Service-Oriented Control-Command Components for Designing Complex Systems

Olga Goubali[1(✉)], Abdenour Idir[1], Line Poinel[2], Laurianne Boulhic[1], Djamal Kesraoui[1], and Alain Bignon[1]

[1] SEGULA Technologies, BP 50256, 56602 Lanester Cedex, France
{olga.goubali, laurianne.boulhic, djamal.kesraoui}@segula.fr, idir.abdenour@hotmail.com, bignon.alain@neuf.fr
[2] SEGULA Technologies, 71 rue Henri Gautier, 44550 Montoir-de-Bretagne, France
Line.poinel@segula.fr

Abstract. An important task in designing control-command systems is defining components for two essential parts of the system: command and monitoring. Instead of developing a monolithic executable, designers use reusable blocs, named components, which are saved in a library. Services Oriented Architecture (SOA) was introduced in the design of control-command systems to improve flexibility and reusability [1]. However, this approach does not consider the composition of the services [2], disregarding this important characteristic in the design of control-command systems. In some industrial areas such as ship-building, the component-based approach is typically used since it enables better legibility for the applications; it uses a modular approach based on the system architecture recorded on the Piping and Instrumentation Diagram (P&ID) [3]. Each software component is associated with a unique type of equipment. This approach enables to produce components highly optimized to their functions. The counterpart is that services integrated in each component cannot be reused in designing another component. In fact, part of the component services can also be present in another component (Fig. 1). Including services in component does not optimize functions reusability. In this paper, we propose an approach that facilitate and improve the design of quality components whilst complying with specifications and timelines in a more efficient way; it also reduces efforts required to redesign services provided by these components.

Keywords: Control-command component · SOA · MDE · EUD

1 Introduction

Designing control-command systems includes an important step of components definition. These components are usually stored in a library. A component is an independent unit that is combined with other components to make an application. Therefore, designing a control-command system is made of the components with different views for the command and the supervision/monitoring parts.

M. Kurosu (Ed.): HCII 2019, LNCS 11566, pp. 101–113, 2019.
https://doi.org/10.1007/978-3-030-22646-6_8

Usually components are stored in a library which contains the required elements for creating command programs and control interfaces for the system being designed. The library often contains two types of components: standardized components which are highly specialized and white components. The white component is an empty component which initial function is not identified but that can be configured to be adapted to a specific case, this gives bespoke components.

Each software component is associated to a unique type of equipment. This approach enables to create components highly optimized for their functions. The counterpart is that services integrated in each component cannot be reused in the design of another component. Indeed, a part of the services (functions) of a component can also be present in another component (Fig. 1). Integration of services in each component does not enable to optimize functions reuse during the component design. To solve this issue, a service-oriented components model was proposed by [2]. Nevertheless, this model is not adapted to the design of control-command components.

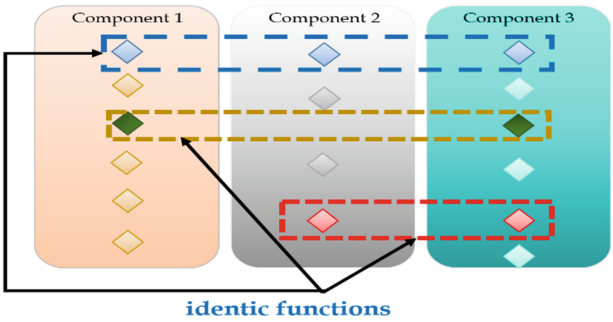

Fig. 1. Duplication functions in component design

Our aim is to improve the approach for components design, to reduce the effort required for re-designing services and to explore solutions to improve reusability during components design. For this, we take advantage of the SOA (Service Oriented Architecture) approach. This approach should enable to encourage quality components design while efficiently complying with the requirements specification and time constraints. We apply our solution to components of an EdS system (Eau douce Sanitaire in French, sanitary freshwater in English). In this system, several components have common functions which are redesigned for each new component.

