Software Components Assembly with an Appreciated QoS

Abdallah CHOUARFIA and Mebarka YAHLALI
Computer Science Department, USTO-MB University
Oran Algeria
Chouarfia@univ-usto.dz , Yahlai@univ-usto.dz

A*bstract* – The objective of CBSE (Component-Based Software Engineering) is the development of big software by integrating of existing components. The traditional concept of applications development by writing code was replaced by the assembly of prefabricated components. The goal of the assembly is to reach a coherent application from a set of software components. We present in this article a method enabling the evaluation of the quality of software components assembly. This method allows us choosing the best components' composition in order to obtain the system required by the user in term of quality (non-functional needs).

***Index Terms* – Software Components, Assembly, Quality model, Quality of assembly, quality factors, quality criteria, quality metrics.**

I. INTRODUCTION

The components approach is relatively recent in the history of software engineering, it appeared around the middle of the nineties in reply to the limits of object-oriented design approach. It introduced a new method for the design of software applications. This method called CBSE (Component-Based Software Engineering), it is based on the composition (assembly) of prefabricated software entities, latter called components. This composition is carried out by connecting the components' interfaces to provide services, clients request a service via its interfaces.

The component composition is currently syntactic and so it poses many problems such as the quality evaluation of assembly to choose the best composition.

This article presents a quality evaluation method of software components assembly. To do so, the section 2 presents an outline on software components, section 3 synthesizes the software components assembly and section 4 relates to the quality of components assembly, in which we present the quality model used as well as the suggested method of quality evaluation.

II. SOFTWARE COMPONENTS

A. Component definition

Nowadays, there is not a standard definition of what is a software component, even if several component models are today commercial standard and products. However, one of the definitions most often quoted is given by **Szyperski** and **Pfister [1]:**

"A Software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A Software component can be deployed independently and is subject to composition by third parties".

The software components can be classified as follows:

- **Blackbox**: The customer does not know any detail beyond the interfaces and their specifications.
- **Whitebox**: The whitebox implementation is entirely available and can be studied in order to increase its comprehension. We can find in the literature, the term of Glassbox. When the distinction is made, that means that the whitebox allows the implementation handling whereas the Glassbox allows simply the study of implementation.
- **Graybox**: Only a controlled part of the implementation is visible.

B. Component structure

A software component has mainly three elements [2]:

1. **Functional interfaces and configuration properties:** The required functional interfaces must be satisfied when a component instance is created so that this latter can be used through the provided interfaces. The configuration properties allow the configuration of a component authority (for example, change the name of a button).
2. **Control interfaces (provided/required):** are the set of methods which allow managing the component instances' life cycle during the execution**.** These methods are intended to be called by the execution environment of the components model.
3. **Dependences and deployment properties:** the dependences are specific to each implementation of a component. They must be satisfied at the deployment time of a component class to allow its use.

The deployment properties [1] are defined on the implementation level, they are used to configure the common characteristics of instances.

[1] The deployment properties are similar to a variable "class" in object approach

III. SOFTWARE COMPONENTS ASSEMBLY

According to [3], the component assembly presents two aspects:

- Ensemblist aspect of assembly (assembly composition).
- Communication aspect between components (interactions between components).

The dependency relationships between the components' interfaces are (cf. Fig1):

1. Offered services dependencies: indicate the links between visible offered interfaces of the composite as well as interfaces offered by components which are members of the composite. In this case, the offered specification of the composite, made visible, is identical to the offered specification of a component member (delegation principle).
2. Required services dependencies: indicate the links between required interfaces of members of components and the visible interfaces of the composite (delegation principle).
3. Inter-component dependencies: which are links of assembly between required interfaces and offered ones. In general, there is dependency relationship between offered services and required ones.
4. Intra-component dependencies: which are created to support the implementation dependencies between offered interfaces and required ones of a component.

