Research on Evolutionary Construction Method of Software Service Architecture Based on ACP

Tilei Gao, Ming Yang*, Xiaohui Jia and Tao Zhang
School of Information, Yunnan University of Finance and Economics, Kunming 650221, China
Email: 83300934@qq.com

Abstract. In the open-source computing environment, the software service individuals that constitute the system are highly independent. The construction of software system is changed into combination and iteration, which puts forward higher requirements for the system architecture. Based on algebra of communication process, this paper abstracts the individual software service into service items, and uses the operation rules in communication algebra to realize the construction and modification of software architecture, thus ensuring the preciseness of the construction process and the stability of the architecture. Finally, a system with 12 sub services is used as a case to demonstrate the evolution process of service adding, deleting and replacing operation, which also verified the effectiveness and feasibility of the proposed method.

Keywords. ACP; software service architecture; evolutionary construction method.

1. Introduction

According to the outline of the 14th five-year plan of the people’s Republic of China for national economic and social development and long-term goals for 2035 [1] released in March this year, open-source software has been explicitly listed in the national development plan [2]. In the past 15 years, the core technologies of emerging industries such as cloud computing, mobile Internet, big data, artificial intelligence and block-chain are all built on open-source software. Open-source model is not only a business model, but also an ecological construction method and a complex system construction method [2]. In this open-source computing mode, software services are mostly individuals with high cohesion and low coupling standards. In this case, the combination development methods such as evolutionary and iterative are gradually becoming the mainstream methods of such system development. In this method, the first problem to be solved is the construction and adjustment of system architecture, that is, the evolution of software service system architecture. The construction process of architecture is essentially a process in which a service individual gradually evolves into a complex system composed of multiple service individuals by adding, modifying, deleting and other evolutionary operations. This is the basis and premise of building entity software system. Without high-quality architecture, there will be no high-quality software system, or even a complete software system.

Traditional software evolution studies the system dynamics activities of continuous upgrading and maintenance of software system in the software life cycle [3]. According to the running state of software during evolution, software evolution can be divided into two types: static evolution and dynamic evolution. Static evolution refers to the evolution of software in a stopped state. The advantage of static evolution is that it doesn’t need to consider the change of running situation, and it doesn’t need to deal with any active process. This type of evolution refers to the changes and modifications of the software system during the period when the application is no longer running [4]. In the aspect of evolution...
research, people tend to study dynamic evolution, including object-oriented software evolution methods [5, 6], component-based software evolution methods [7], and Web Services evolution methods [8]. However, there is little research on static evolution and the research on the technology and theory of static evolution also lays a solid theoretical foundation for the study of dynamic evolution [9].

Therefore, this paper takes the evolutionary construction method of software system architecture in open source environment as the research content, and draws on the concept and operation method of progressive item in ACP (algebra of communication processes) [10, 11], abstracts the single service that constitutes the system into service item; Through algebra operation, the architecture can be added, deleted and modified, so as to achieve high-quality software architecture construction and modification.

2. ACP Based Software Service Architecture Evolution Method

Under the open-source software environment, the software system is composed of multiple software sub services. Each sub service completes a function independently. Sub services have atomic properties in the system. There are three kinds of composition relationships among sub services: sequence, selection and concurrency. Therefore, this paper uses the evolution method of composition operation between process items and components in ACP for reference, uses service items to represent the static structure of the service block where the service to be evolved is located, and uses operators to connect various services to describe the relationship between services in the system.

Definition 1 Service Item The service items in the software system are composed of various software services in the system through +(_c), ,(), × and *(_m), in which:

1. +(_c) represents selection operation, ser_1 +(_c) ser_2 indicates that two services cannot be performed at the same time, and subscript c indicates the selection condition;
2. represents sequential operation, ser_1 · ser_2 indicates that ser_2 must be followed by ser_1;
3. represents concurrent operations, ser_1 || ser_2 indicates that two services occur or are in progress at the same time;
4. represents collaborative operations, ser_1 × ser_2 indicates that two services cooperate to form a large granularity service;
5. *(_m) represents a cyclic operation, (ser_1)*(_m) indicates that ser_1 is repeatedly executed, and subscript _m represents the number of cycles.

