Toward a Conceptual Comparison Framework between CBSE and SOSE

Anthony Hock-koon and Mourad Oussalah

University of Nantes, LINA
2 rue de la Houssiniere, 44322 NANTES, France
{anthony.hock-koon,mourad.oussalah}@univ-nantes.fr

Abstract. In this paper, we discuss the theoretical differences between component-based and service-oriented software engineering (CBSE and SOSE). We present a conceptual comparison framework which confronts their quantitative and qualitative aspects and provides a better understanding of their use. This comparison takes the object orientation (OO) into account to illustrate changing concerns of software engineering between object, component and service.

Key words: SOSE, CBSE, OO, Comparison Framework, Quality Measurement

1 Introduction

Component-based software engineering (CBSE) [1] proposes to reuse existing software entities, called components, to build new applications. Service-oriented software engineering (SOSE) [2, 3] proposes to reuse provided capabilities of existing software entities called services. Both of them have the reusability as theoretical root and rely on the concept of software architecture [4] to describe and manage collaborative software entities. They use numerous similar concepts, approaches, and technologies. Meanwhile, they have continued with their development tracks in parallel and focused on their specific interests. Consequently, there is a mixture of similarities and specialized concepts.

All comparison works between CBSE and SOSE have a bottom-up approach in which they focus on specific technologies to identify the resulted software qualities [5–7] (e.g. comparison of performance between technologies [8]). However, they do not allow a direct comparison at a conceptual level which can offer a global understanding of the differences between the paradigms. To our knowledge, only one other work [9] has a top-down approach which focuses on this conceptual comparison. However, it only tackles some confusions of vocabularies and does not analyze the consequences of these differences on the quality of products and production processes.

In this paper, we propose a conceptual comparison framework which can deduce the resulted software qualities. It is divided into quantitative and qualitative aspects. It takes object orientation (OO) into account to provide a global point of view about the evolution of concerns between object, component and service.

2 Quantitative aspects

The top-level categories of our quantitative part are the product and the process. It does not intend to be exhaustive, but it aims at listing the core concepts of each paradigm to underline their theoretical stance.

2.1 Product and Process

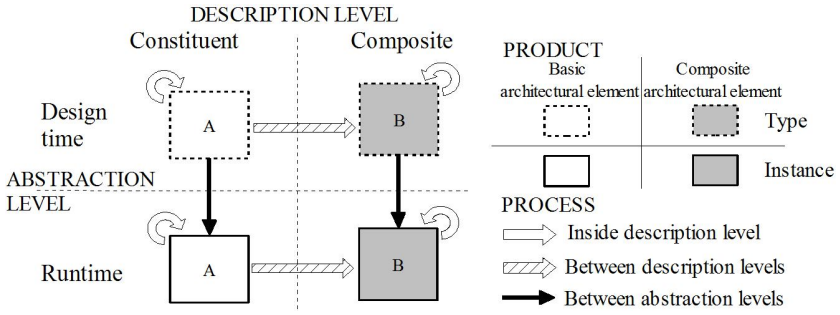

Fig. 1. Abstraction levels and Description levels

A *product* is a software entity or a conceptual entity which is the result of an action or a process. A *process* is an action or a succession of actions which is used to create or modify a product and obtain a specific one.

The product category is divided into two sub categories (Figure 1):

- **Basic architectural element** - basic building blocks of each paradigm;
- **Composite architectural element** - complex products built from existing architectural elements. Their structure clearly identifies the reused architectural elements and their relationships.

Each sub categories is also divided into two groups according to two *abstraction levels*: the *design-time* and the *runtime* (Figure 1).

The process category focuses on the reusability principle shared by OO, CBSE and SOSE, i.e how to reuse existing software entities, the *constituent*, to build new ones, the *composite*. A constituent is a basic or a composite architectural element. These notions of constituent and composite define two *description levels* (Figure 1). The process category is divided into three sub categories according to abstraction and description levels.