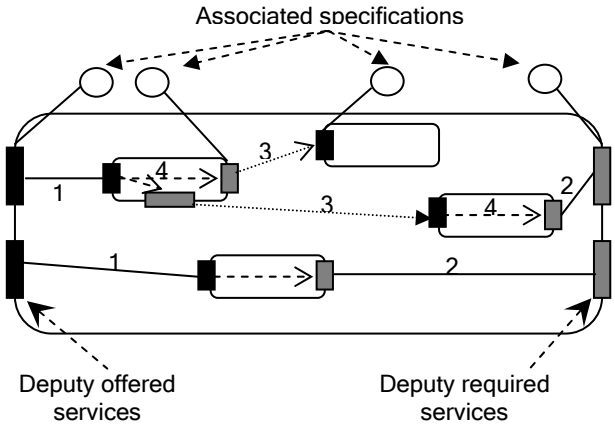

Figure 1: Dependence relationships between the components' interfaces

IV. SOFTWARE COMPONENTS ASSEMBLY QUALITY

Provided quality is the principal concern of the user. It does not have a consensual definition which would be appropriate for all the fields to which it applies.

Some is the context, quality covers with the non functional properties of a data-processing entity. The level of quality is characterized by the particular "values" of the non functional properties.

A. Factors, criteria and metric of quality

- **Factor:** Characteristic of the software which contributes to its quality [4]. It relates to the use characteristics linked to:
  - Exploitation Environment.
  - Monitoring and Maintenance Environment.

The factors translate the **external** vision [5].

- **Criteria:** Attribute of the software, a factor can be evaluated via this attribute [4]. They are :
  - Oriented developer.
  - Components of quality factors.
  - Linked to metrics.

The quality criteria concern the characteristics of use according to the intern vision (software structure) [5].

- **Metric:** The quality criteria are connected to metrics which are a posteriori measurements.

There are three types of metrics to measure the attributes [6]:

1. **Presence**: This metric identifies, that an attribute is present in a component or not. It is described by "Boolean" type.
2. **IValues**: This metric is used to indicate the exact values. It is described by "Integer" type.
3. **Ratio**: This metric is used to describe percentages.

B. Component quality model

Table 1 shows the component quality model classified into two classes:

- The quality characteristics which can be observed at runtime.
- The quality characteristics which can be observed during the life cycle.

TABLE 1. COMPONENT QUALITY MODEL

Characteristics	Sub-Characteristics (Runtime)	Sub-Characteristics (life Cycle)
Functionality	Accuracy Security	Suitability Interoperability Conformity Individual contents
Reliability	Faults Tolerance Recoverability	Maturity
Usability	Configurability	Understability Operability
Efficiency	Time Behavior Resource Behavior Scalability	

- **Functionality:** This characteristic expresses the ability of a component to provide required services,
- **Reliability:** This characteristic expresses the ability of a component to maintain a specified level of performance,
- **Usability:** This characteristic expresses the ability of a component to be understood, learned, used, configured, and executed,
- **Efficiency :** This characteristic expresses the ability of a component to provide appropriate

performance, relative to the amount of resources used,

- **Maintainability:** This characteristic describes the ability of a component to be modified,
- **Portability:** This characteristic is describes as the ability of a component to be transferred from one environment to another,

C. Components assembly quality

C.1. Quality specification of software component

Currently, there is no work concerning the quality specification of software components. What exists relates to QoS.

QML language makes it possible to describe the QoS constraints of software components, but it is not possible to specify what the service can be really provide. To fill this lack, we present in what follows a syntax for specification of software components quality. The advantage of the latter is that it can be used with any quality model.

