SCS Introduction

Author: Xiaoqun Gu(Rafael Gu)
Version: 1.1 (Revised by my friend, Dayong Li(Dave Li))
Date: December, 2011
Editor: Emacs23 and LibreOffice on Ubuntu 11.10

内容目录

Why there is an English document about SCS/ComEgg

==========

In June 2011, I shared some SCS/ComEgg ideas and relative Chinese documents with some connections on LinkedIn, because most of them are not Chinese, I promised I would write an English one.

Since finished the first version (demo) of ComEgg, I had met some people to discuss it in China from 2005 to 2011, but few of them could understand what's ComEgg. Most of them are just interested how to use or copy technologies from other countries, looks like they would never create something to meet their own requirements, share with others or benefit other people. And at the same time, ComEgg is not a kind of products which can make money for them directly and quickly, they don't like this. In 2008, Cloud Computing became popular in China. Although ComEgg can be used to implement some interesting Cloud features (I will talk about this below), none of them were care about this. The reality is, they stood up to tell others they were doing Cloud things and got money and supports from the government by some traditional Web applications or some copies of cloud products from other countries. So maybe I didn't make myself clear or I didn't find the right people, and further more I need to expand the scope from China to the world. For these reasons, it's a good idea to have an English document with more details about SCS/ComEgg.

With this document, I wish I can find someone from all over the world who can understand what SCS/ComEgg is and what I am doing, and I also wish maybe some of those people can gather together to do some interesting things in future. If I can find nine more this kind of person, I will be very happy and I will be encouraged to keep going. That's why I have to write an English document about SCS/ComEgg.

In November 2011, I finally have a break between my start-up project version 0.5.0 and 0.6.0. It's a good chance to finish the document while I am looking for a cheap Linux server to run the version 0.5.0 and optimize the UX. (This project is named Gongfuxin which means Kungfu Message, which a simple and easy to use tool is running on mobile phone with mixed elements from Weibo/Twitter and IM and LBS. I hope this product can bring me some money in future, also hope using SCS/ComEgg to re-establish the product's servers/cloud in the future. So if you have some friends are interested in investing in an innovative, interesting and project in China, trust me, Gongfuxin is a perfect one.)

There are some examples with C/C++ and XML code in this documents, if you don't know programming and coding, just ignore these code and continue. And also there are some special terms, like SCS, ComEgg, MagicEgg, RDS, etc, you can find out what they mean in the right place. So just ignore them before that.

You are welcome to share this document to anyone else.

Chinese is my native language. To me writing an English document is not easy as writing a Chinese one. I will appreciate if you could tell me the errors and ambiguities you find out, and also I am happy if you can share some advices and ideas with me(Thank my friend, Dayong Li(Dave Li), he reviewed the document and made some updates). The document has not a lot of contents, so that you can read it quickly. If you want more details, please let me know, we can chat by IM tools or email each other. Following are my email addresses, you'd better send to both for safe:

shoutrain_goo@yahoo.com.cn

shoutrain.goo@gmail.com

No one know more about Cloud than others before 2015

==========

It's similar to the beginning of 1990s(1989), when Internet made the first formal step, and a lot of people thought Internet will change the world by connecting people and sharing information with

each other although they don't know exactly how. At that time, people just knew modem, switch, router, telnet BBS, email, and maybe a little about HTML and browser. Then in the middle of 1990s, Yahoo rose. That time, people thought "Yeah, Internet should change the world like this". Then the Bubble rose and followed by an big burst, many people might think "Oh, just a bubble, game over!". Then the end of 1990s and beginning of 2000s, Google rose, then blog, then Facebook, Twitter ... So we can say no one know more about Internet than others before 1995 or 2000 or later. If I replace the Internet by Cloud and change the date of year, it still works now: No one know more about Cloud than others before 2015 or later.

Yes, from technology perspective Cloud now is like Internet at the beginning of 1990s. Now we have a lot of technologies to build the infrastructure of Cloud, like Virtualization, Distributed Storage, etc. There are also some kinds of Cloud categories, like IaaS, PaaS, SaaS, etc. Actually I am not an expert in any of these "new" technologies, and I has no development experience on Virtualization, Distributed Storage either, I just know what they are for and how to use them. But I do has lots of development experiences on distributed communication applications(C/C++) and Web 1.0/2.0 applications(Java&PHP) and mobile phone applications(Windows Mobile and Android).