- **Inside description level** - gathers processes which target products from the same description level (Figure 1 white arrows) at the two abstraction levels;

- **Between description levels** - gathers processes which target a product from a different description level than the produced one (Figure 1 hatched arrows). This category is divided into design-time and runtime;
- **Between abstraction levels** - gathers processes which target a product from the design-time and then produce a product of the runtime (Figure 1 black arrows).

2.2 Comparison between OO, CBSE and SOSE

We want to emphasize two main differences illustrated by the Table 1. First, SOSE relies on the dynamic service provisioning (discovery and selection) between abstraction levels to produce a concrete service from an abstract service. On the contrary, OO and CBSE rely on the instantiation process between class and object, and component type and component. Then, SOSE relies on additional runtime processes to support the self-adaptation of composite elements. Even if some similar processes are proposed by CBSE, they are not required to specify a component model.

3 Qualitative aspects

Existing works about software quality [10, 11] introduce a huge number of quality criteria based on point of views (developer, user and so forth) or application domains. Our approach is different and proposes a set of core features which are used to express every quality criteria. Users choose a quality criterion they desire to evaluate. Then, they define this quality by combining the measurement of each core feature. This combination is what we call a *qualitative perspective*.

3.1 Core features

We identify six main features:

Loose coupling - measures the dependencies between entities.

Expressiveness - based on the number of concepts and processes provided by the paradigm to specify and manipulate its products.

Abstraction of communication - ability of a paradigm to abstract the communication layer which drives the execution of the application.

Explicit architecture - ability of a paradigm to provide a clear architectural view of the application which follows its principles.

Evolutionary ability - ability of a paradigm to provide a powerful set of concepts and processes to evolve its products.

Ownership - allocation of responsibilities (development, QoS, maintenance, deployment, execution, management, use) among the provider of the reused products and its clients. It expresses the level of liberty granted by the provider to the client.

Table 1. Product-Process: Comparing Object, Component and Service

Product		**OBJECT**	**COMPONENT**	**SERVICE**
Basic architectural elements	**Design-time**	Class	Component type, Connector type	Abstract service
	Runtime	Object	Component, Connector	Concrete Service, Service description
Composite architectural elements	**Design-time**	Composite class	Configuration type, Composite component type	Composition schema type, Composite service type
	Runtime	Composite object	Configuration, Composite component	Composition schema instance, Composite service instance, Composite service description
Process		**OBJECT**	**COMPONENT**	**SERVICE**
Inside description level	**Design-time**	Association Inheritance	Horizontal composition, Interface inheritance Versioning, Refinement	Choreograph, *Inheritance of composition schema type*
	Runtime	Method call	Functionality call,	Choreography, Service provisioning, Service invocation, Service publication, Self-adaptation of composite architectural element
Between description levels	**Design-time**	Composition	Vertical composition	Orchestration,
	Runtime	Method call	Functionality call, Delegation	Orchestration, Service invocation, Service provisioning, *Composition of service descriptions*
Between abstraction levels		Instantiation	Instantiation	Service provisioning, Instantiation of composition schema

3.2 Comparing Object, Component and Service

Figure 2 shows the classification of OO, CBSE and SOSE according to our six features and three levels of importance (low, medium and high). It illustrates an instance of our qualitative comparison. These levels are not a precise measurement but they are used to define a hierarchy between paradigms based on our analyze.

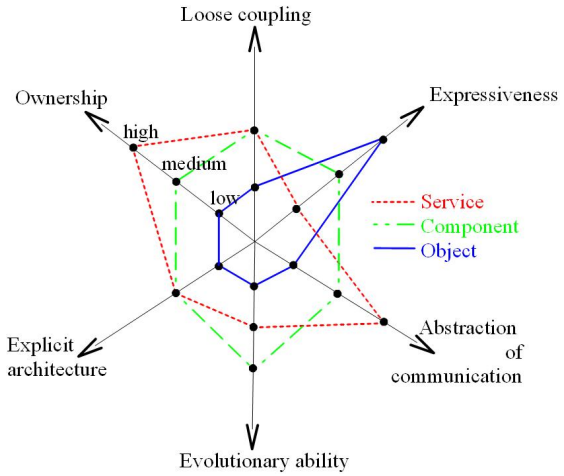

Fig. 2. Comparison of the features

Loose coupling - typically, an Object-based system is built from a set of classes which are tightly coupled while a Component-based or a Service-based system intends to be more loosely coupled. In fact, related topics such as the self-adaptation or the management of heterogeneities are deeply studied by CBSE and SOSE. We set that existing researches reach the same level of maturity. However, our previous work [12] on the definition of the loose coupling notion shows that numerous challenges are still unsolved.

