Chapter 87
Contract-Based Combined Component Interaction Graph Generation

Haiqiang Li, Min Cao and Jian Cao

Abstract Component interaction graph is used to describe the interrelation of components, which provides an objective basis and technology to test component composition. However, the traditional component interaction graph cannot serve as a basis to test a component itself and the state transition between components for lacking of description of states of individual component. Therefore, a novel model, named Contract-Based Combined Component Interaction Graph (CBCCIG) is put forward in this paper. CBCCIG combined the thought of contract test with the UML state diagram which is introduced in the paper. The proposed model can not only support the quick assembly of the software system depending on developer's own willing, but also the automatic or semi-automatic generation of test cases which are the state transition and information interaction between components. Thus, CBCCIG improves the efficiency of development and testing.

Keywords CBSD · Component composition · CIG · UML state diagram · Testing by contracts

87.1 Introduction

Component-based Software Development (CBSD) is an effective and efficient approach to improve the productivity and quality of software development [1]. In CBSD, the most important thing is how to obtain suitable components and integrate them to product a reliable software system. Component composition is a

H. Li (✉) · M. Cao
School of Computer Engineering and Science, Shanghai University, Shanghai, China
e-mail: lhqmaillove@163.com

J. Cao
School of Physics, Nankai University, Tianjin, China

W. E. Wong and T. Ma (eds.), *Emerging Technologies for Information Systems, Computing, and Management*, Lecture Notes in Electrical Engineering 236, DOI: 10.1007/978-1-4614-7010-6_87,

critical process which determines whether your CBSD can acquire, reuse, or build a component. Due to different component versions, different component technologies, and different integrated environments, there is no mature technical standard and feasible method to capture the mistakes in the integration testing [2]. CBSD improves the efficiency of software development, but it brings the testing difficulties at the same time.

Component Interaction Graph (CIG) is used to describe the interrelation of components, which provides an objective basis and technology for the implementation of component composition testing [3]. Nowadays, there are many researches on the generation of CIG. Ye Wu et al. [4] presented a method to construct the CIG in which the interactions and the dependency relationships among components are illustrated. By utilizing the CIG, they propose a family of test adequacy criteria which allow optimization of the balancing among budget, schedule, and quality requirements typically necessary in software development. Based on the direct and indirect correlation analysis, CIG was established [5]. By using the component specification structure and the established CIG, the component interactions can be modeled to provide support for testing component-based software. Lun [6] represented software architecture possessing C2 style through CIG, and abstracted the behavior of interactive between components and connectors, then they defined three testing criteria and introduced algorithms to generate testing coverage set according to edge types of CIG.

The above methods provide theoretical and experimental basis for the generation of CIG. However, these traditional CIGs do not describe the state of individual component. They cannot serve as a basis for testing a component itself and the state transition between components. In this paper, a component is firstly represented in the form of UML state diagram. Then, synthesizing CIG and UML state diagram, we propose a novel model, named (CBCCIG), to generate test cases of the state transition between components and information interaction.

87.2 Related Concepts

87.2.1 Component and Component Composition

Component interfaces are the access points of components, through which a client component can request a service declared in an interface of the service providing component. Each interface is identified by an interface name and a unique interface ID.

Definition 1 A component is a 2-tuple C = (P, R), where:

- $P = \{P_1, P_2, \ldots, P_n\}$ is the set of providing services interface.
- $R = \{r_1, r_2, \ldots, r_n\}$ is the set of required services interface.

In this case, collections and the elements in the collection are represented by capital letters and small letters respectively.

Definition 2 Component composition: The composition of two component means that the required services of one component are provided by another partly or fully.

87.2.2 Testing by Contracts

A contract is a stipulation between two parties, containing benefits and obligations for each part. Design by Contract (DBC) is an object oriented design technique that ensures high-quality software by guaranteeing that every component of a system lives up to its expectations [7]. Under the design by contract theory, a software system is viewed as a set of communicating components whose interaction are based on precisely defined specifications of the mutual obligations–contracts. Every good contract entails obligations as well as benefits for three parties: (1) the precondition; (2) the post-condition; (3) the invariant.

