to be aware that they works on top of the framework.**

***Keywords*—Chimera, Language Agnostic, Component-Based Software Engineering, CBSE, Node.js, CLI.**

I. Introduction

Component based software development approach is based on the idea to develop software systems by selecting appropriate off-the shelf components and then to assemble them with a well-defined software architecture [1].

In order to implement component-based software engineering (CBSE), several approaches has beeen performed. The earliest attempt was UNIX pipe mechanism [2]. Pipe mechanism was not the only attempt to achieve CBSE. The more modern approaches including XML-RPC [3] and JSON-RPC [4]. Later, Object Management Group (OMG) introduced a new standard named CORBA (Common Object Request Broker Architecture) [5]. Another interesting approach was introduced by Two Sigma Open Source. Two Sigma created a platform known as Beaker Notebook [6]. Beaker Notebook is mainly used for research purpose. On 2016, Feilhauer and Sobotka introduce another platform called DEF [7].

Aside from Unix Pipe, all other mechanism require the components to be aware that they are part of the framework. This means that you cannot use old programs (e.g: *cal* and *cowsay*) as XML-RPC or CORBA component. At least additional layer and adjustment has to be built.

CORBA, XML-RPC, SOAP, and JSON-RPC also needs HTTP protocol since they were designed for in client-server architecture. It imply that developers need to build a web server in order to use the mechanisms. However, in any use case that only need a single computer, this is not ideal.

Considering the advantages and disadvantages of earlier approaches, in this paper, we develop a new CBSE framework named Chimera. This framework is much simpler since HTTP is only required for distributed computation. Chimera also use CLI mechanism that works in almost all OS and most programming language. The only dependency of Chimera are Node.js and several NPM packages.

II. Previous Research

In this section we will have an indepth discussion about previous CBSE implementation preceeding Chimera.

A. UNIX Pipe

The very first implementation of CBSE was UNIX pipe mechanism [2]. UNIX pipe allows engineer to pass output of a single program as an input of another program. Since a lot of server is UNIX or linux based, this pipe mehanism availability is very high. Even DOS also provide similar mechanism [8].

Pipe mechanism works by letting a program's standard output being used as another program's standard input. By putting several programs into a single pipeline, we can compose a more complex process as shown in figure 1.

For more explanation, we provide a simple test case. Consider two different program, *cal* and *cowsay*. Given no argument, *cal* will show you current month's calendar. On the other hand, given a single argument, *cowsay* will show the argument alongside an ASCII art of a cow.

Listing 1. cal and cowsay

```
#! cal
      June 2017
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

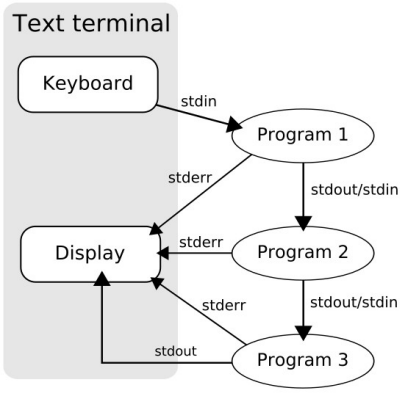

Fig. 1. Unix Pipeline Mechanism

```
#! cowsay "hello"
 _______
< hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Unix pipe mechanism allows you to combine those two programs. For example, if you want the output of *cal* become the input of *cowsay*, you can use pipe command as shown in listing 2

Listing 2. Unnamed pipe example

```
#! cal | cowsay
 ______________________________________
/  June 2017 Su Mo Tu We Th Fr Sa      \
|                                      |
| 1 2 3                                |
|                                      |
| 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 |
\ 19 20 21 22 23 24 25 26 27 28 29 30  /
 --------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Beside of it's high availability and simplicity, UNIX pipe also support parallel processing through named-pipe mechanism. The named-pipe mechanism can be used to provide cheap parallel processing [9].

In listing 3 we show a simple named-pipe mechanism. First, we make a named pipe called *backpipe* by using *mkfifo* command. Next, we redirect standard output of *cal* and *cowsay* into *backpipe*. Finally, we show the content of the *backpipe* by using *cat* command.

Listing 3. Named pipe example

```
#! mkfifo backpipe

#! cal > backpipe | cowsay hello > backpipe |
    cat backpipe
      June 2017
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

 _______
< hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Although pipe mechanism provide high availability and capability, it has several limitations. For example, named-pipe needs external file as temporary container. The external file has to be deleted once the operation performed. This approach is not straight forward, thus, some efforts is needed in order to to build a working named-pipe based computation.