Syntax Description[8]: Figure 2 presents suggested syntax. The quality definition <qual-defi> consists of a sequence of characteristics definition <carac-defi>, a characteristic is presented by an identifier <ident> which can have a numerical value or be presented by a set of sub-characteristics <subs-carcs>. In the same, a sub-characteristic can have a numerical value or be presented by a sequence attributes form <attributes>, an attribute <attribut1> is presented by a numerical value.

```
<definition>::"DEFINE_QUALITY " <qual-defi>'.' <error>   1
<qual-defi> ::<qual-defi> ',' <carac-defi >   2
                | <carac-defi >                3
<carac-defi>  :: <ident> '='<subs-caracs>     4
<subs-carcs>::<num-val>                       5
                | '{' <subs-carac> '}'        6
<subs-carac>::<subs-carac> ',' <sub-carac>    7
                | <sub-carac>                 8
<sub-carac> ::<ident> '=' <attributs>         9
<attributs> ::<num-val>                      10
                | '{' <attribut> '}'         11
<attribut> ::<attribut> ',' <attribut1>      12
                | <attribut1>                13
<attribut1>::<ident> '=' <val-num>           14
< val-num>::<num> ',' <num>                  15
< num>::<chiffre>+                           16
< ident> :: <lettre>(<lettre>| <chiffre>)*   17
<lettre> ::  a | b | c | . . . | z | A | B| C| . . . | Z  18
<error> :: 'ERROR =' <val-num>               19
<chiffre>::0| 1 | 2 | 3 | . . . | 9           20
```

Figure 2: Quality Specifiication Syntaxe

Exemple :

DEFINE-QUALITY
Reliability = {
Fault-Tolerance ={ Mechanism availability =0,6 ,
Mechanism Efficiency =0,4}
},
Usability = {
Configurability ={ Configuration Effort =0,3 },
}.
Error=0,01

C.2. Quality of components assembly

Let us assume that:

- S: the system requested by the user, S can be built by assembling the software components.
- F: a set of non-functional characteristics (Factors) fixed by the user.

$$F= \{f_i /i=1. .N\}$$

- A: a set of attributes relating to F.

$$A= \{a_j/j=1.. m\}$$

The system S is the result of software components assembly taking into account the quality characteristics. It can be represented as follows:

$$S=\sum c_{i/i=1. ..l} /F$$

Such as: $\sum$ represents the assembly symbol. c_i: components

- SR: a set of the real systems SR_k resulting from the various compositions of existing components.
- FR_k: a set of characteristics provided by SR_k.
- AR_k: a set of attributes relating to FR_k.

$$AR_k=\{ar_{kj}/j=1..m\}$$

Each component c_i has a set of non-functional parameters, which are represented by he attributes vector: Vc_i

$$Vc_i= (v_{i1}, v_{i2}, v_{i3}, ..., v_{im})$$

Where v_{ij} represent the quality attribute j of the component i.

It can exist a case where several software components compositions (several SR) lead to the functional needs for the desired system S [7] however the goal is to choose the best system in term of quality

The steps of the proposed method [8]:

Step 1:

Consists in fixing the value of desired quality (Qd) by specifying:

- Various criteria and attributes of quality.
- Importance of each criteria (and attribute) compared with another.

Step2:

Construction of the application by assembling the suitable software components (to determine all possible compositions (all $SR_{k/\ k=1.\ .g}$)) ,g being the number of possible compositions

Step 3:

Evaluate the provided quality (Qr) by each system then compare it with the desired one.

Step 4:

Choice of the optimal application:

Each real application (SR_K) has α_K / α_K= |Qd-Qr_K|, where Qr_K is the quality provided by the system SR_K. SR_K is the optimal application if α_K= Min |Qd-Qr_K|

The problem is articulated around two axes which are:

1. The evaluation of provided quality.
2. Production of the total value of quality.

C.2.1. QuaLITY EVALUATION

The real properties of system SR (FR elements) can be derived from the properties of its components c_i. Thus an attribute ar_j of a vector AR can be represented as follows:

$$ar_J = \vartheta\ v_{ij/i=1..l}$$

Where ϑ is evaluation function of the quality (additive rules, concave rules…). If an attribute a_J does not exist in the quality attributes vector of the component c_i (Vc_i). A Null value is associated to this attribute.

C.2.2. Production of the total quality value

In order to answer the second question, we use ROC (Rank Order Centroides) concept [9]. The centroides of the classification constitute a means to converting rows (1^{st}, 2^{nd}, 3^{ed}) into notes or weightings which are numerical values.