2 State of the Art

Heterogeneity and fast evolution of applications confront designers with an ever growing complexity [4] and with major challenges of system engineering: scale-up, administration and autonomy. To propose potential solutions to this problem, industrial and academic communities looked towards component-based approaches, which are inescapable since 1990s. These are based on the construction of reusable, modular and

spreadable software components. In the literature, there are several définitions of a "software component", amongst which we select the following:

"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" [5].

A software component is therefore a code unit that can be characterised as a black box. Some component models have been proposed to facilitate design and composition of components, including models of software components.

Defined by OMG (Object Management Group), CORBA (Common Object Request Broker Architecture) component Model (CCM) [6] is a model of allocated software components that specifies distributed and heterogeneous components which are in turn independent of any platform and programming languages. A CORBA component model is defined in IDL3 (Interface Definition Language) language and has different types of ports and attributes that represent configurable properties. CCM enables to model the complete life cycle of a component by proposing a structure to define its behaviour, its integration in an application and its deployment in the CORBA distributed environment. Moreover, CCM provides a global framework for applications design based on distributed components: specification, installation, processing, assembly, deployment and execution of components. However, this model is not easy to implement and does not enable to model adaptive parallel components [7], which execute parallel activities.

To solve this issue of parallelism, CCA model (Common Component Architecture) [8] was defined. Indeed, parallelism does not act on CCA components because they are defined in the same way whether they are arranged in parallel or sequencially. CCA is a software component model written in SIDL (Scientific Interface Definition Language), software interface specification language. A CCA component has two types of ports: ProvidePort and UsesPort that enable, through interfaces, to provide and use functionality of other components. This way, CCA components can be connected to other components through their ports. Each application made of CCA components will be a CCA component itself. However, the use of CCA component requires to know its internal working, which is hardly understandable [4].

A Fractal component model [9] is a consortium project ObjectWeb defined by France Télécom R&D and INRIA. This model is presented as a specification for the design of complex software systems (middleware and operating systems), in different languages such as Java, C, C++, SmallTalk, .Net. Designing such systems requires definition of Fractal components. These are entities with two types of interfaces: server interfaces and client interfaces. Server interfaces correspond to services provided by a component and clients interfaces correspond to services required by the component. A Fractal component contains either its implementation code (primitive component), or other components (composite components). Indeed, Fractal enables to build shared components which can be included in several composites. One advantage of Fractal is that the developer can customize control capacities of each component, while taking into account functional and non-functional aspects. However, developers can quickly be confronted to an increase in programming complexity [4].

Whilst simplifying applications design, existing component models are not adapted to the designing of control-command applications because they do not allow for easy

designing that type of applications. More and more supervision software propose predefined components to assist the designing of control-command systems. However, getting used to handling these components can take a considerable amount of time. It is sometimes easier for designers to develop their components themselves.

3 Problem Description

Designing components enables experts to focus on their expert knowledge where they have most added value [10]. With Component Oriented Programming (COP), designers use reusable blocs that are the components, instead of building a monolithic executable. However, reusability remains at high level (component level). A component used in a given context (subject area, application in the same area, etc.) is unlikely to be reusable in other contexts [11]. As an example, components defined for a sanitary freshwater system are not all reusable for a gasoil processing installation. In fact, some components are standards and other are bespoke, to respond to a requirement that is only relevant for the system to be designed. However, bespoke components are rarely maintainable, because such an approach favors spreading of hacked and incompatible versions of components that are vaguely similar, of which none is really reusable in the end [12]. Moreover, in some industrial areas where additional requirements modifying initial specifications can be added if necessary, inaccessibility to the component code inhibits adaptation of the code to specific needs by developers [13]. Even if models such as SMARTTOOLS [14] do focus on extensibility making it possible to add functionalities to the component as long as they are compatible, it is not possible to use internal services of one component to create another one. This limits the concept of reusability at the component level. Easily creating bespoke components and make them reusable is the first challenge we have identified.

