The Design and Construction of Architecture Description Language for Distributed Software

Yaghoub Karimilivari

***Abstract*—The aim of this paper is to design and creation of architecture description language to distributed software systems. Nowadays, distributed system is one of the very important subjects in the field of computer systems on which a lot of research is done. Hence, this paper tries to describe automatically distributed software architecture. In this method, first a sequential program is clustered and then the equivalent distributed architecture of this program on the base of suggested language is described. In order to describe distributed software architecture we need some necessary information which should be extracted in the form of short code. Using this short code which is obtained in the previous phase, distributed software architecture for sequential program is extracted. The name of the suggested architecture description language is "EAJAVA".**

***Index Terms*—Architecture description, connector, distributed software, component.**

I. Introduction

The aim is to present an automatic architecture description language for distributable code from a serial code. This new description language is created by our studies on two languages, C2 and Archjava. In designing and creation of this language Archjava is used as a base language. The most important section of this paper is to determine synchronization points. Synchronization points occur when after clustering a sequential program there can be found some requests which both caller method and called method are put in different clusters[1]. As these clusters are located in different components and are performed in a parallel form, so all the components in caller code which need response of request should be distinguished and should expect response result. In order to describe distributed architecture the necessary information which helps us to replace the sequential program to distributed one should be available and should be saved in a file. In all the ADL architecture description consist of component description, connector and topology.

II. How to Describe Software Architecture

In distributed software systems like other ADL, description of component, connectors and topology will be found. Below they are discussed.

A. Component Description

By means of the obtained code which consists of the necessary information to describe distributed architecture, and as the program is in a clustering form, every cluster will be described by a component. In this description ports, classes, service mapping and synchronization points should be described. Every port in a component indicates the connection between other components[2], [3].

Meanwhile every component only by means of its ports can be connected to other component[4]. As a result in order to distinguish the ports of a component, all requests occurred between the mentioned components and the next one should be considered. One component in case of being connected to the other, it is carried on through a port [5]. So all requests taken place between these two components are considered as required services or prepared services in this port. In order for two components to be able to be connected there should be defined a protocol between them. The classes used in clusters, will be mentioned in the consisting clusters and when describing the component only the name of classes will be mentioned. In a component it should be distinguished that the prepared service should be carried out by which classes, in other words there should be a mapping between the classes and the services [6].

B. Connector Description

In this language, description of connectors will be based on their type. In connectors the mechanism of communication is based on a protocol by which the components are connected to each other [6]. Every connector will be able to connect two components to each other.

C. Topology Description

In order to describe distributed architecture topology samples of component and connectors are created and then the way of communication among them are distinguished.

III. Sugested Architecture Description Language for Distributed Software

A. Component Description

In this language component description is based on the description of functionality of component or their type. Different samples of every component can be used in architecture. Modelling of component type can help us to reuse components and enables us to analyse architecture[7]. In description of component type the existing ports and classes in a component should be described and the mapping of services and synchronization points of components should be determine. The construct of component description is writing below:

Manuscript received October 18, 2012; revised November 24, 2012.
Yaghoub Karimilivari is with Islamic Azad University of Khosroushahv (e-mail: karimi@karimiazari.com)

Component_Description::=component class
component_name '{'
Port_ Description
Classes_ Description
Services_Mapping
Synchronization_point
'}'

1) Port description: Using existing ports in components we can analyse the communication between architecture components. Every port in architecture describes an interface from a component which distinguishes a set of communication points of component with environment. Every interface distinguished required and provided services by components[7]. By required services we mean expectable interaction of component in environment and by provided services we mean expectable interaction of environment from component. These services decrease the dependence among components. Port description construct will be written below:

Port_Description::=Port Port_name'{'
Requires:
{service_Definition}
Provides: [this]
{service_Definition}
Protocol:
Protocol_Definition
'}'

Service_definition::=[this] return_type service_name
(parameter type

In the above mentioned construct "requires" refers to the required services of component which consists component expected interaction from architecture also "provides" shows the provided services by component which is expected by architecture. In definition of each service the name, type of parameters and type of return should be distinguished. If one service had no return, the keyword "void" will be used. The keyword "this" is used for broadcast services. This word shows that the service will be sent to several ports. As the providing components of these services use unique port and protocol to communicate with other components, so in architecture different samples of this type of ports will be created. "Protocol" refers to a protocol by which a port can make request and return response to environment. This protocol will be in form of semi-regular statement. In this protocol each request and response that is received from environment is shown by the prefix <<?>> and each request and response that is sent to outside of component is shown by the prefix "<<!>>". In order to make difference between request and response, the suffix "<<->>" is used to show response and the suffix "<<+>>" is used to show request. For example if we consider the following statement as a protocol is one of the ports:

?aa+ ; !ca+ ?ca- !aa-

It can be said that first the mentioned component receives the "aa" request and then requests "ca" service. After that, it receives the response of the request of "ca" and finally responses to "aa". The operators that are used this language, are of this type.