If n is the number of attributes, the weighting of the attribute K is:

$$(\sum_{i=k}^{n} (1/i))/n$$

i.e. the production of the total quality of a software system implies:

1. The classification of the criteria (and the attributes) according to their importance.
2. The conversion of the rows into weighting using ROC.

Example

Let:

- S a software system.
- The performance and the accuracy are the factors of this system.
- It is considered that the total accuracy is the most important factor, and total performances are the least important factors.

In this case, weightings are calculated as follows:

Accuracy: w1= (1+1/2) /2=0.7500

Performance: w2= (0+1/2) /2=0.2500

Now, we calculate weightings for each attributes set concerning these two factors.

Supposing that:

- The three attributes of performance are:

1. Research performances.
2. Starting performances.
3. Recording performances.

- The research performances are the most important.

Weightings are then presenting as follows:

Research performances:

$$w1 = (1 + 1/2 + 1/3)/3 = 0.6111$$

Starting performances:

$$w2 = (0 + 1/2 + 1/3)/3 = 0.2778$$

Recording performances:

$$w3 = (0 + 0 + 1/3)/3 = 0.1111$$

- The two attributes of accuracy are:

1. Global accuracy.
2. First-screen accuracy.

- We decide that global accuracy is more important than the first-screen one.

These weightings are calculated as follows:

Global accuracy: w1 = (1 + 1/2) /2 = 0.7500

First-screen accuracy: w2 = (0 + 1/2) /2 = 0.2500

Factors	Attributes	ar_j values	Factors Values
Performance **(0.2500)**	Starting Performances **(0.2778)**	.4567	(0.2778) (0.4567) + (0.6111) (0.2567) + (0.1111) (0.4567)= **0.3045**
	Research performances **(0.6111)**	.2567	
	Recording Performances **(0.1111)**	.4567	
Exactitude **(0.7500)**	Top-result accuracy **(0.7500)**	.2567	(0.7500) (0.2567) + (0.2500) (0.0900) = **0.2150**
	First-screen accuracy **(0.2500)**	.0900	
Total quality			(0.2500) (0.3045) + (0.7500) (0.2150) = **0.2374**

V. CONCLUSION

This article presents a quality evaluation method of components assembly. The problems are articulated around two axes, on the one hand how to evaluate the provided quality, on the other hand how to produce the total quality value.

ROC concept is used to produce the total quality value. This concept aims at the comparison of the software systems by producing a single numerical value representing total quality.

We used the quality model suggested in [6], this latter is based on ISO 9126 with some adaptations for the components.

Considering the importance and the volume of work, we limited the evaluation of quality to a component.

REFERENCES

[1] C.Szyperski and C. Pfister, “COP’96 Workshop Repor”, June 1996, International Workshop on Component-Oriented Programming, University of Linz, Austria.

[2] Humberto CERVANTES, “Towards a components directed services model to support the availability dynamic“, Doctoral Thesis, 29 March 2004, University of Joseph Fourier, GrenoubleI.

[3] ACCORD project, “Abstract model of components assembly by contracts”, Deliverable 1.1-4, June 2003.,France.

[4] A.Beugnard, “Introduction to the software genius # 3”, course support, 1998, ENST Bretagne.

[5] C. Million-Rousseau,”Test, quality and maintenance”, Course, 2006, University Polytechnic school , University of Savoie.

[6] Alexandre.A and Eduardo.S and Silivio.R, “Quality Attributes for a Component Quality Model”, 2006, 10th International Workshop on Component Oriented Programming (WCOP), Scotland.

[7] Bart.G.A and Rogis.F and Salah.S, "Substitution Model for software components", mars 2006, Workshop on software evolution in partnership with LMO 2006, Nimes.
[8] Chouarfia Abdallah, Yahlali Mebarka, "Software Component Qualité" , January 4-6,2008, 9th International Business Information Management Association Conférence Marrakech, Morroco.
[9] Stillwell, W.G, D.A.Seaver, and W. Edwards, "A comparison of Weight Approximation Techniques in Multi-attribute Utility Decision Making.",1991, 62-77.