A component designed are often integrated in a library. This library is therefore made of components that can be combined in different ways to respond to different needs. Usually, designing a component starts with the choice of programming language. Once this and the development environment chosen, how the components should be developed still needs to be defined. In the components definition, the COP approach focuses on the individual application rather than looking at a much larger software process. However, components engineering does not only focus on the development of software components but can concern all aspects of software development, from collecting and specifying requirements up to designing and implementing. Most advantageous to reuse artefacts are often not the software components themselves but the knowledge of the field and of the generic designs. Software reuse is more successful if prepared in advance. Until requirements are specified and systems designed, numerous reuse opportunities may have been wasted. It is then interesting to rethink, in applications life cycle, the way components should be designed as well as how to introduce new activities linked to services provided by components.

Search for new ideas led us to look at the definition of a component. A component should provide services that are specific, accurate, defined and implemented during its design. However, some of the services can be present in another component. Encapsulation of services inside a component makes it difficult to reuse them. Issues of

interoperability and reuse at the structural level of the component, and not only during designing of the system, should be addressed. Identifying and modeling reusable services, and implementing them for component designing, is the second challenge we have identified.

4 Methodology

Our solution (Fig. 2) is based on the combination of Service Oriented Computing (SOC), Model Driven Engineering (MDE) and End User Development (EUD) approaches. It allows to capture expert knowledge for designing services oriented components and to facilitate fast integration of evolutions. Applying each of these approaches enables to build a solution to automatically generate a complete chain of control-command components from the requirements specification by the expert user (non-programmer).

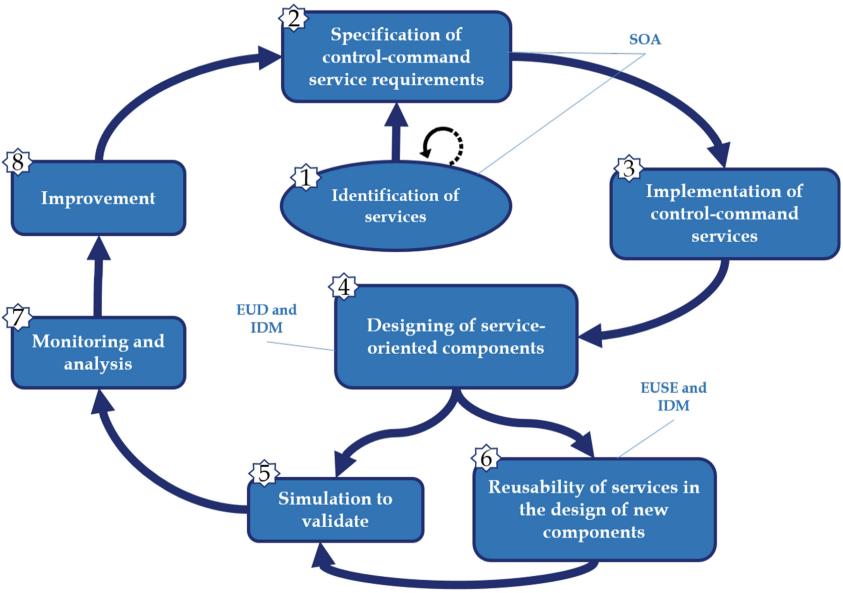

Fig. 2. Our approach

SOC [15] enables to address the limitations identified in the component approach. The aim is not to completely replace the existing component approach, but to reuse some of its principles while adding new ones from SOA. Based on SOA approach, we have identified and modelled control-command services. Identification of services is the same as identifying the most used functionalities of control-command systems, using a user centered approach [16]. HMI centered methods are used to collect business requirements from different actors in the designing process. The aim is to promote a

loose coupling between the different technical functionalities of a component. This first step in our methodology is broken down in several analysis steps. First, we identify the main business actors of the designing of control-command systems, by analyzing different parts of the system. Then, we analyze the operators tasks on the components of control-command systems. This analysis enabled to break down a system in several sub-systems to identify and list the operator actions on the supervision components, as well as the system feedback. We also review existing control-command systems to identify functionalities that are reused inside their different components. With this, we can identify the main services reusable in the components. Eventually, these business services are categorized using SOA approach. The step of services identification is iterated until the designers are satisfied with the output.