For simple use cases involving a single computer, pipe mechanisme might be ideal. However, at some point, when the program become more complicated, memory sharing and network access might be needed. Using a mere pipe mechanism to support those requirement is either hard or impossible.

B. CORBA

From CORBA official website, CORBA is defined as standard created by the Object Management Group designed to facilitate the communication of systems that are deployed on diverse platforms [5]. CORBA 1.0 was released on August 1991. The last version, CORBA 3.3 was released on November 2012 [10]. CORBA is heavily affected by object oriented paradigm.

The main component of CORBA is the Object Request Broker (ORB). ORB act as bridge between client and service provider. The service provider (server) provide an implementation of an object. While the client can be a user interface that depend on the service provided by the server. Both, client and server needs to agree about the object structure. This agreement is written in an Interface Description Language (IDL). The IDL in server side is called skeleton, while the IDL in client side is called stub.

Figure 2 shows the interaction between ORB, server, and client. IDL can be written in Java, C++, or any other language, depend on the implementation of the ORB.

An IDL example is shown in listing 4.

Listing 4. CORBA IDL Example in C++

```
module Finance {
  typedef sequence<string> StringSeq;
  struct AccountDetails {
    string      name;
    StringSeq   address;
    long        account_number;
    double      current_balance;
  };
  exception insufficientFunds { };
  interface Account {
```

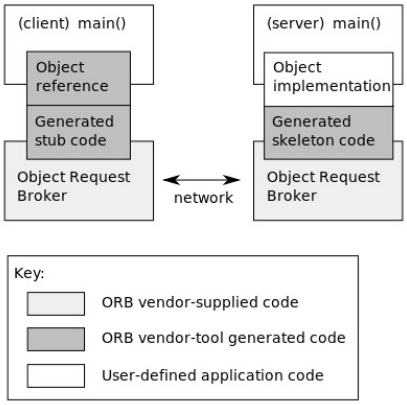

Fig. 2. Object Request Broker

```
    void deposit(in double amount);
    void withdraw(in double amount) raises(
        insufficientFunds);
    readonly attribute AccountDetails details;
  };
};
```

Compared to UNIX Pipe, CORBA is more feature rich and complex. The developer needs to embrace OOP paradigm as well as being familiar with IDL and the CORBA architecture. Despite of it's language agnoticism, some non OOP language (e.g: Matlab and GNU Octave) is not supported by CORBA [7]. CORBA also suffer of several criticism [11]. Even OOP as the foundation of CORBA, also face several critics [12] regardless of it's popularity.

C. XML-RPC, SOAP, and JSON-RPC

XML-RPC is a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet. XML-RPC using HTTP as the transport and XML as the encoding. It is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned [3].

SOAP stands for Simple Object Access Protocol. SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment [13]. SOAP was built on top of XML-RPC. It uses XML format as well as HTTP protocol.

JSON-RPC is lightweight remote procedure call protocol similar to XML-RPC [4]. The main difference between XML-RPC and JSON-RPC is the data transfer format. In most cases, JSON is more lightweight compared to XML.

XML-RPC, SOAP, and JSON-RPC are heavily depend on HTTP for inter-process-communication protocol. This is ideal for client-server architecture as HTTP is quite common and easy to be implemented.

Those three methods are basically another implementation of RPC (Remote Procedure Call). Compared to CORBA, these three methods are more flexible. With the exception of SOAP, they don't enforce developer to embrace OOP paradigm.

In terms of language agnoticism, XML-RPC and JSON-RPC support any language that can access HTTP and parse/create the data format. However, in order to use these protocols, a developer should be aware that the components they built will works as a part of the bigger system. Tools or programs that were built without this consideration will need some adjustment or additional layers in order to make them works with the protocol. For example, using *cowsay* or *cal* as components of XML-RPC might require developer to build another program to catch the output and wrap it in XML envelope.