	Internet	*Cloud*
Infrastructure	Modem, Switch, Router, Telnet ...	Virtualization, Distributed Storage, On-line Office, ...
More and more information	Yahoo, eBay, YouTube, Google, Gmail...	See Q1 below
More and more users	Facebook, twitter, Weibo, Skype, ...	See Q2 below

Q1: **"Can information in Cloud connect each other to compose a set of new information to meet the people's new and dynamic requirements?"**

Q2: **"Can people and information involved by Cloud connect each other automatically?"**

In this document you will not find anything like how Virtualization or on-line technologies change old computer mode from providing hardware and software copies to on-line custom-built services, that's what most of us are doing in CURRENT STEP, which I call it Infrastructure of Cloud. Actually in the document I will talk about the NEXT STEP: What we will do when more information and users involved by Cloud and how to manage them by using SCS/ComEgg(like the above two questions).

Yes, we always know clearly about current step, no one know more details about next steps. But some of us always make tries; we always create future by tries instead of by some simple predictions. I am making tries.

Unless you don't think you are some kinds of Cloud experts anymore or you want to know what the tries I am making for next step, you'd better throw the document away. Remember this, no one know more about future than others.

Why I developed ComEgg

==========

I love TCP/UDP and RCP binary based protocols more than HTTP/SOAP/Web Services string based protocols. The former is more efficient, faster, less size and server-side-push supported. But the latter is easier to use, so we can see more and more applications are based on the latter. My first goal to design and develop ComEgg(in 2004 and 2005) is to make binary based protocols easy to use, even easier than string based protocols.

That time(2004) I was working for a small company which provides value-added services for telecom companies. Every day I had to handle a lot of binary PDUs(Protocol Define Units) based on SMPP/CMPP/USSD and corresponding transactions. Normally I created a PDU by C/C++, then loaded its fields to memory, then process them and sent it to other peers by sockets. It's not easy to handle these PDUs no matter it's size fixed or not, like following examples, we need to identify them, define a stack variable or new memory to process it, debug it between sender and receiver ...

```
// Fixed size
// ask server to get friends list
typedef struct
{
    unsigned int uiSize; // = 20
    unsigned int uiCommandId; // = N_GET_FRIENDS_REQ
    unsigned int uiExtra1;
    unsigned int uiExtra2;

    unsigned int uiOwnerid;  // = current user id, used for server without session kept
} TGetFriendsReq;

// Fixed size
// be sure server has got TGetFriendsReq on time
typedef struct
{
    unsigned int uiSize; // = 8
    unsigned int uiCommandId; // = N_GET_FRIDNES_REQ_ACK
} TGetFriendsReqAck;
```

It's even harder to handle these PDUs with variable size, which means there are some variable fields in the PDU, like the example below:

```
// unfixed size
// give client the friends list it wants
typedef struct
{
```

```
    unsigned int uiSize; // we don't don't know the size in advance
    unsigned int uiCommandId; // = N_GET_FRIENDS_RES

    unsigned int uiFriendsNum;
// fake code for the variable fields
/*
    struct TNameUnit
    {
        unsigned char ucNameSize;
        char szName[ucNameSize];
    };

    TNameUnit *pNameList;
/*
} TGetFriendsRes;

// fixed size
// be sure client has got TGetFriendsRes it wants
typedef struct
{
    unsigned int uiSize; // = 8
    unsigned int uiCommandId; // = N_GET_FRIENDS_RES_ACK
} TGetFriendsResAck;
```

Of course, we should define the command ids in advance, like below:

```
const unsigned int N_ACK = 0x10000000;
const unsigned int N_RES = 0x01000000;

const unsigned int N_GET_FRIENDS_REQ      = 0x00000001;
const unsigned int N_GET_FRIENDS_REQ_ACK  = N_GET_FRIENDS_REQ | N_ACK;
const unsigned int N_GET_FRIENDS_RES      = N_GET_FRIENDS_REQ | N_RES
const unsigned int N_GET_FRIENDS_RES_ACK  = N_GET_FRIENDS_RES | N_ACK;
```

Composing and parsing this kind of PDUs are difficult. We have to identify the PDUs when we get them, and then control memory to load them, and process the fields by calculate the offset of the memory. It's also hard to debug it on both sides due to we have to dig into the memory to see what the bytes mean. And these PDUs are also easy to causes bugs.