";": the set of request or responses which are mentioned consecutively.

"||": the set of request or responses which can be selected alternatively.

"*": shows a repeating interaction.

"()": shows probability.

The requests existing in this protocol are carried out based on synchronization. That is, a component after requesting can go on its way, and expects response only when it needs the response of its request.

2) Class definition: In definition the classes of component in component description only the name of the classes will be mentioned and as our discussion mostly is on the relationship among components and not on internal construct, then existing classes in every component are not considered as sub-component. The definition of a class will be as following

Classes_definition::=classes:class_name{ class-name}

3) Service mapping: This part aims to determine that provided services of a component are carried out by which class and which function. When one class in a component requests functions outside the component it should be distinguished that which ports and by which titles are going to be delivered[8,9]. There fore, the requests that from external environment of component to the existing classes of the component one done, after receiving by port will call a method of a class which can respond to that request[10]. It is obvious that the requests done by component classes will be available in the ports. BY means of this operation, the component located in its environment will be dependent. As there operates an interface port for providing required services of a component from external environment. The following diagram shows service mapping description of a component.

Services_Mapping::=
Rservice:
{class_name.required_service_name (parameter_type) to port_name }
Pservice:
{prot_name.Provided_service_name(parameter_type) to class_name

The keyword "Rservice" refers to the matching of the required called services inside the component and the port from which responds.

4) Synchronization points: We considered synchronization point, because in replacing sequential programs to distributed programs caller and called methods may exist in different components. The components should go on running in a parallel form. So that, when a component calls a method which is located on other components, the caller can go on his operation until the program progresses the lines which contain the return. Therefore in description of synchronization points, the name of class, the name of function and line numbers which the response of the request is need, should be mentioned. Also it happens that return result of function affects one or several variables, so these variables should be mentioned in the description, too. The way in which we describe synchronization points in component is written below.

Synchronization_point::=synchronization point:sync_point
{sync_point}

Sync_point::=wait '@line_number' ' ' variable_name'
' 'class_name

B. Connector Description

In this language each connector connects two or more ports to each other and only two properties of them are discussed by this language. The first property is called connect pattern; That is, the way that the components are connected to each other by means of connectors. The second property is call acceptability control among components. This control is based on the defined protocol [11]. The connectors are described as below:

Connect_decl::=Connector connector_name '{'

Connect pattern: 'CORBA'|'RMI'

Control protocol: 'yes' |'no'

'}'

C. Topology Description

In topology description some samples of components to each other is distinguished. In this description in order to create samples of components or connectors we use the keyword "prod". The keyword "bind " helps us to show connection of ports to each other, too. Using this keyword illuminates that which port of which component is connected to which port of which other component through which connector. The way in which we describe topology is written below:

Topology_decl::=topology architecture_name '{'
Component_production{component_production}
Connector_ production{connector_ production}
Connection_definition{connector_definition}
Component_ production::=prod component component_type component name
Connector_ production::=prod connector connector_type connector_name
Connector_definitinon::=bind component_name
' 'Component_name ' ' connector_name
'}'

IV. Example

As an example, a component of the following sequential program by three existing classes are considered, architecture description of his program in distributed form, based on the suggested language will be done. In order to describe distributed architecture, each cluster will be considered as a component, and we suppose that each cluster is equivalent to programs which separately are written in different classes (each component may have more than one class).

```
Class class1 {
……………
Public void funa()
{
  …………...

Class2 b=new class2();
……………
St=b.funb(j) // line number(10)
        …………
           }

  Void funa1()
     {
        …………
     }
   ………………
}
Class Class2{
   ……….
Public string funb(int i)
{
    ….
Class3 c=new class3();
a=c.func(m); //line number(10)
    …..
}
   …...
}

class class3{
         ……………..
          public integer func(int r)
                                        {
                        …………….

                class2 a=new class2();
                b=a.funb(k); // line number (22)
                                 …..
                                    }
…………
}
```

