Testing As A Service for Component-based Developments

Hien Le
Department of Telematics
Norwegian University of Science and Technology
hiennam@item.ntnu.no

***Abstract*— In this paper, we present an approach to model testing as a service for component-based development. The approach is based on the Service-oriented Architecture in which testing services are modeled using UML collaboration structure to support the validation of components. We categorize two types of components: elementary and composite. Elementary components are non-decomposable and reusable computing units. Composite components are developed by composing existing components, which can either be elementary or composite ones. Our main contributions presented in this paper are: (1) to provide an approach for modeling component testing as a service; and (2) to provide a constructive mechanism for composing testing services. In this paper, testing services for railway control system will be used to illustrate our approach.**

Keywords – software components; component testing; testing as a service

I. INTRODUCTION

A component, in general, may be defined as a reusable software or computing unit [1], which is designed to partially or fully perform specific functionalities invoking through component interfaces. The reusable components are normally verified, validated and stored in a repository. Component-based development is a software development approach in which new components are developed by composing existing components retrieved from the component repository [2] to satisfy new requirements. By this approach, on the one hand, new components and software systems can be rapidly developed [3, 4] while reducing development efforts and costs. On the other hand, however, there are many challenges, for examples, how to ensure that these newly developed components do not posse any unusual behaviors [7, 8, 10] while fulfill the requirements.

Component verification and validation are software development activities whose aim is to ensure that newly created components fulfill the requirements without introducing any emerging or unexpected behaviors [7, 10]. In this paper, an approach to model testing as a service to support the component validation, also known as component testing to guarantee that the component fulfills its expected functionalities when performing in the intended environment [12], is presented. The approach is based on the Service-oriented Architecture in which testing services are modeled and composed using UML collaboration structure to support the validation of components.

As shown in Figure 1, the *ComponentUnderTest* represents the service clients, which are newly developed components. These components must be validated. *TestingServiceProvider* plays the role of the service providers, i.e., providing simulation environments and testing suites for validating the new components. *TestingServiceRegistry* is where the descriptions of testing services are published so that they can be found by the service clients, i.e., the *Component Under Test*. When a suitable testing service has been matched with the testing requirements of the new components, the validation process (refereed as testing process in this paper) for these new components can be carried out. The testing process, which is modeled and deployed as a service, emphasizes that testing services are independently developed from the component-based development view; and newly testing services can be developed by composing existing testing services in the same manner as service composition [9]. However, to be able to apply the Service-oriented approach for supporting component testing, we must answer two questions: (1) how to model testing as a service; and (2) how to compose testing services, i.e., constructing new testing services as a composition of existing ones. In this paper, we focus our discussion on these two issues.

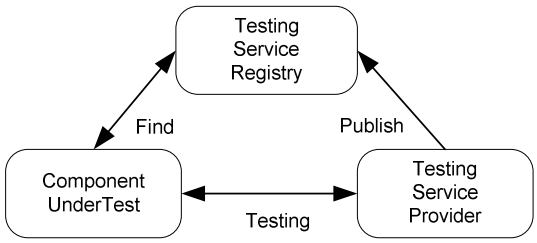

Figure 1: SOA for testing services

In the following discussion, we categorize two types of components: elementary and composite components. Elementary components are the ones which can not be decomposed further. Composite components are composed from existing components, which can be either elementary or composite one. Normally, an elementary component is first designed, verified and validated and stored in a repository to be re-used [4]. The validation of components is ensured by applying test suites to the component interfaces in a simulation environment [11].

The rest of the paper is organized as follows. Related work is discussed in Section II. Section III presents the modeling approach which is based on UML collaboration structure to model testing as a service. Section IV discusses how to create new testing services by service compositions. Conclusion and future works are given in Section V. A railway control system which is built by component-based development approach will

 ISBN: 978-1-61208-168-7

be used to illustrate the applicability of our testing service modeling approach.