D. DEF

DEF - A programming language agnostic framework and execution environment for the parallel execution of library routines [7]. DEF focus on parallel processing by enabling shared memory and message passing. DEF needs several components, using JSON as data exchange format. Compared to CORBA, Matlab, and Parallel Fortran, DEF is better in term of parallelism and language agnosticism. CORBA for example, doesn't support matlab and octave [7].

However, DEF still depend on HTTP for inter process communication. Consequently, in order to build DEF architecture, a web server is needed. Also, the developer needs to make sure that each components aware of the architecture. As in CORBA, XML-RPC, SOAP, and JSON-RPC, additional layer might be needed to make use of old components.

E. Beaker Notebook

Beaker Notebook [6] is also considered as an interesting approach of CBSE. The platform was developed by Two Sigma Open Source and mainly used for research use.

Beaker provides native autotranslation that lets a developer declare specific variables in a cell in one language, then access these seamlessly in a different cell and language.

Using Beaker Notebook, a developer can access a global inter-language variable from different cells. The cells can also be written in any language supported.

For example, in listing 5, we create a 6 by 4 table populated with random numbers. The table is then saved as global variable df. Later in listing 6, we load the data and show it.

Listing 5. Beaker Python Cell Example

```
import pandas
beaker.df = pandas.DataFrame(np.random.randn
    (6, 4), columns = list('ABCD'))
```

Listing 6. Beaker R Cell Example

```
beaker::get('df')
```

Beaker notebook is good for prototyping. It also has a very simple API compared to CORBA or XML-RPC. However, it still require the developer to add additional layer in order to use old components like *cal* or *cowsay*

III. Chimera Architecture

From the previous section we conclude that Unix Pipe Mechanism was the simplest one despite of it's lack of features. We also notice that Beaker Notebook's like memory-sharing mechanism is much simpler compared to CORBA and other network-based protocols.

Our goal is to make a very simple framework that is truly language agnostic. A framework that also play nice with old components and not enforce developer to embrace any particular programming paradigm. Also, we try to avoid making unnecessary new standard. By make use of tecnologies most developers familiar with, we hope the adaptation is going to be easier.

We assume that most programming languages are supporting command line interface and command line arguments. By creating a framework that depend on command line protocol, we aim on maximum language agnosticism with less effort.

In figure 3 we show the architecture of Chimera. Suppose *Program1* and *Program2* should run in parallel, and *Program3* should be executed once *Program1* and *Program2* finished.

IV. Chimera Technical Implementation

We already publish Chimera as NPM package. It is accessible through https://www.npmjs.com/package/chimera-framework.

A. Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient [14]. Compared to Python and PHP, Node.js has an overall better performance [15].

Node.js has a package-manager named NPM (Node Package Manager). This allows developers to use libraries that were already written by other developers. Chimera itself depend on several packages:

- async
- express
- fs-extra
- http
- js-yaml
- node-cmd
- path
- process
- querystring