The schematic diagram of various operations is shown in figure 1.

This paper uses these five operators to describe the composition relationship between the internal services before and after the evolution of software service system. For each software service, only the network structure of the software service is used in the evolution process. Therefore, the operation rules of each operator are as follows:

1. ser_1.P +(_c) ser_2.P = ser_1.P or ser_2.T
ser_1.T +(_c) ser_2.T = ser_1.T or ser_2.T
ser_1.F +(_c) ser_2.F = ser_1.F or ser_2.F
2. ser_1.P · ser_2.P = ser_1.P ∪ ser_2.P
ser_1.T · ser_2.T = ser_1.T ∪ ser_2.T
ser_1.F · ser_2.F = ser_1.F ∪ ser_2.F
3. ser_1.P || ser_2.P = ser_1.P ∪ ser_2.P
ser_1.T || ser_2.T = ser_1.T ∪ ser_2.T
ser_1.F || ser_2.F = ser_1.F ∪ ser_2.F
4. ser_1.P × ser_2.P ⊆ ser_1.P ∪ ser_2.P
ser_1.T × ser_2.T ⊆ ser_1.T ∪ ser_2.T
ser_1.F × ser_2.F ⊆ ser_1.F ∪ ser_2.F
5. (ser_1.P)*(_m) ⊇ ser_1.P
(ser_1.T)*(_m) = ser_1.T
(ser_1.F)*(_m) = ser_1.F
In particular, for the selection operation “+(_c)”, which is to select a qualified service for operation under a specific condition C; For cooperative operation “×”, a new service with larger granularity is generated after the cooperation, which participates in other operations in the form of a new service. Some states, transitions and arcs in the original service are encapsulated in the new service.

3. Case Study
Suppose a software system is composed of 12 sub services, and the structure before and after evolution is shown in figures 2 and 3. Using service items to describe its structure, we can get the following operation expression of pre-evolution system.

Figure 2. An example of architecture evolution (before evolution).

From top to bottom and from left to right, system $S$ is composed of three parts $S_1$, $S_2$ and $S_3$. 

Figure 1. Service composition operation.
\[ S_1 = \text{ser}_1 \cdot (\text{ser}_4 \times \text{ser}_5); \]
\[ S_2 = ((\text{ser}_3 \cdot \text{ser}_5) \| (\text{ser}_5 + (c) \cdot \text{ser}_6)) \]
\[ S_3 = (\text{ser}_2 \cdot ((\text{ser}_7 \times \text{ser}_8) \times ((\text{ser}_9 \cdot \text{ser}_12) + (c) ((\text{ser}_{10*}(n)) \cdot \text{ser}_12) + (c) \text{ser}_11))) \]
\[ S = S_1 \times S_2 \times S_3; \]

There are three types of evolution operations: replace service, add service and delete service.

Replace service operation, that is, replace the operation of old service in the corresponding position in all systems with new service. For example, in this example, \text{ser}_5 and \text{ser}_9 are to be replaced by \text{ser}_{13} and \text{ser}_{14} respectively. After this evolution, the service item changes from \( S \) to \( S' \):

\[ S' = (\text{ser}_1 \cdot ((\text{ser}_4 \times \text{ser}_{13}) \times ((\text{ser}_9 \cdot \text{ser}_{12}) + (c) ((\text{ser}_{10*}(n)) \cdot \text{ser}_12) + (c) \text{ser}_11))) \]