So I designed(2004) and implemented(2005, two times with two versions) ComEgg to make the binary PDUs easier to use. How did I achieve that? You will know that after reading following example from ComEgg RDS(Role Describe Script), which is the corresponding solution for above PDUs:

```
<protocol xmlns="http://www.comegg.com/protocol" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.comegg.com/protocol protocol.xsd" name="example_get_friends">
  <command_id_value name="N_GET_FRIENDS_REQ" value="0x00000001"/>
  <!-- I have not implemented definition like this: N_GET_FRIENDS_REQ | N_ACK; -->
  <command_id_value name="N_GET_FRIENDS_REQ_ACK" value="0x10000001"/>
  <!-- I have not implemented definition like this: N_GET_FRIENDS_REQ | N_RES; -->
  <command_id_value name="N_GET_FRIENDS_RES" value="0x01000001"/>
  <!-- I have not implemented definition like this: N_GET_FRIENDS_RES | N_ACK; -->
  <command_id_value name="N_GET_FRIENDS_RES_ACK" value="0x11000001"/>
  <head>
    <normal name="size" length="4" signed="false"/>
```

```
    <normal name="command" length="4" signed="false"/>
  </head>
  <pdu name="GetFriendsReq">
    <string name="UserName" size="16"/>
    <string name="Password" size="16"/>
  </pdu>
  <pdu name="GetFriendsReqAck" />
  <pdu name="GetFriendsRes">
    <normal name="FriendsNum" length="16" signed="false"/>
    <group name="FriendName" size="FriendsNum">
      <normal name="NameSize" length="1" signed="false"/>
      <string name="Name" size="NameSize"/>
    </group>
  </pdu>
  <pdu name="GetFriendsResAck" />
</protocol>
```

Yes, we defined the PDUs by XML with corresponding XSD rules. Even more, actually you can use these fields directly like global variables or local variables or parameters you used in C/C++, Java programs. Yeah, you don't need to identity the PDUs and parse them and new memory to load them and calculate offset to dig them, just use them like common variables and let them work with defined variables directly. Besides this, I also implemented some basic operators, like +, -, *, /, ~, !, =, ==, >, >=, <, <=, %, |, ||, &, &&, if, else if, while, do ... while, break, continue, ...; and also let the ComEgg can support .dll(windows) or .so(Linux) plug-ins. Now let me show you an example below, which is a small part from an SCS example(named SCSBlog, with MySQL supported) to register and login a simple blog system, all PDUs has defined in another XML file with the same way as example above:

```
<transactions xmlns="http://www.comegg.com/transactions" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.comegg.com/transactions transactions.xsd">
  <processor name="scsblog_server">
    <data_block>
      <v_ub_4 name="LoginedUserID"/>
    </data_block>
    <handle identity="sscblog_server.RegisterUser" direction="in">
      <data_block>
        <v_ub_4 name="UserID" />
      </data_block>
      <process_block>
        <empty>
          <solid_variable name="LoginedUserID"/>
        </empty>
        <use_module module="SCSBlog" interface="RegisterUser">
          <parameter_in>
            <network_variable field_name="UserName"/>
            <network_variable field_name="Password"/>
          </parameter_in>
          <parameter_out field_name="uiUserID">
            <solid_variable name="UserID"/>
          </parameter_out>
        </use_module>
        <send protocol="scsblog" pdu="RegisterUserAck">
          <field_variable field_name="UserID">
            <solid_variable name="UserID"/>
          </field_variable>
```

```
        </send>
      </process_block>
    </handle>
    <handle identity="sscblog_server.VerifyUser" direction="in">
      <data_block>
        <v_ub_4 name="UserID" />
      </data_block>
      <process_block>
        <empty>
          <solid_variable name="LoginedUserID"/>
        </empty>
        <use_module module="SCSBlog" interface="VerifyUser">
          <parameter_in>
            <network_variable field_name="UserName"/>
            <network_variable field_name="Password"/>
          </parameter_in>
          <parameter_out field_name="uiUserID">
            <solid_variable name="UserID"/>
          </parameter_out>
        </use_module>
        <send protocol="scsblog" pdu="VerifyUserAck">
          <field_variable field_name="UserID">
            <solid_variable name="UserID"/>
          </field_variable>
        </send>
        <if>
          <condition_unitary>
            <solid_variable name="UserID" />
          </condition_unitary>
          <program>
            <process_block>
              <unitary_calculate>
                <solid_variable name="LoginedUserID"/>
                <solid_variable name="UserID"/>
              </unitary_calculate>
            </process_block>
          </program>
        </if>
      </process_block>
    </handle>
  </processor>
</transactions>
```