87.3 Contract-Based Combined Component Interaction Graph

87.3.1 CIG

87.3.1.1 Semantic Analysis

A CIG is a directed graph which is used to depict interaction scenarios among components. The major elements related to interactive feature are interfaces, events, context dependence and content dependence [4].

- Interfaces:
  Interfaces are the basic access means via which components are activated.
- Events:
  We define an event as an incident in which an interface is invoked in response to the incident. The interface events defined in the CIG are usually methods.
- Context dependence:
  One event has a context dependence relationship with the other event if there exists an execution path which triggers one event directly or indirectly.
- Content dependence:
  The content dependence relationship is defined as follows: a function (named functions 2) depends on another function (named function 1) if the value of a variable defined in function 1 is used in function 2.

87.3.1.2 Mathematical Definition

Definition 3 A CIG is a 2-tuple CIG= < V, E > , where:

- V = VCUVE is a set of nodes. Accordingly to definition 1, VC = (P, R) is the set of component interface nodes, VE is the set of event caused by component.
- E = ECUED represents a set of directed edges. EC represents context dependence, ED represents content dependence.

If there is an existing edge form C1.P1 to C2.R1 in the CIG, it means the required service R1 of C2 has been satisfied by the providing service P1 of C1, namely, C2.R1 = C1.P1. We denote an interface with an ellipse and a component with square, and the interfaces belong to one component that was drawn in the same square. Then the CIG is built as follows shown in Fig. 87.1.

87.3.1.3 Effects and Problems of CIG

Component-based software is often built through component composition. Component interfaces are the access points of components and define all content of interaction with the external. The only way that a component communicates with the outside is component interface. We modeled component interaction by establishing CIG, which can describe the interaction semantic better and also provide support for testing component-based software. At the same time, the test model can be useful to explain interactive and dependent relationship between components. Both the direct and indirect interaction relationship between components, based on which the test cases are chosen, by traversing the CIG. However, in the traditional component interaction graph, components are presented in the form of interfaces, which does not describe the state of individual component. CIG cannot serve as a basis for testing a component itself and the state transition between components. Therefore, we introduce UML state diagram to represent the state of components. The combination of CIG and UML state diagram can be used to generate the test cases of the state transition and interaction between components.

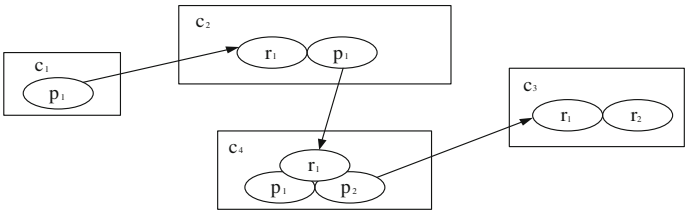

Fig. 87.1 Component interaction graph

87.3.2 UML State Diagram

87.3.2.1 Semantic Analysis

UML state diagrams depict the various states that a specific component may be in and the transitions between those states, which will be modified by events. UML state diagram consists of states, transitions, events and actions [8].

- States:
  States are defined as a condition in which a component is in. State will change when some event is triggered.
- Transitions:
  A transition is a progression from one state to another, triggered by an event which is either internal or external to the component. It also causes an important change of state.
- Events:
  Events will cause some actions and the transitions between states. Generally, they are method invocations.
- Actions:
  An action is an operation of an active component. When an event is dispatched, the component responds with performing actions, which cannot be interrupted.