Adding service not only needs to find the service to be added, but also needs to know the relationship between service and context. For example, on the basis of \( S' \), \text{ser}_4 needs to add a new function \text{ser}_{15} after the service; In the environment of service \text{ser}_3, a new application scenario needs to be added. This scenario will lead to the execution of service \text{ser}_6, so a new service \text{ser}_{16} and selection operation need to be added. After this evolution, the service item changes from \( S' \) to \( S'' \):

\[ S'' = (\text{ser}_1 \cdot ((\text{ser}_4 \cdot \text{ser}_{15}) \times (\text{ser}_{13} + (c) \cdot \text{ser}_6)) \times (\text{ser}_2 \cdot ((\text{ser}_7 \times \text{ser}_8) \times ((\text{ser}_{14} \cdot \text{ser}_{12}) + (c) ((\text{ser}_{10*}(n)) \cdot \text{ser}_12) + (c) \text{ser}_11))) \]

Deleting a service, if the service termination node in the graph is deleted directly, its related composite relationship will be invalid automatically; If it is an intermediate service node, it needs to associate its subsequent connected services with its connected services, and delete the irrelevant composite relationship. For example, on the basis of \( S'' \), services \text{ser}_3 and \text{ser}_{11} are useless and need to be deleted. After this evolution, the service item changes from \( S'' \) to \( S''' \):

\[ S''' = (\text{ser}_1 \cdot ((\text{ser}_4 \cdot \text{ser}_{15}) \times (\text{ser}_{13} + (c) \cdot \text{ser}_6)) \times (\text{ser}_2 \cdot ((\text{ser}_7 \times \text{ser}_8) \times ((\text{ser}_{14} \cdot \text{ser}_{12}) + (c) ((\text{ser}_{10*}(n)) \cdot \text{ser}_12) + (c) \text{ser}_11))) \]

![Figure 3. An example of architecture evolution (after evolution).](image-url)
4. Conclusion
The architecture of software services is the framework of the whole system and the basis and premise of the evolution of software services in open-source environment. In this paper, ACP is used as the main formal method, and the single software service that constitutes the system is regarded as a service item. The system architecture before and after the evolution is represented by mathematical operation, which ensures the rigor of logic before and after the evolution, the stability and reliability of the method. Finally, a case study shows that the method is effective and feasible.

Acknowledgments
This research is supported by Yunnan Fundamental Research Projects (grand NO. 202101AT070211).

References
[1] The fourteenth five-year plan for national economic and social development of the people’s Republic of China and the outline of long-term goals for 2035 http://www.gov.cn/xinwen/2021-03/13/content_5592681.htm.
[2] Sun N H 2021 On the spirit of open source Communication of Chinese Computer Society 4 (17) 7.
[3] Barry E J, Kemerer C F and Slaughter S A 2007 How software process automation affects software evolution: A longitudinal empirical analysis Journal of Software Maintenance and Evolution: Research and Practice 19 (1) 1-31.
[4] Vitrand L D E, Zuo X, Yu H L and Zheng S 2017 An approach to evolving internetware application Proceedings of the 2017 2nd International Symposium on Advances in Electrical, Electronics and Computer Engineering (ISAEECE 2017) pp 427-435.
[5] Ma J, Pan J P and Liu F 2018 Object oriented software system evolution model Journal of Jilin University (Engineering Edition) 48 (02) 545-550.
[6] Cheng L, Liu Y and Li T 2017 Component importance measurement method based on node betweenness in software evolution environment Computer Application and Software 34 (10) 29-34.
[7] Zhong L H and Xie B 2014 Research on modeling and obtaining method of component based software evolution information Computer Application Research 31 (02) 401-403.
[8] Zhang Y M, Ni K and Lu J W 2017 Research on automatic evolution method of web service composition based on global dependency network Acta Electronica Sinica 45(02) 267-277.
[9] Sanchez E, Sipper M and Haenni J O 1999 Static and dynamic configurable systems IEEE Transactions on Computers (48) 556-564.
[10] Yang Y C and Chao W S 2020 Model-based systems engineering for sharing economy service systems design using structure-behavior coalescence process algebra Proceedings of International Conference on Human-Computer Interaction pp 240-253.
[11] Bouwman M, Luttik B, Schols W and Willemse T 2020 A process algebra with global variables Logic in Computer Science (322) 33-50.