Expressiveness - OO manipulates a huge number of concepts such as granularity, reflexion, template, inheritance, abstraction level, description level and so forth. CBSE's expressiveness mainly relies on OO's researches. However, some advanced concepts such as reflexion or inheritance and polymorphism do not reach the same level of maturity. SOSE has the weaker expressiveness of the three. In fact, it shares the lack of CBSE and adds some others (e.g. the abstraction levels and the distinction between type and instance are still unclear).

Abstraction of communication - SOSE provides the best abstraction of communication. In fact, the global collaboration pattern between constituent services is located inside a single entity: the composition schema which expresses the overall behavior in terms of workflows and dataflows. In CBSE, the communications are located inside the connectors which split the global behavior.

The workflow is not explicit. Therefore, the overall collaboration is harder to understand and manipulate. In OO, the fine granularity of the class and the association link accentuates this drawback of CBSE.

Explicit architecture - typically, an Object-based system lacks an explicit architecture which is easily understandable. CBSE was first introduced to enhance this aspect and develop the concept of software architecture. SOSE directly uses this experience and its difference with CBSE is not significant.

Evolutionary ability - dependent on the architectural graph and the evolution processes which target its nodes, edges, or the overall graph. Typically, OO does not provide an explicit architecture and thus, its community only focuses on the evolution of the nodes and edges. Both CBSE and SOSE handle an explicit architecture. Their communities also study the evolution of the overall graph. However, some works of the CBSE community such as [13, 14] go further and study the evolution process at the meta and meta-meta-architecture levels.

Ownership - SOSE has taken the concept of ownership to the extreme and thus, the provider of services is responsible for the development, the quality of service, the maintenance, the deployment, the execution, the management. On the contrary, CBSE splits the responsibilities at the deployment level. In fact, the client is responsible for the instantiation of the component inside his application and its execution and management. Typically, the object orientation defines the class as a glass box entity and provides some powerful processes to easily manipulate it.

3.3 Qualitative perspectives

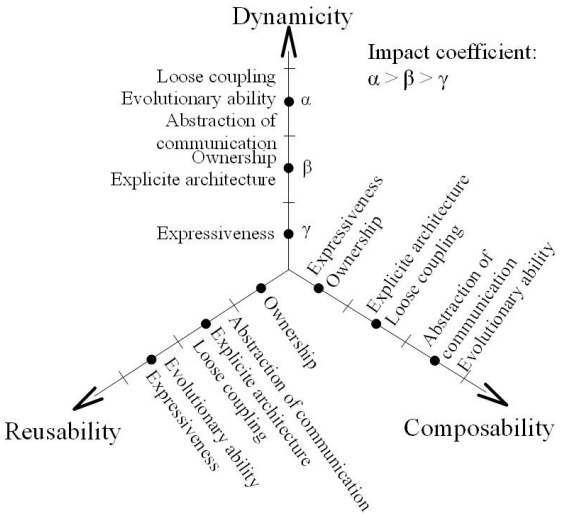

Fig. 3. Quality definition

Our six features represent the main elements of a software development paradigm which impact on the software quality. However, the importance of their impact can vary according to the chosen quality and the user's point of view about this quality. In the figure 3, we act as users and define three quality criteria: reusability, composability and dynamicity. We express our understanding about these qualities and divide the features into three groups, from the γ group which has the weaker impact to the α group which has the stronger impact.