By the goal to make the binary based protocols easy to use, I developed ComEgg on Windows in 2005, two times, in the beginning of 2005 and the end of 2005 respectively. I also developed some examples, in which the clients are based on MFC and the servers are all based on ComEgg. Actually I don't need to care about binary PDUs anymore on server side, and combine the binary protocols and server applications which means you can define a field in a PDU as a local variable in applications ... I achieved my goal.

But it's only the first step and I didn't stop. I was inspired a lot by what I did and some biology books I read later. So I thought more and had more ideas. You see I implemented a system that makes the binary protocols easy to use, even more I implemented an interpreter and prototype of light VM(Virtual Machine, like JVM) and corresponding development language(like Java), actually they are similar with cell and gene more than JVM interpreter and Java. I will talk more

about this in next sections: *What's ComEgg, MagicEgg and RDS*, *What's SCS*.

What's ComEgg, MagicEgg and RDS

==========

You should have an rough idea about ComEgg by the previous section. Actually, in 2005 ComEgg is just a Windows .exe file(needs ACE and Xerces4C libs), and it will load RDS(XML files) every time before it launches, and just works as the RDS describes. And then in 2009, I ported ComEgg to Linux, and let ComEgg support some I/O functions(to implemented some Gene and Cell features, I will discuss it in next section) and third part modules loading(to connect Database, like MySQL), like C/C++ .so dynamic link files. And also implemented MagicEgg(client version of ComEgg needs ACE, Xercex4C and FLTK libs), which can work with ComEgg, like a web browser works with web servers(like Apache, Tomcat, etc).

In this section, I will take a comparison between ComEgg/MagicEgg/RDS and Web Server/Web Browser/Dynamic Web Script(like IIS/Firefox/ASP, Apache/Firefox/PHP, Tomcat/Firefox/JSP, etc. And you can replace the Firefox by IE, Chrome, Safari, etc.) to make you have a thorough understanding about ComEgg. And then I will explain why I choose XML as the RDS format, instead of some c-like language format, like C/C++, PHP, Java, C#, etc.

Please notice the comparison between ComEgg/MagicEgg/RDS and Web Server/Web Browser/Web Dynamic Script doesn't mean the former is good enough, actually we have to take a long time to make it grown-up. For now I only implemented the prototype of ComEgg/MagicEgg/RDS. I do the comparison because most of you know Web Server/Web Browser/Web Dynamic and the both patterns are similar.

	ComEgg	*Web Server*
Protocol Type	Any binary protocols	Mostly string protocols, like HTTP/HTTPS, FTP, etc.
Push supported	Yes	Mostly doesn't support
Size	Small, Current ComEgg is about 300k	Big, Mostly 10M or more
Plugin supported	Yes	Yes

	MagicEgg	*Web Browser*
Protocol Type	Any binary protocols	Mostly string protocols, like HTTP/HTTPS, FTP, etc.
UI Components	Desktop controls	HTML/Web controls
Locate	User defined	URL

	RDS(Role Describe Script)	*Dynamic Web Script*
Script Format	XML/XSD	Traditional language, like Java, PHP, etc. HTTP/HTTPS, FTP, etc.