II. RELATED WORK

In this section, we discuss the related work on modeling testing as a service for components and how to compose testing services. To our knowledge, there are many approaches that support the validation of elementary components [13, 14]. However, the current research which focuses on validating of composite components is very limited [7, 8]. These existing approaches mainly focus on testing specification [14], generating test cases for component testing [7] or performance [11, 13]. Furthermore, these testing approaches do not differentiate the different between elementary and composite components. In [11], a testing method which utilizes the Service-oriented architecture to support testing of complex and safety-critical systems is presented. However, this testing approach focuses on the distribution and performance of testing process, e.g., distributed testing among testing hosts, rather than how to model testing as a service. Existing approaches for designing test suites of elementary components may not be applicable to composite components due to, for example, the new dependencies between sub-components which are the results of composed behaviors of components. Furthermore, the question of how to re-use the test suites or simulation environments, which have been used to validate the elementary components, in the new testing services for composite components may not be fully addressed.

In our recent research [15], a service can be defined as "*an identified functionality aiming to establish some desired effects among collaborating entities*". We have also shown that, based on the collaborative service models, reusable components can be automatically synthesized and such components can then be composed together [16]. Based on this approach, we argue that testing can also be modeled as a service, whose desired goal is to validate the behavior of components, i.e., the two collaborating entities are the component under test and the testing component. From the service models and choreography models of testing services, testing components will be generated and deployed for testing process. Our approach presented in this paper does not focus on issues related to generate test suites for component testing or testing specification (e.g., TTCN-3 [14]), but contributes to modeling testing as a service at abstraction level and to support composition of testing services. This way, the testing of components can be specified at the early phase in the component development lifecycle [2].

III. MODELING TESTING AS A SERVICE

In this section, we first present a railway control system, which is built using a component-based development approach. Second, we will discuss how to model testing as a service for component testing.

A. Train control scenario

Figure 2 shows the overview of the train control system, which is modelled using UML collaboration structure. The operation of the train control system is described as follows. While moving in a geographical region, a *Train* must always be supervised by the *Train Controller Center* (TCC). The TCC responsibility is to monitor and control all train movements in a region.

- The train position on the railway track system is always monitored by the *TCC*. The train, while moving, keeps sending its position report to the TCC. This is modeled as collaboration activity between the *Train* and the *TCC* (i.e., the *PositionReport* collaboration shown in Figure 2).
- The TCC validates the received position information of the train and will issue successive movement authorities (MA) to the train. The MA specifies a safe distance that the train can travel. This is modeled by the *Movement Authority* collaboration.

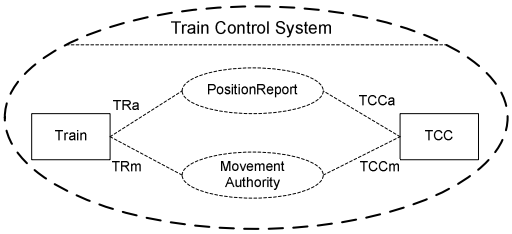

Figure 2: Collaboration structure of the train control system

Based on the collaboration models, the service models and the behavior models of the train control system will be developed and finally the components of the train control system will be synthesized [15, 16]. Figure 3 shows the architecture overview of components of the train control system. The train control system will have the following components.

- The *Position Report* component, which is a sub-component of the *TrainMovementControl* component, reads the location of the train from the external environments, i.e., location indicator installed on the railway tracks [5], and sends this information to the *TCC* component at the control center.
- The *Movement Authority* component handles the movement authority, which is send by the *TCC* to the train. The *Position Report* and the *Movement Authority* components also collaborate to ensure that the train will not travel beyond the safe distance.

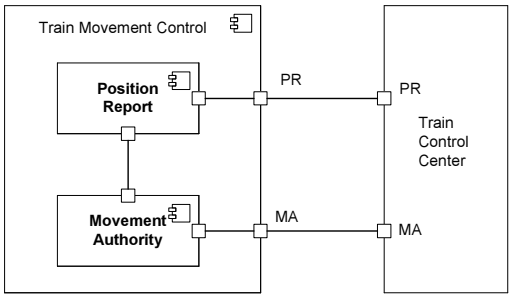

Figure 3: Component view of the train control system

In order to validate the behavior of the *TrainMovementControl* component, which is composed from the *PositionReport* and *MovementAuthority* components, the developer must carry out the following component testing:

- Testing of sub-components: in this case, both the *Position Report* and *Movement Authority* components must be fully tested. The testing of sub-components may in addition require several simulation modules or components [11] which represent the external environments, e.g., location indicators on the railway track systems.
- Testing of the composite component: in this case, the behavior of the composite *TrainMovementControl* component must be verified and validated. In order to validate the *TrainMovementControl* component, the *TCC* counterpart must be available. By our approach to model testing as a service, the corresponding *TCC* will be replaced by a testing component, whose behavior is equivalent to the real *TCC* component (i.e., the *TrainControlCenter* component as shown in Figure 3) during the testing process.

In order to support the testing process, a testing service for components must first be modeled and developed. Next, we present the approach to model testing as a service for components.

B. Testing service for components

The objective of the testing service for components is to support the validation of components at the early stage of development, i.e., design step. Our testing service is based on the concepts of services in which services are defined as a collaboration activity among entities to achieve service goals [6, 15]. Figure 4 shows the basic service structures of the testing service for components.

As shown in Figure 4, the testing service has two main structures, which are specified based on the UML collaboration structure [5], *Simulating* and *Inspecting*. The objective of the *Simulating* is to provide a structural view if the component under test (CUT) requires additional simulation modules. The *Component* role represents the component under test (CUT), and the *EnvSimulator* represents the simulation environment which is required so that thorough test on the component can be performed. The *Inspecting* structure presents the actual testing activity applied on the component, i.e., test suites execution via the *Inspector* role.

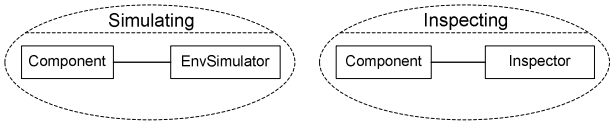

Figure 4: Testing service structures for components

Figure 5 shows the structure of the *Testing Service*, which is the composition of the two testing services, i.e., the *Testing Service* is the composition of *Simulating* and *Inspecting*. The *Testing Service* collaboration includes two main roles: the *ComponentUnderTest* (CUT) role and the *Tester* role. When the testing is performed, the role *ComponentUnderTest* will be dynamically binding to the actual component which will be tested. The main operation of the *Tester* role is to play the role of the testing component which includes the environment simulator (i.e., *EnvSimulator* role) and generated test suites, i.e., to submit test cases to the *ComponentUnderTest* via the *Inspector* role in an intended operation environment. In other words, the *Tester* will implement the interface of the complement testing component. Based on this model, we can identify the structure and specify the test services which take into account the correlation between the required simulation modules and test cases executors.

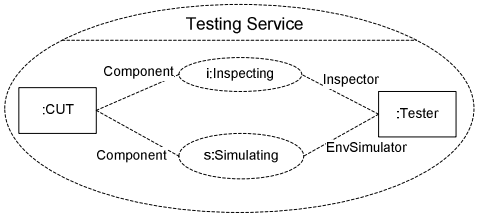

Figure 5: Test model for components

Figure 6(a) illustrates how the *Testing Service* is applied for testing the *Position Report* component. The role *CUT* of the *Testing Service* will be performed by the *Position Report* component, and the *Tester* role will be executed by the *PR_Tester* component, whose functionalities includes both the environment simulation and inspector. Figure 6(b) shows the involved components in the testing process: the *Position Report* is the developed component, and the *PR_Tester* component is synthesized from the testing service model.

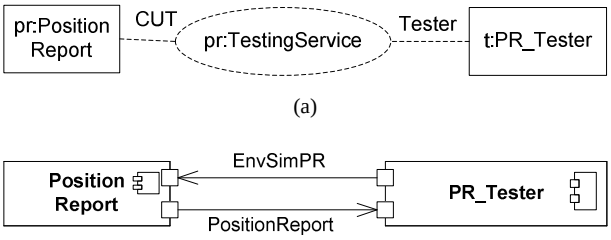

Figure 6: Testing service for Position Report component

IV. COMPOSITION OF TESTING SERVICES

In this section, we present our approach to create testing services which are applied to composite components. In this approach, we discuss an integrated testing service to generate required composite testing services which are composed based on the existing testing services (i.e., of existing components). To simplify our discussion without losing the general discussion details, we assume that all the sub-components of the train control systems have been verified and validated.