After identifying the services, it is required to specify how they are to be built. The step of requirements specification for the control-command services enables to define the process by which we establish the physical designing of services and how to compose them to implement business components. During this step, services designers specify, for each identified service, its functionalities, exchanged variables and information sent back to the supervision operator.

Once these services requirements validated, control-command systems designers implement the identified services. Each designer focuses on its expertise field, using interfaces models previously described, and adapted tools. This step of control-command services implementation enables to transform specifications into graphical representations specific to each field of expertise (automatism, computing, etc.). Services implementation and promotion of reusability at this level enables to reduce designers effort for designing components.

Despite SOA being presented today as the most efficient approach for the development of complex company applications or web applications, in the industry we notice a lack of analysis tools to support development based on reusability [17], especially for the development of software components for which operation is known and documented, but not its internal structure. Indeed, in order to quickly and correctly obtain control-command components, it is best to capture knowledge from the expert who has all the functional knowledge of the system to be designed. Designing of services oriented components will be based on EUD/EUSE techniques, which have already proven useful in the area of industrial supervision [18]. Those techniques are integrated in a proof of concept enabling to validate our approach. A tool will be developed and will exploit the previously designed services to easily obtain reusable standard components and bespoke components. Though our tool, an expert who has no knowledge in HMI programming (computing) nor in command coding (automatism), easily composes services to design both command and supervision parts of components.

To quickly take into account component evolutions, the use of MDE is a well-adapted solution to issues of control-command system migrations from one platform to another [19]. Implementation of MDE therefore enables to get a complete control-command chain for the bespoke components and standard components the expert needs to design its system. The tool enables to design new components from existing components, promoting reusability at the level of components functionalities and ensuring a good coupling between services.

All designed components could then be simulated, analyzed and reintegrated in the tool to be improved. The specified services could also be improved and expanded depending on the identified shortcomings.

5 Case Study and Results

5.1 SOA Implementation: Services Identification and Implementation

SOA recommends to break down functionalities in a set of basic elements called «services». SOA implementation enables to finely describe the services interaction scheme as part of a business process. Services were then identified and deployed as independent software components. SOA use enabled to skip the step of heavy components construction (considered as black boxes) for which internal functionalities are not reusable. Analysis of components present in control-command systems led us to identify a set of services such as measurer, indicator light, command buttons, etc. Each service has inputs/outputs (Fig. 3) and service interfaces depending on the interface meta-model defined in Fig. 4.

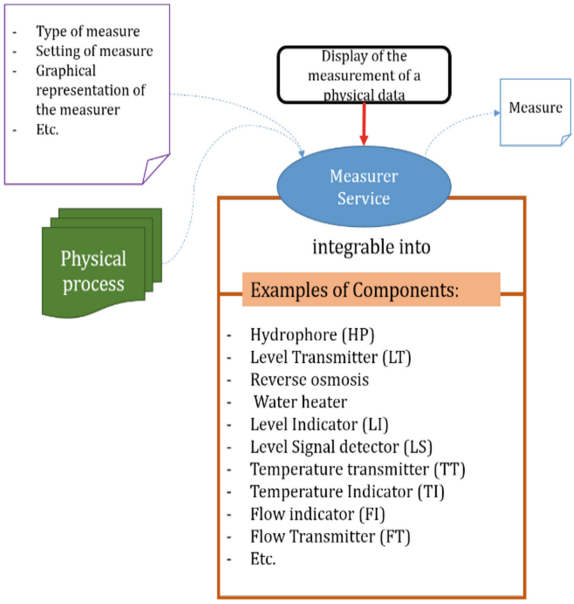

Fig. 3. Illustration of the measuring service

Figure 3 presents the measuring service that we defined, as well as examples of components in which we can find this service. Some components can be present in a same control-command system. Services definition reduce the designing effort on the system and increase the reusability on other systems, which can have different components using the same services. We therefore exploited service properties in the SOA

meaning: reusability, composability, independence and variable granularity. In good respect of these properties, we described services interfaces based on the meta-model of Fig. 4a. Parameters contained in this meta-model have been entered in a spreadsheet by the business expert, then automatically translated in XML format (Fig. 4b).