You have known how ComEgg works by the previous section. Actually MagicEgg works in the same way, that is loading RDS and works as RDS describes, the only difference is MagicEgg is for client when ComEgg is for server. I have an example - SCSBlog which is based on ComEgg and MagicEgg. You can find register, login and blog list RDS below, which is a part of SCSBlog example(there are more containers in this RDS actually):

```
<?xml version="1.0" encoding="UTF-8"?>
<interface xmlns="http://www.magicegg.com/interface" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.magicegg.com/interface interface.xsd">
  <container name="register" left="0" top="0" width="200" height="200" title="Register">
    <label name="label_1" left="0" top="0" width="100" height="50" value="register Name:"/>
    <text name="text_register_name" left="100" top="0" width="150" height="50" maxlength="16"/>
    <label name="label_2" left="0" top="50" width="100" height="50" value="register Password:"/>
    <text name="text_register_password_1" left="100" top="50" width="150" height="50" maxlength="16"
secret="true"/>
    <text name="text_register_password_2" left="100" top="100" width="150" height="50" maxlength="16"
secret="true"/>
    <button name="button_register" left="0" top="150" width="100" height="50" value="Register"/>
    <button name="button_back" left="0" top="100" width="100" height="50" value="Back"/>
  </container>
  <container name="login" left="0" top="0" width="200" height="200" title="Login">
    <label name="label_1" left="0" top="0" width="100" height="50" value="Login Name:"/>
    <text name="text_login_name" left="100" top="0" width="150" height="50" maxlength="16"/>
    <label name="label_2" left="0" top="50" width="100" height="50" value="Login Password:"/>
    <text name="text_login_password" left="100" top="50" width="150" height="50" maxlength="16"
secret="true"/>
    <button name="button_login" left="0" top="100" width="100" height="50" value="Login"/>
    <button name="button_register" left="100" top="100" width="100" height="50" value="Register"/>
  </container>
  <container name="main" left="0" top="0" width="800" height="600" title="SCS Blog Index">
    <table name="table_blog_list" left="0" top="50" width="800" height="500">
      <column name="User Name" width="200"/>
      <column name="Blog Title" width="200"/>
      <column name="Create Time" width="200"/>
    </table>
    <button name="button_open_blog" left="0" top="550" width="100" height="50" value="Open Blog"/>
    <button name="button_write_blog" left="100" top="550" width="100" height="50" value="Write Blog"/>
    <button name="button_my_blog" left="200" top="550" width="100" height="50" value="My Blog"/>
    <button name="button_my_info" left="300" top="550" width="100" height="50" value="My Info"/>
    <button name="button_logout" left="400" top="550" width="100" height="50" value="Logout"/>
    <button name="button_weather_c" left="550" top="550" width="100" height="50" value="Weather(C)"/>
    <button name="button_weather_cs" left="650" top="550" width="150" height="50" value="Weather(C and
S)"/>
  </container>
</interface>
```

Following RDS is the corresponding register container handle RDS, do you remember we can use PDU fields same as the local defined variables in ComEgg, it's the same in MagicEgg, further more you can use UI controls as a local defined variable too.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions xmlns="http://www.magicegg.com/actions" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.magicegg.com/actions actions.xsd">
  <data_block>
    <v_ub_4 name="user_id"/>
    <v_string name="user_name"/>
    <v_b_4 name="blog_id"/>
    <v_ub_4 name="zero" value="0"/>
  </data_block>
  <start>
    <process_block>
      <show_window name="login"/>
    </process_block>
  </start>
  <processor name="register">
    <on_click name="button_register">
      <process_block>
        <if>
          <condition_duality calculate="gl">
            <interface_variable name="text_register_password_1"/>
            <interface_variable name="text_register_password_2"/>
          </condition_duality>
          <program>
            <process_block>
               <alert message="Tow passwords is not match!"/>
               <block/>
            </process_block>
          </program>
        </if>
        <send pdu="RegisterUser">
          <field_variable field_name="UserName">
            <interface_variable name="text_register_name"/>
          </field_variable>
          <field_variable field_name="Password">
            <interface_variable name="text_register_password_1"/>
          </field_variable>
        </send>
        <wait_message pdu="RegisterUserAck"/>
      </process_block>
    </on_click>
    <on_click name="button_back">
      <process_block>
        <show_window name="login"/>
      </process_block>
    </on_click>
    <on_message pdu="RegisterUserAck">
      <process_block>
        <if>
          <data_block>
            <v_ub_4 name="invalid_user_id" value="0"/>
          </data_block>
          <condition_duality calculate="gt">
            <network_variable field_name="UserID"/>
            <solid_variable name="invalid_user_id"/>
          </condition_duality>
          <program>
            <process_block>
              <show_window name="login"/>
            </process_block>
          </program>
          <program>
            <process_block>
```