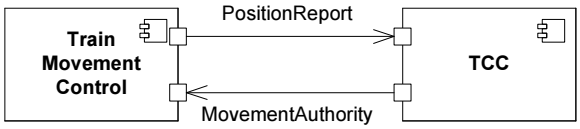

Figure 7: Composite component testing

A. *Integrating testing services for composite components*

As described in Section III, based on the information of the position of the train, the *TCC* will issue movement authority to the train so that the train can safely continue to travel. This means that, for testing the composite component *TrainMovementControl*, the *Tester* role now will be performed by the composite testing component *TCC* which includes both *PR_Tester* and *MA_Tester* roles (as shown in Figure 7). In other words, the output of the *PR_Tester* testing will be validated before the testing of movement authority functionality, i.e., the *MA_Tester*, can be performed. In order to handle the dependency of testing services, we propose an *Integrating Test Service* which provides a mechanism so that the two sub-roles of the *Tester*, i.e., *PR_Tester* and *MA_Tester*, can collaborate. The structural model of the *Integrating Test Service* is shown in Figure 8(a). There are two main roles: *outTester* and *inTester* which perform the sending results from the previous testing service, i.e., testing of the *Position Report* component, and initiating the next testing service, i.e., testing of the *Movement Authority* component.

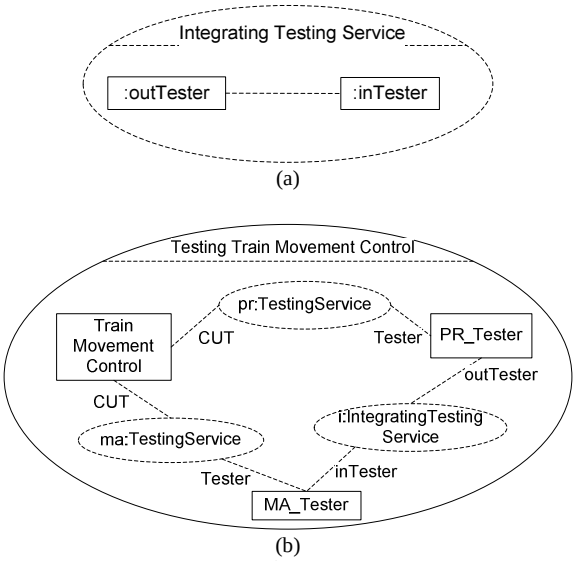

Figure 8: Integrating testing service for Train Movement Control component

Figure 8(b) shows how the *Integrating Testing Service* is re-used and composed to the composite testing service, explained as follows:

- The *pr:TestingService* collaboration is the original testing service for the *Position Report* component and involves two roles *CUT* and *Tester*.
- The *ma:TestingService* collaboration is the original testing service of the *Movement Authority* component and involves two roles *CUT* and *Tester*.
- The *IntegratingTestingService* is re-used to integrate the two existing testing services *pr:TestingService* and *ma:TestingService*. In this situation, the role *outTester* and *inTester* is binding to the *PR_Tester* and *MA_Tester*, respectively.

There are several advantages of our *Integrating Testing Service*. First, the testing service provides a flexible mechanism to support the integration of testing services which have been applied to existing components. Second, the integrating test service focuses on describing the integration of testing services at the design stages while components are being developed. This ways, the testing of composite component can be early specified and carried out.

B. *Realization and deployment of testing services*

The *Integrating Testing Service* provides a mechanism for composing testing services for composite components. This testing service can be deployed in either centralized or distributed testing systems. For example, a centralized testing system can be deployed if both *outTester* and *inTester* roles are realized, i.e., implemented, as testing sub-components of the *Tester* component. In other words, the *Tester* will now perform both *PR_Tester* and *MA_Tester* roles. Figure 9 illustrates a distributed testing scenario in which the sub-components *Position Report* and *Movement Authority* are tested in different systems. In this case, both the distributed testing sub-components *PR_Tester* and *MA_Tester* must implement the *Integrating Testing Service* interface, i.e., *outTester* and *inTester* roles, respectively.