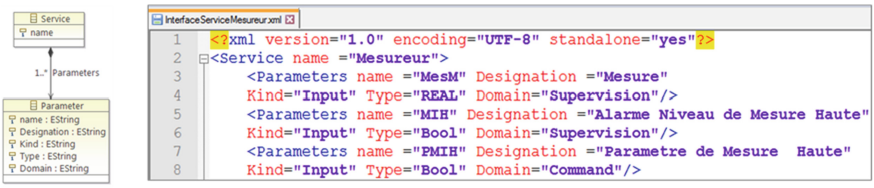

Fig. 4. Service interface meta-model

5.2 SOCD (Service Oriented Component Design) Tool for the Composition of Services in Components

Ideally, encapsulation of each service must ensure its reusability and its interoperability. The SOCD tool developed (Fig. 5) enables the designer without any HMI programming knowledge nor of command code to create, modify, visualize and delete command and supervision codes for the components of the system being designed. To do this, we used approaches based on MDE and EUD techniques to enable fast integration and reuse of components services.

By integrating these approaches, the proposed interface is made of three areas. The main area (1 on Fig. 5) displays the list of components already designed. The expert can modify a component from this interactive list. To modify a component, the user can click on the component name and the corresponding services are displayed in the area 2. He can modify service parameters by clicking on its name and save the changes («Enregistrer» in area 2). He can also delete a service from a component («Supprimer service» in area 2) or delete a component («Supprimer composant» in area 2). The third area regroups widgets enabling the user to create a new component from scratch or from an already existing component, to integrate the designed components in specific software («Test composant» in area 3) and to actualize the synoptic. Synoptic refers to a top-down approach for design [20] - it is an extract design of a control-command system which contains composition of necessary components for this system. Synoptic actualization is useful for designing bespoke components, to integrate them into automatic or manual design process and to take into account the changes made to components.

To add a new component from scratch, the user must enter the component name and choose the services he requires. The user has a choice between three pre-defined functions as services: measurer, indicator and command. He should be able to define the parameters characterizing each service through the interface.