```
            <alert message="Register failure!"/>
          </process_block>
        </program>
      </if>
    </process_block>
  </on_message>
 </processor>
</actions>
```

Web Browser can read HTML files locally and remotely(HTML content produced by server side), for now MagicEgg only support read RDS locally. MagicEgg should support read RDS remotely(that is get RDS from ComEgg), but I have not implemented it yet. I implemented MagicEgg in 2009, and I am not sure whether I should abandon it and focus on ComEgg later.

Beside ComEgg and MagicEgg, RDS - Role Describe Script, is a very important part of SCS. RDS describes how ComEgg and MagicEgg work. Maybe you think it's not a good choice to use XML(instead of Java or C#) to present RDS, you see, XML is redundant, big volume. But why XML is the one? 4 reasons:

1.As the name says, Role Describe Script, we have to describe all features about the ROLE, include the static and dynamic data, modules, and how it works, and what it looks like, ... it's a bit like to describe a role which will be performed by an actor/actress. Obviously only c-like language format is not enough. It's one of the two most important reasons.

2.XML is easy to write and read for both people and machines, especially for machines. It's very easy to merge two RDSs for two different services into one and run it in one ComEgg or Magic. It's also very easy to split one big RDS into several parts and run them in several different ComEggs or MagicEggs. And XML is easy to exchange between peers. It's also easy to do above jobs in running time. It's another one of the two important reasons why I choose XML. You can see why this feature make SCS so different in the next section.

3.XML is easy to transform from one form to other forms. We can transform from XML to diagrams, or Java, or C++, etc. Imaging this we develop a whole distributed system only by drag&drop some diagrams, and then transform these diagrams to XML as RDS, and let ComEgg/MagicEgg to run the RDS to implement the whole distributed system. For now, we have to develop any kind of applications by coding, except the UI part (We can use XML or drag&drop to implement the UI part now).

4.XML and XSD(XML Scheme) make my work easy. You see I don't need to develop a parser, just using XML parser(Xerces4C) to validate, parse RDS. Actually I have no Compiler or Interpreter development experience before.

By the way, RDS is not powerful enough to do anything like C/C++, Java. So it's very important to let RDS work with other modules which is written by C/C++, Java. For now it can only work with C/C++. Actually RDS should be a good "stick" script later to stick different modules written by different languages and platforms.

You can find more reasons in the next section: What's SCS.

What's SCS

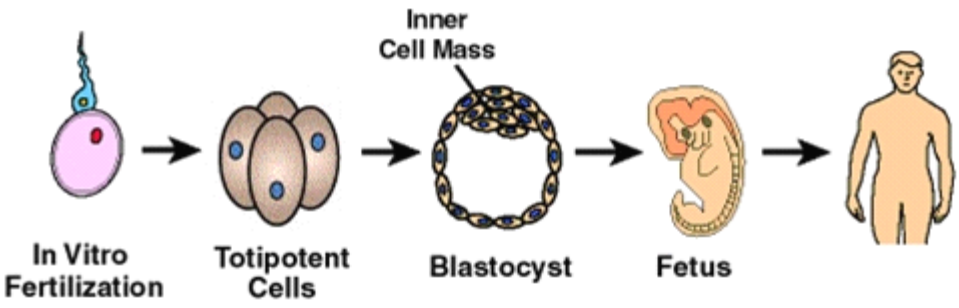
==========

SCS, Simulate Cellula System.

After I finished the first version of ComEgg at the beginning of 2005, some friends introduced some books about gene and then I was inspired by one of these gene books and bought another Biology book. After I read the first three chapters of the Biology book I am interested by the relationship between gene and Cell: everyone of us starts form one cell(In Vitro Fertilization) with DNA(gene information), and then it becomes Totipotent Cells, Blastocyst, Fetus and our body in succession. In brief I just described a simple process a cell with gene information becomes a lot of different kinds of cell in which there are same gene information.

It's a little hard to understand, right? Actually I am not so professional for this process and Biology, I suggest you take a short time to read some books about it. I just like the abstract idea of the process by which I can use ComEgg to compose an interesting system, in which every unit is composed by actor and role description like cell and gene information. There are two methods in the system, one is Division, another one is Differentiation.

Division



In one unit on actor can clone itself many times to take the same role to produce more units, and we can achieve Elastic Computer. That is if we want more units to do the same job, we just need to clone more this kind of actors with the same role. And of course we can kill some this kind of actors if we need fewer units later. Besides Elastic Computer, clone some actors to other place and killing some actors here can implement Liquid Computer(you can read the content below to find the meaning).

Differentiation

In one unit on actor can copy itself many times to take some different roles to produce more units and let these units work with each other like Team Works. That is if we want more units to do some different and interrelated jobs, we just need to copy more this kind of actors with different roles. And of course we can kill some this kind of actors if we want to close some function later. Here I said Team Works, it not only means making different modules from one company work together, also means making different modules from different companies work together.