Then, we define a set of formula (e.g. for the reusability(1)) which combine our vision of these qualities dropped to our six features and the previous classification of the three paradigms following these features (Figure 2). Each level (low, medium and high, Figure 3) is associated with a weight (respectively 1, 2 and 3).

$$\begin{aligned} \textbf{Reusability} = & Object : \alpha(4) + \beta(3) + \gamma(1) \\ & Component : \alpha(5) + \beta(6) + \gamma(2) \\ & Service : \alpha(3) + \beta(7) + \gamma(3) \end{aligned} \tag{1}$$

From our qualitative perspective, CBSE has a better reusability than SOSE or OO. The same work can be done for other quality criteria (flexibility, robustness and so forth). The definition of the impacts of each feature on each quality depends on the user's expertise. In fact, each quality represents a particular perspective on our six features and this perspective has to be defined by the user. The following formula $Quality = f(\alpha, \beta, \gamma, \delta, \epsilon, \zeta)$ emphasizes the user role which has to provide:

- the different coefficients (α to ζ) which define the respective importance of each feature according to the chosen quality perspective;
- the function f which defines the way to combine these features.

4 Conclusion

This paper presents a conceptual comparison between CBSE and SOSE divided into quantitative aspects and qualitative aspects. The quantitative aspects classify products and processes which are used to develop some new applications. The qualitative aspects compare the three paradigms following six features which are reused to specified any software quality such as reusability, composability and dynamicity. We show how a user can exploit these features and combine them to evaluate some quality criteria according to their own expertise.

For now, measures of each feature presented in Figure 3 only confront the three paradigms at a conceptual level to provide a hierarchy between them. This hierarchy is based on our own expertise. This approach is sufficient for a direct comparison between theories at a paradigm level, however it is not precise enough

to descend to implementation and technological level. In [12], we propose a new definition of the loose coupling which comes along with an objective evaluation formula. Therefore, the same work needs to be done for the five other features. A better measurement of each feature will ensure a better evaluation of the qualitative perspectives defined by users.

References

1. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Professional (2002) isbn 0201745720.
2. Stojanovic, Z., Dahanayake, A.: Service-oriented Software System Engineering Challenges And Practices. IGI Publishing, Hershey, PA, USA (2005) isbn 1591404274.
3. OASIS: Reference architecture for service oriented architecture 1.0. (April 2008) http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf.
4. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Trans. Software Eng. **26**(1) (2000) 70–93
5. Masek, K., Hnetynka, P., Bures, T.: Bridging the component-based and service-oriented worlds. In: Euromicro Conference on Software Engineering and Advanced Applications, SEAA. (2009) 47–54
6. Stojanovic, Z.: A Method for Component-based and Service-Oriented Software Systems Engineering. PhD thesis (2005) Delft University of Technology, isbn 90-9019100-3.
7. Vassilopoulos, D., Pilioura, T., Tsalgatidou, A.: Distributed technologies corba, enterprise javabeans, web services — a comparative presentation. In: Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. (2006) 280–284
8. Kim, S., Han, S.Y.: Performance comparison of dcom, corba and web service. In: PDPTA. (2006) 106–112
9. Breivold, H.P., Larsson, M.: Component-based and service-oriented software engineering: Key concepts and principles. In: Euromicro Conference on Software Engineering and Advanced Applications, SEAA. (2007) 13–20
10. Kithchenham, B., Lawrence, S.: Software quality: The elusive target. IEEE Software **13** (1996) 12–21
11. Bianco, P., Kotermanski, R., Merson, P.: Evaluating a service-oriented architecture. Technical Report Software Engineering Institute Carnegie Mellon (2007) http://www.sei.cmu.edu/reports/07tr015.pdf.
12. Hock-koon, A., Oussalah, M.: Defining metrics for loose coupling evaluation in service composition. In: International Conference on Services Computing, IEEE SCC. (2010) 362–369
13. Goaer, O.L., Tamzalit, D., Oussalah, M., Seriai, A.: Evolution shelf: Reusing evolution expertise within component-based software architectures. In: International Computer Software and Applications Conference, COMPSAC. (2008) 311–318
14. Garlan, D., Barnes, J.M., Schmerl, B.R., Celiku, O.: Evolution styles: Foundations and tool support for software architecture evolution. In: Working IEEE/IFIP Conference on Software Architecture, WICSA/ECSA. (2009) 131–140