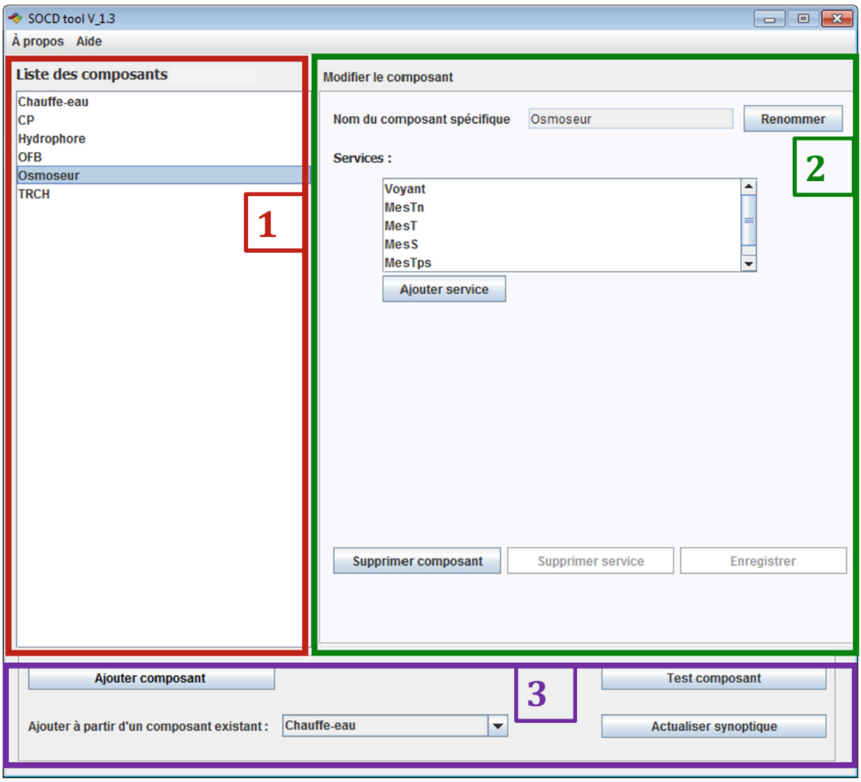

Fig. 5. SOCD tool for designing services oriented components

Adding a new component based on an existing component relies on the reusability mentioned in the EUSE approach [18]. The tool proposes a list of existing service oriented components for the expert to choose an existing component in the field «Ajouter à partir d'un composant existant» (Fig. 5) (in English "Add a component using existing one". Once the component chosen, the user can modify, delete or add new services to the component being created. This way, he chooses to reuse services of an existing component and to add new services; this leads to a time gain in the first designing steps.

Information related to user inputs during component designing is recorded in an XML file. This file enables MDE implementation for the automatic generation of command and supervision codes after the component creation. Once generated, the component can be tested by the expert using the «Test composant» button on Fig. 5. This button enables the expert to integrate component command codes in Straton tool [18] and its supervision codes in Panorama E2 tool [18], to visualize command views and components supervision respectively.

6 Evaluation

To validate the usefulness of our approach and the usability of the developed SOCD tool, we carried out user testing. User testing enables to understand the user real objectives and to identify any issue with using the tool. This method is efficient to increase a system or product ergonomics since it identifies up to 95% of ergonomic problems [21].

Tests of the SOCD tool are carried out with future designers, to check whether they would be likely to use our approach in designing component by services association. We also want to ensure that the interface will respond to their needs. We therefore evaluated the designing of four components: hydrophore, chlorination, water heater and osmosis. The aim of testing four components is to check the ease of use of the interface and to lead them to take most advantage of the reusability.

6.1 Method

For this evaluation, tests were carried out with 11 participants (6 students and 5 professionals) expert in process engineering from Polytech Nantes and University Technology Institute of Saint Nazaire (France) (9 men and 2 women). From 20 to 63 years old, they had a variable experience with control-command components.

The experimentation was carried out with a PC, a 23′ screen, a mouse and a keyboard and technical specifications were given for the components to design during the tests. With their authorization, the screen and their comments were recorded during the tests.

User testing were individual in separate rooms. Upon arrival, a commented slide show presented the context of the study to participants. They were then placed in front of the SOCD tool with an experimenter beside them.

Participants were asked to design the four components for which they had the technical specifications. They were asked to test at least one of the created component and to observe the result in the supervision view in Panorama E2. They were invited to think out loud for us to better understand their actions on the tool.

At the end of the test, they were asked to fill a questionnaire AttrakDiff [22] and another with more open questions to give their views on the tool.

6.2 Results and Interpretation

User testing results were encouraging even if some efforts remain to be made to improve the interface design. Some missing functionalities were highlighted, such as the possibility to all new services to components previously created or the arrangement of the elements on the interface. Despite these minor issues, we observed a good first use of the tool. Participants noticed it in their comments and the time required to build a component decreased from the 2nd component. Participants have been reported to appreciate the reusability of existing component services to create a new one. Furthermore, the approach of designing components by association of services was well received. They qualified the tool as controllable, foreseeable, practical, simple, clear, new, motivating, and easy to use; all being very positive.