If you want to understand these things easier, you can look the actor as Cell, the role as gene information; on the other hand, you can look the actor as ComEgg/MagicEgg, the roles as RDS. So we can compose a system by ComEgg/MagicEgg and RDS, in which we can change the computing power(Elastic Computer) dynamically by Division and change the functions(Team Works) dynamically by Differentiation, that is the thing I call SCS, and you should believe I have implemented Division and Differentiation by reading previous sections(I can demo for you if you want).

Elastic Computer automatically

I know current Cloud has implemented Elastic Computer manually, and recently someone told me actually some Clouds has implemented Elastic Computer automatically in Virtualization Layer. I am not sure how they implemented it and if it can support decreasing and increasing computer units in running time. I am exactly sure ComEgg can implemented Elastic Computer automatically in Application Layer, and it can support decreasing and increasing computer units in running time.

For example, now there is a service named Gongfuxin, which is described by RDS and working on SCS Cloud. At the beginning, Gongfuxin has less than 10000 users, and one SCS computer units can support 10000 users. Some days later, more users want to use Gongfuxin, SCS Cloud will find some free computer units and let the ComEgg to run Gongfuxin RDS. How these new ComEgg can find Gongfuxin RDS? Actually ComEggs can transform RDSs each other. How these ComEggs can know each other? We can let one computer unit to work as a Load Balance server or proxy, so that all other ComEggs can know each other by querying this unit.

Liquid Computer automatically

On the other hand, I don't think it's enough to implement Elastic Computer, actually implementing Liquid Computer is also important too. What? Liquid computer? Never heard that. It means some computer units can be moved from one

place to another place.

For example, we use the service above - Gongfuxin. Today there are 10,000 users in China and 0 users in America, and Gongfuxin are running a SCS Cloud which is variable for both America and China and is run by one telecom company. Tomorrow there will be 0 users in China and 1,000 users in America, how should we do? In SCS Cloud, it should kill the computer unit close to China and transform RDSs to computer units close to America automatically.

Team Work automatically

Cloud Teams Works anther one you never heard before. Do you remember the two questions mentioned in the previous section: No one know more about Cloud than others before 2015? The questions are:

"Can information in Cloud connect each other to compose a set of new information to meet the people's new and dynamic requirements?"
"Can people and information involved by Cloud connect each other automatically?"

Actually now most of companies put their service into Internet or Cloud independently. I mean they have their own domain names, applications and servers. If two or more companies want to work together to provide end-users a set of services, like eBay + PayPal + FedEx or Taobao + AliPay + DHL, they have to work manually in advance to make protocols or interfaces (based on HTTP/HTTPs or WebService) to connect each other. This approach takes time and money, and it's bad in a sense that company cannot focus on their own business, for example, B2C companies have to consider payment and logistics besides their main business - selling stuffs.

Although more and more companies entere into Internet or cloud, such as B2C companies, payment companies, logistics companies, why we cannot connect them automatically? In cloud, plenty of companies will put their services in the cloud, I believe more and more users need combined services from several providers at the same time. We should make some standards to make it true automatically.

By reading previous sections, you should

know RDS is a kind of full-featured description script. So for different services, we can make some of their RDSs public, like what the services are and how the services are connected, then we can borrow ideas from TCP/IP and Internet classify these services to indexes (like Yahoo did before) and store these indexes in some special computer units(like DNS) of the Cloud, and make standards to define how different services can work together(like TCP/IP defined switch and router rules in the 3rd and 4th layer). We can call this kind of cloud as SCS Cloud, and it supports Team Works automatically.

The End

==========

I made the first step, and I cannot make the next steps by myself. So the last purpose to write this document is to share the idea to any person or any company. I don't believe I can implement the SCS Cloud alone, but I do believe WE can. I am glad to join any company or open source organization to do this.

Besides ComEgg/MagicEgg, RDS, and some Chinese documents since 2004, recently I just finished Gongfuxin 0.5.9. I hope I can put Gongfuxin into SCS Cloud later when Gongfuxin grows up.