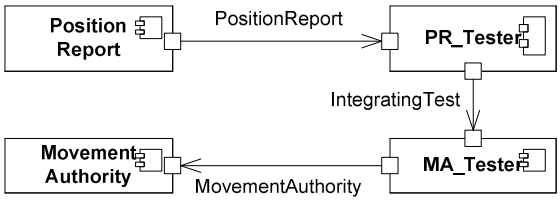

Figure 9: Distributed testing scenario

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented an approach to model testing as a service for component-based development approach. An *Integrating Testing Service* which supports the composition of testing services, i.e., to support the integration and re-usability testing services of existing components, is also presented. This ways, new testing services for composite components can be quickly composed and deployed in either centralized or distributed testing systems.

In future work, we plan to further using the Model-Driven Development approach to automatically synthesize the testing components. A full testing framework, which includes both service models [15] and component-based approach [16], can be developed to dynamically discover and compose for testing of composite components.

REFERENCES

[1] Clemens Szyperski. Component Software: Beyond Object- Oriented Programming. Addison-Wesley Longman Publish- ing Co., Inc., Boston, MA, USA, 2002.

 ISBN: 978-1-61208-168-7

[2] Ivica Crnkovic, Stig Larsson, and Michel R. V. Chaudron. Component-based Development Process and Component Lifecycle. CIT 13(4), 321-327, 2005.

[3] Jisa Dan Laurentiu. Component based development methods: comparison, Computer systems and technologies, 1-6, 2004.

[4] Kung-Kiu Lau and Zheng Wang. Software Component Models. IEEE Trans. Software Eng. 33(10): 709-724, 2007.

[5] Surya Bahadur Kathayat, Rolv Bræk, and Hien Le. Automatic derivation of components from choreographies - a case study. International conference on Software Engineering, 2010.

[6] Surya Bahadur Kathayat and Rolv Bræk. From flow- global choreography to component types. In System Analysis and Modeling (SAM), LNCS 6598, 2010.

[7] Camila Ribeiro Rocha and Eliane Martins. A Method for Model Based Test Harness Generation for Component Testing. 14(1): Journal of the Brazilian Computer Society (JBCS), 7-23, 2008.

[8] Gerardo Padilla, Carlos Montes de Oca, and Cuauhtemoc Lemus Olalde. An Execution-Level Component Composition Model Based on Component Testing Information. 10th International Symposium on Component-Based Software Engineering, 2007.

[9] Surya Bahadur Kathayat, Hien Le, and Rolv Bræk. A Model-Driven Framework for Component-based Development, SDL forum 2011 (to appear).

[10] Jerry Gao and Ming-Shih Shih. A Component Testability Model for Verification and Measurement. International Computer Software and Applications Conference (COMPSAC), 2005.

[11] Renato Donini, Stefano Marrone, Nicola Mazzocca, Antonio Orazzo, Domenico Papa, and Salvatore Venticinque. Testing Complex Safety-Critical Systems in SOA Context. International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), 2008.

[12] 7CMU/SEI-2000-TR-028, CMMISM for Systems Engineering/Software Engineering, Version 1.02, Software Engineering Institute, 2000.

[13] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. Cloud9: a software testing service. Operating Systems Review 43(4), 5-10, 2009.

[14] Bernard Stepien, Liam Peyton, and Pulei Xiong. Framework testing of web applications using TTCN-3. Journal on Software Tools for Technology Transfer (STTT), 10(4), 371-381, 2008.

[15] Surya Bahadur Kathayat, Hien Le and Rolv Bræk. Collaboration-based Model-Driven Approach for Business Service Composition. Book chapter in Handbook of Research on E-Business Standards and Protocols: Documents, Data and Advanced Web Technologies, Ejub Kajan, Frank-Dieter Dorloff, Ivan Bedini, IGI, 2011 (to appear).

[16] Surya Bahadur Kathayat, Hien Le and Rolv Bræk. A Model-Driven Framework for Component-based Development, SDL 2011 (to appear).

 ISBN: 978-1-61208-168-7