87.3.2.2 Mathematical Definition

Definition 4 A state chart diagram is a 5-tuple $\mathrm{SD} = (\mathrm{S}, \mathrm{E},\ \mathrm{F}, \mathrm{s}_0, \mathrm{s}_\mathrm{F})$ where:

- $\mathrm{S} = \{\mathrm{s}_0, \mathrm{s}_1, \ldots \mathrm{s}_\mathrm{i}, \ldots, \mathrm{s}_\mathrm{n}\}$ is a finite set of states, where $\mathrm{i} \in (0, \mathrm{n})$.
- E is a finite set of event driven of state chart diagram.
- F is a finite set of transitions. $f : \mathrm{S} \times \mathrm{E} \rightarrow \mathrm{S}, f(q, e) = p$, where $\forall \mathrm{e} \in \mathrm{E}$.
- $\mathrm{s}_0 \in \mathrm{S}$ is an initial state. A SD must have one and only one initial state.
- $\mathrm{s}_\mathrm{F} \subseteq \mathrm{S}$ is a nonempty set of final states.

87.3.2.3 Effects

In our model, we represent the components of CIG in form of UML state diagrams. The proposed approach indicates not only the state transition of a component itself, but also the state transition between components. Therefore, Combined Component Interaction Graph (CCIG) is putting forward by combining CIG and state chart diagram. CBCCIG introduces the thought of contract in CCIG.

87.3.3 CBCCIG

87.3.3.1 Thought and Semantic Analysis

In CIG, the interaction among components can be described as directional arrows, with the providing service interface points to required services interface. We define an event as an incident in which an interface is invoked in response to it. They are usually methods. Also, the state transition of UML state diagram can be described as directional arrows, with initial state points to final state. The arrows with state input information are methods too. Therefore, CIG and UML state diagram in the same system can be combined. We represent the components of CIG in form of UML state diagram, and combine the special features of these two models.

In our model, there is a one-to-one mapping between the interfaces of CIG and the states of UML state diagram. At the same time, we analyze the transition arrows with method name and finally form the CBCCIG.

87.3.3.2 Mathematical Definition

Definition 5 A CBCCIG is a 5-tuple CBCCIG = (C, CS, CE, CF, CG), where:

- C is a finite set of all components.
- CS is a finite set of states of C. $c_i s_j$ represents the sate j of component i.
- CE = (precondition, E, postcondition) represents a set of events of C. We have described the concepts of event in the previous chapters in the component interface and state transition manner. However, there are no restraint conditions to guarantee the proper operation of the method, such as the accuracy of input parameters and return results. Therefore, in order to ensure the accuracy of the interaction and connection among components, we introduce the basic idea of contract testing, and add some constraint rules like precondition and postcondition to event.
- $s_0 \in S$ is an initial state. A SD must have one and only one initial state.
- CF is a finite set of transitions between component states. $f: CS \times CE \rightarrow CS, cf(c_i s_j, e) = c_p s_q$ represents the transition from $c_i s_j$ to $c_p s_q$, where $\forall e \in CE$.
- $CG \subseteq CS$ called intermediate state. These states neither cause an event to interact with another component actively nor need the service provided by component itself or other components.

87.3.3.3 Generation Algorithm

Definition 6 The abstract mapping between interface and state: let a component c_1 be at state s_1, and a component c_2 be at $states_2$. If c_1 is triggered by an event and

it reaches at state s_2 automatically, we definite s_1 as an interface which provides service and in reverse s_2 as an interface which requires service.

According to above definition, we present a way to generate the CBCCIG.

Algorithm:
Components $C=\{c_1, c_2, \cdots, c_i, \cdots, c_n\}$
States $S=\{s_0, s_1, \cdots, s_j, \cdots, s_m\}$
1. Begin
2. Remove the states which switch between the components.
3. For i=1 to n
4. For j=0 to m
Search a state s_j from component c_i from which, occurring of an event activate another component.
5. Set s_j as p_j.
6. EndFor
7. EndFor
8. For i=1 to n
9. For j=0 to m
10. Search a state s_j from the component set which require services.
11. Set s_j as r_j.
12. EndFor
13. EndFor
14. If a state is not $p_j \| r_j$
15. Then set it as an intermediate state G_j.
16. EndIf
17. End