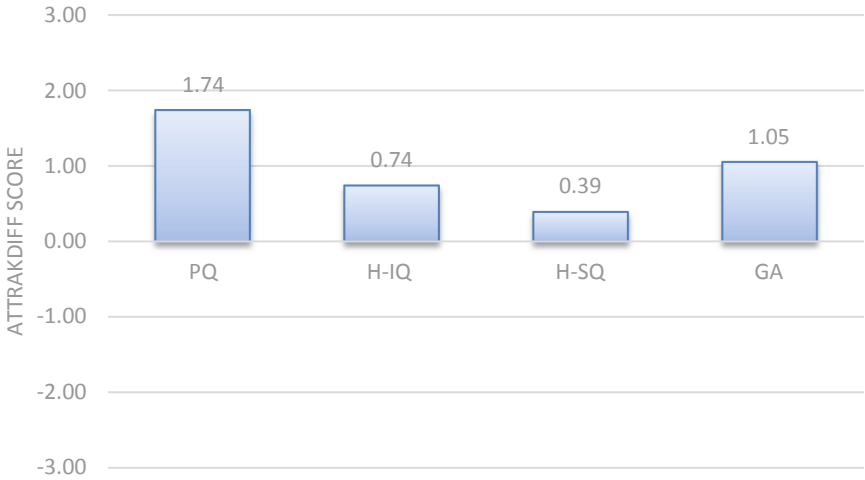

Fig. 6. Average score by items category (AttrakDiff questionnaire)

Figure 6 presents the AttrakDiff questionnaire results which evaluate the perceived quality of an interactive system. It includes 28 items in 4 sub-scales (pragmatic quality PQ, hedonic-stimulation quality H-SQ, hedonic-identity quality H-IQ and global attractivity GA). These different items are detailed in [22]. To analyze the results, average values are presented in the −3 to 3 range. Values between 0 and 1 are considered in a neutral zone, and values outside this zone are considered either as positive (1 to 3) or negative (−3 to −1).

Scores collected with the AttrakDiff questionnaire are very encouraging. The average PQ score of 1.75/3 describes the usefulness, utilisability and the success in carrying out the tasks when using the system. The analysis by sub-scale show that H-SQ and H-IQ have the lower average scores with 0.74/3 and 0.39/3 respectively. These sub-scales correspond to hedonic characteristics which are linked to emotions, affects, etc. These low scores are explained by the fact that the proposed tool is described as not fun, which some participants found normal since they qualified the tool as professional. These user tests have confirmed the usefulness of our approach, the usability of our tool and have evaluated the tool based on measuring scales linked to user experience. However, the tool must now be improved to better facilitate user tasks, since user centered design is an iterative process where evaluation is fully integrated in the designing process.

7 Conclusion

A component is a system element providing a predefined service and capable of communicating with other components. In the designing of control-command systems, component design is a very important step. Existing technologies, such as CORBA or

CCA components, do not fulfill our expectations. Moreover, limits of the component approach lead us to rethink the way we design components. Our approach combining SOA, EUD/EUSE and MDE enables us to bring solutions to identified challenges on reusability and interoperability of functionalities (services) encapsulated in the components during their designing.

Integration of SOA techniques made it possible to develop a solution to obtain components that are simple, modular and with loose couplings. These characteristics allow for quick and easy recombination of the arrangement of functionalities they provide. By its approach of designing and construction of services as independent applicative blocs, SOA facilitates process instrumentation. Our work demonstrates that this approach can be used for developing services more frequently found in control-command systems. It led us to the definition and implementation of the SOCD tool enabling a user that is expert in his field, but not an automatism or computing specialist, to create and modify or reuse standard components and bespoke components.

Integration of EUD and EUSE approach enabled the reusability of functionalities (services) internal to components, as well as errors correction. MDE through the DOM API for the XML files processing allowed for the joint and automatic generation of command and supervision codes for the components.

Our approach therefore facilitates the designing of components whilst respecting the information contained in the requirements specification and while promoting reusability of internal services of a component in different contexts. Thanks to this approach, the component is not seen as a black box but rather a flexible element for the designing, which can be handled by non-programmers. Usability (ISO 9241-11 and ISO-13407) of our tool was demonstrated during the users testing.