B. YAML Chain

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

C. JSON Formatted Temporary Global Storage

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

D. Parallel Processing

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

E. Web Service

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

V. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue,

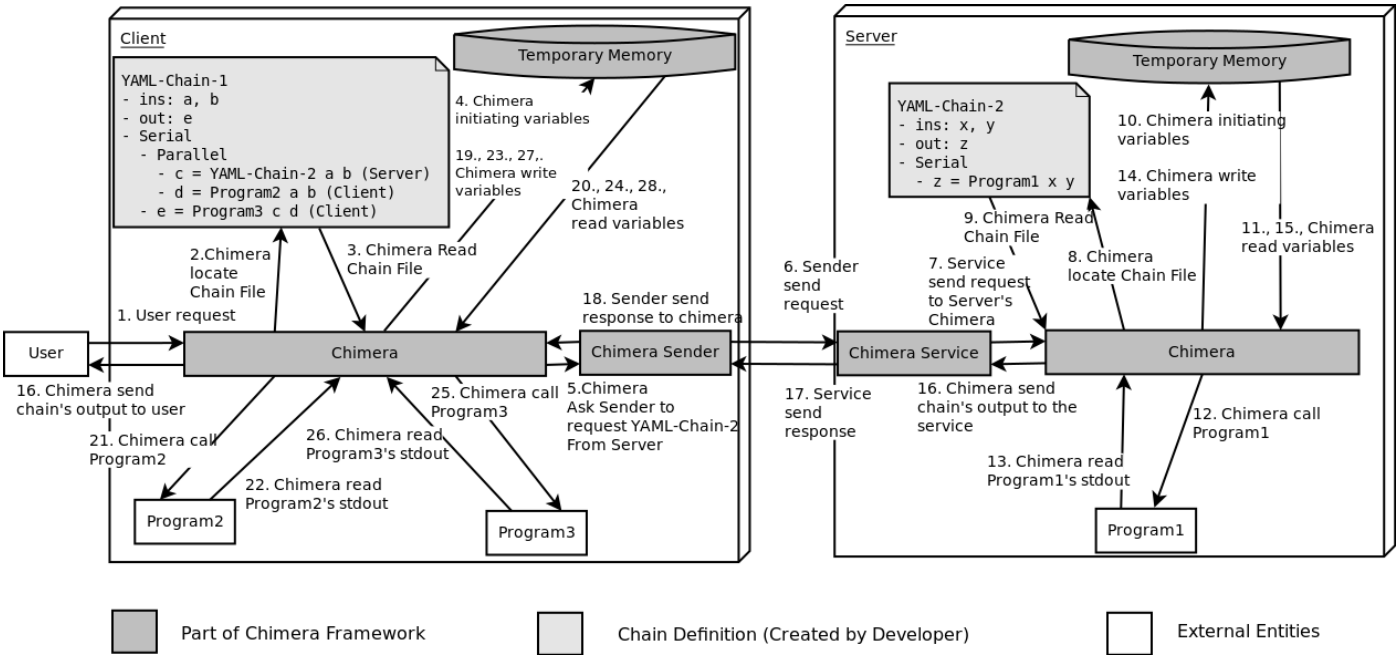

Fig. 3. Architecture of Chimera

a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] A. Kaur and K. S. Mann, "Component based software engineering," *International Journal of Computer Applications*, vol. 2, no. 1, pp. 105–108, 2010.

[2] 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.

[3] Trac, "Json-rpc," http://json-rpc.org, accessed: 2017-05-30.

[4] I. UserLand Software, "Xml-rpc.com," http://xmlrpc.scripting.com, accessed: 2017-05-30.

[5] O. M. Group, "Corba," http://www.corba.org/, accessed: 2017-05-30.

[6] L. T. Two Sigma Open Source, "Beaker notebook," http://beakernotebook.com/index, accessed: 2017-05-30.

[7] 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.

[8] B. Watson, "Dos 7 command," http://www.lagmonster.org/docs/DOS7/pipes.html, accessed: 2017-06-19.

[9] T. Conway, "Parallel processing on the cheap: Using unix pipes to run sas programs in parallel. sas users group international (sugi 28) proceedings. march 30–april 2, 2003. seattle, washington," 2003.

[10] O. M. Group, "Corba," http://www.omg.org/spec/CORBA/, accessed: 2017-05-30.

[11] M. Henning, "The rise and fall of corba," *Queue*, vol. 4, no. 5, pp. 28–34, 2006.

[12] I. Hadar, "When intuition and logic clash: The case of the object-oriented paradigm," *Science of Computer Programming*, vol. 78, no. 9, pp. 1407–1426, 2013.

[13] W. X. P. W. Group, "Soap version 1.2," https://www.w3.org/TR/soap12, accessed: 2017-05-30.

[14] N. Foundation, "Node.js," https://nodejs.org/en/, accessed: 2017-05-30.

[15] K. Lei, Y. Ma, and Z. Tan, "Performance comparison and evaluation of web development technologies in php, python, and node. js," in *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*. IEEE, 2014, pp. 661–668.