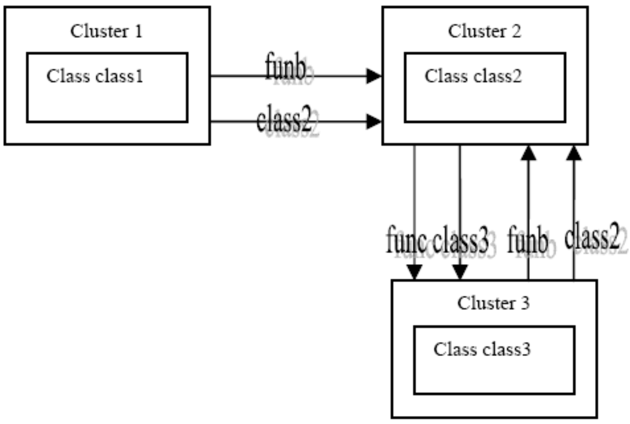

Fig. 1. Components interaction

A. Distributed Architecture Description Language for Fig. 1

By these assumptions architecture description will be as following:

Component C1{
Prot Pc1_c3
Requires:
String funb[class2 , int]
Class2 new class();

```
Classes:
Class1;
Rservice:
Class1.funb[class2 ,int] to Pc1_c3
Class1. new class( ) to Pc1_c3
Synch point:
Wait @11 st, class1;
Component C2{
Port PC2-C3
Requires:
String funb[class2 ,int];
Class2 new class2( );
Provides:
Integer func[class3 ,int]
Classes:
Class3;
Rservice:
Class3.funb[class2,int] to PC2-C3;
Class3.new class2( );
Pservice:
Pc2-C3.func[class3,int] to class3;
PC2-C3.new class3( ) to class3;
Synch point
Wait @ 23 b,class3;
Component C3{
Port PC3-C2
Requires:
Integer func[class3,int];
Class3 new class3( );
Provides:
String funb[class2,int];
Class2 new class2( );
Port Pc3-C1
Provides:
String funb[class2,int];
Class2 new class2();
Classes:
Class2;
Rservice:
Class2.func[class3 ,int] to PC3-C2;
Calss2.new class( ) to PC3-C2;
Pservice:
PC3-C2.funb[class2,int] to class2;
PC3-C2.new Class2( ) to class2;
PC3-C1.funb[class2,int] to class2;
PC3-C1.new Class2( ) to class2;
Synch point
Wait @31 a,class2;
```

V. Conclusions

In this paper there created a language by which is obtained form a prepared serial code. Therefore in order for that the Archjava language was used as the basic language. Also, in order to describe distributed architecture we considered each cluster as a component. We discussed three basic elements, component, connector and topology. It will be mentioned that determining synchronization points is the basic port of the paper that were designed and on different samples were implemented.

References

[1] A. Tang, J. Han, and R. Vasa, "Software Architecture Design Reasoning: A Case for Improved Methodology Support," *IEEE Software*, pp. 43-49, 2009.

[2] C Zannier, M. Chiasson, and F. Maurer, "A model of design decision making based on empirical results of interviews with software designers," *Information and Software Technology*, vol. 49, no. 6, pp. 637-653, 2007.

[3] A. Amirat and M. Oussalah, "Enhanced Connectors to Support Hierarchical Dependencies in Software Architecture," in *Proc. of 5th NOTERE'08 International Conference on New Technologies in Distributed Systems*, Lyon, France, vol. 1, pp. 252-261, June 23-27, 2008.

[4] A. Smeda, M. Oussalah, and T. Khammaci, "A Multi-Paradigm Approach to Describe Complex Software System," *WSEAS Transactions on Computers*, no. 4, vol. 3, pp. 936-941, October 2004.

[5] M. Oussalah, A. Smeda, and T. Khammaci, "An explicit definition of connectors for component based software architecture," in *Proceedings of the 11th IEEE Conference Engineering of Computer Based Systems*, Czech Republic, 2004

[6] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Littl, R. Nord, and J. Stafford, Documenting Software Architectures: Views and Beyond. Boston, MA, Addison-Wesley, 2002.

[7] B. Scholten, MES Guide for Executives: Why and How to Select, Implement, and Maintain a Manufacturing Execution System. ISA, Durham, 2009.

[8] M. Ricken and B. V. Heuser, "Engineering von Manufacturing Execution Systems. SPS/IPC/Drives Kongress, Nürnberg, 2009.

[9] Y. Garlan, S. Aldrich, and K. Discotect, "A system for discovering architectures from running systems," in *ICSE*, pp. 470–479, 2004.

[10] Vasconcelos and Werner, "Software architecture recovery based on dynamic analysis," in *Proc. of 18th Brazilian Symp. On softw. Eng.*, 2004.