In the future work, we will improve the tool to better facilitate user tasks.

Acknowledgements. We would like to thank students and teachers of the Polytech Nantes and University Technology Institute of Saint Nazaire for their participation to the user tests.

References

1. Dai, W., Peltola, J., Vyatkin, V., Pang, C.: Service-oriented distributed control software design for process automation systems. In: 2014 IEEE International Conference on Systems, Man and Cybernetics, pp. 3637–3642 (2014)
2. Cervantes, H.: Vers un modèle à composants orienté services pour supporter la disponibilité dynamique. Ph.D. thesis. Informatique (2004)
3. Kesraoui-Mesli, S.: Intégration des techniques de vérification formelle dans une approche de conception des systèmes de contrôle-commande. Ph.D. thesis, no. Umr 6285 (2017)
4. Pichon, V.: Contribution à la conception à base de composants logiciels d'applications scientifiques parallèles. Ph.D. thesis, Ecole Nationale Supérieur de Lyon - Université de Lyon (2012)
5. Szyperski, C.: Component Software – Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley Longman Publishing Co., Boston (2002)
6. OMG. CORBA component model, v4.0 (2006). http://www.omg.org/spec/CCM/4.0/
7. Courtrai, L., Guidec, F., Mahéo, Y.: Gestion de ressources pour composants parallèles adaptables. In: JC (2002)

8. Bernholdt, D.E., Elwasif, W.R., Kohl, J.A., Epperly, T.G.: A component architecture for high-performance computing: an overview of the common component architecture, pp. 1–10. SANDIA National LABS LIVERMORE CA (2005)
9. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.-B.: The FRACTAL component model and its support in Java. Softw. Pract. Exp. **39**(7), 701–736 (2006)
10. Bignon, A.: Génération conjointe de commandes et d'interfaces de supervision pour systèmes sociotechniques reconfigurables. UBS (2012)
11. Zachman, J.A.: Enterprise Architecture Artifacts Versus Application Development Artifacts, vol. 28 (2000)
12. Dami, L., Konstantas, D., Pintado, X.: Object-Oriented Software Composition, vol. 1. Prentice Hall, Englewood Cliffs (1995)
13. Ait Abdelouhab, K., Idoughi, D.: Conception centrée utilisateur de services dans le cadre des architectures orientées services. Thèse de doctorat, Université de bejana (2016)
14. Courbis, C., Degenne, P., Fau, A., Parigot, D., Variamparambil, J.: Un modèle de composants pour l'atelier de développement SmartTools. Journée systèmes à Compos. Adapt. extensibles (2002)
15. Papazoglou, M.P., Van Den Heuvel, W.: Service-oriented computing: state-of-the-art and open research issues. IEEE Comput. **40**(11), 38–45 (2007)
16. 9241-210:2010 ISO: Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems (2010)
17. Stojanović, Z., Dahanayake, A.: Service-Oriented Software System Engineering: Challenges and Practices. IGI Global, Hershey (2005)
18. Goubali, O.: Apport des techniques de programmation par démonstration dans une démarche de génération automatique d'applicatifs de Contrôle-Commande. ENSMA (2017)
19. Vernes, P.J.: Algorithmes de Contrôles Avancés pour les Installations à Gaz du LHC au CERN suivant le Framework et l'approche dirigée par les modèles du projet GCS. Ph.D. thesis, Université de Picardie Jules Vernes (2008)
20. Bignon, A., Berruet, P., Rossi, A.: Joint generation of controls and interfaces for sociotechnical and reconfigurable systems. In: 2010 IEEE International Conference on Systems, Man and Cybernetics (2010)
21. Boucher, A.: Ergonomie web: pour des sites web efficaces, 2nd edn. Eyrolles (2009)
22. Diefenbach, S., Hassenzahl, M., Eckoldt, K., Hartung, L., Lenz, E., Laschke, M.: Designing for well-being: a case study of keeping small secrets. J. Posit. Psychol. **12**(2), 151–158 (2017)