Figure 87.2 is an incorporative CIG, in which a rectangle, a circle and a square denotes component, component state and component interface respectively. In addition, the solid line represents the state transition of individual components, and

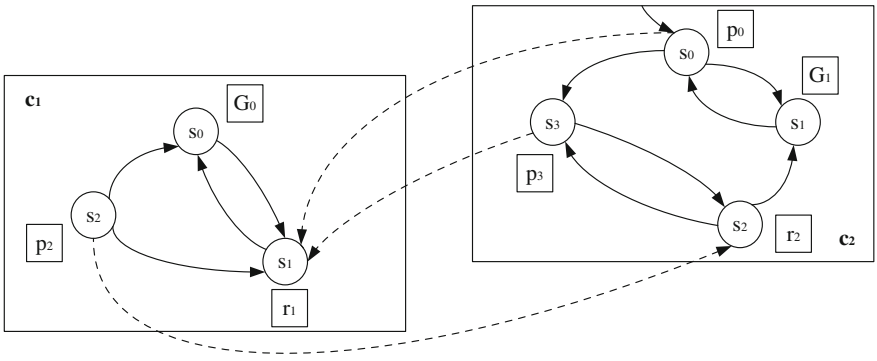

Fig. 87.2 Combined component interaction graph

the dotted line represents the state transition among components. p is the providing service interface, while r is the requiring service interface.

87.3.4 Research Significance of the CBCCIG

The core technology of CBSD is component composition. A large number of components need to be assembled together for a complex system. Most of the component composition testing is realized with the help of the combination of the component models, for example the CIG which provides an objective basis and technology for the implementation of the component composition testing. In this paper, the proposed model, named CBCCIG, not only contains the assembly and interactive elements that the development model needed, but also increases the testing elements, such as state, contract and so on. The model fully plays the role of testing in CBSD. Test driven component composition treats the test as the center, thus makes every step of the development process have the measure criteria. At the same time, with this model framework, we can not only assemble the software system according to own willing quickly, but also generate test cases of the state transition between components and information interaction automatically or semi-automatically. Both the development and testing efficiency are improved under this novel model framework.

87.4 Conclusion

Take CIG and UML state diagram together, and also consider about the thought of contract test, this paper proposes a new type of CBCCIG model, which will be much helpful with the generation of test cases of the state transition between components and information interaction. The future work includes the development of a tool to support automation of the CBCCIG generation, and automated generation of test cases from this model.

References

1. Shang, M., Wang, H., Jiang, L.: The development process of component-based application software. In: 2011 International Conference of Information Technology, Computer Engineering Management Sciences, pp. 11–14 (2011)
2. Fu, L., Sun, G., Chen, J.: An approach for component-based software development. In: International Forum on Information Technology and Applications, pp. 22–25 (2010)
3. Li, L., Wang, Z., Zhang, X.: An approach to testing based component composition. In: International Colloquium Computer Communication, Control Management, pp. 735–739 (2008)

4. Wu, Y., Pan, D., Chen, M-H.: Techniques for testing component-based software. In: Proceedings of the Seventh IEEE International Conference on Engineering of Complex Computer Systems, vol. **2**, pp. 222–232 (2001)
5. Cao, W., Zhang, W., A software test method based on CBD. Comput. Sci. **2,** 156–158 (2005) (Chinese)
6. Lun, L., Chi, X.: Software architecture testing in the C2 Style. In: 2001 3rd International Conference Advanced Computer Theory Engine, pp. 123–127 (2010)
7. Valentini, E., Fliess, G., Haselwanter, E.: A framework for efficient contract-based testing of software components. In: Proceeding of the 29th Annual International Computer Software and Applications Conference, pp. 219–222 (2005)
8. Mohanty, S., Acharya, A.A., Mohapatra, D.P.: A model based prioritization technique for component based software retesting using UML state diagram. In: International Conference Electronics Computer Technology, pp. 364–368 (2011)