2017 International Conference on Applied Mechanics and Mechanical Automation (AMMA 2017)
ISBN: 978-1-60595-471-4

One Message Chain Based Cooperation Mechanism for Component Based Software Generation

Yong-kai FAN*, Sheng-le LIU, Xiao-dong LIN and Jian-rong BAI

Department of computer science, China University of Petroleum (Beijing), Beijing, China

*Corresponding author

Keywords: Message chain, System composition, Component communication.

Abstract. During software development, one of the most critical activities for software engineers or software users is the exact and correct description and identification of requirements. This activity involves understanding the problem, which is essential to allow one to define a solution for it. To do this, it is important not only to understand the tasks routinely performed which is part of the problem, but more importantly to understand the domain in which the system will take place. In order to make the software development more comfortable, we present a novel architectural style directed at supporting larger grain reuse and flexible system composition in the special software domain. Under this new architecture, build a software for special domain is easy and feasible. However, one problem we have to tackle with is the data sharing among the various components. In order to make components work together well, we present asynchronous messages for components communication. This paper describes how to composite software components based on the architecture and asynchronous messages in order to eliminate the control and design process in the software development procedure.

Introduction

Software development is becoming more and more difficult for their complex structures, non-functional requirements. Component-based software development (CBSD) reduces the complexity of software development and improves the maintenance by increasing software reuse. CBSD decomposes the system into reusable entities called components [1]. Furthermore, one advantage provided by software architectures is due to the fact that architectural models are constructed using components. As a result, software architectures have emerged as a solution for the development process of complex software systems [2]. Component-based software engineering (CBSE) means building software systems by (re)using pre-fabricated software components each representing pieces of code. Constructing systems in this way is neither a new idea in the area of software engineering nor in other engineering disciplines where this principle has succeeded for many decades. Potential benefits of CBSE include lower development cost, higher software quality, increased productivity, shorter time-to-market and etc. [3].

This paper proposes one way of components communication and introduces how to make components work together in one special field in which application realized by component composition. The remainder of the paper is structured as follows: section 2 introduce the idea of software generation in special field by which component work together to build software, section 3 presents message chain communication among components, and related experimental work in section 4 and finally, section 5 concludes the paper.

Component Based Software Generation

Software is the core and key part of virtual instrument (VI) which is regarded as an important orientation of scientific instruments research field and graphical developing mode for virtual instrument software development [4] combined direct characteristic of data flow chart and of text description feature of high-level language functions. Graphical development mode paves a way for industry software development design for its convenient and intuitive features [4-6]. However, the professional abilities are necessary for using this type of development environment. For example, in the design phase and control phase: line-design among icons and settings for different functions in design phase and management of software program running in control phase. In order to provide a less software professional requirement developing environment for VI or even eliminate any domain specific knowledge, we puts forward one kind of VI software generation architecture based on the [7,8] research. The way we crystallize the software requirement based on the architecture of VI software generation and software generation mechanism of components relationship reasoning.

In our architecture, first thing is packaging code sources or functions to a class. It is one key step for component based software development. Through the packaging process, we divided functions VI need into four categories: data acquisition, data displaying, data storing and data processing. Moreover, those four categories can be subdivided into different classes according to functional differences or characteristics. In the realization of virtual instrument software generation, all the functions belong to those four categories represented in the screen by icons. While users are developing software for VI, users just choose icons appropriately according to the goal of complete functions. Through the generation mechanism which realizes the function combination organizationally meet the demand of generation finally.

Moreover, our architecture contains following partial compositions: components library, software generation system, updating system, generation rules library. The architecture is shown in figure 1. 1) Components library is the collection of certain functional component which may be functions, or can be open source code for certain purpose, but must be wrapper class with declared interfaces which is standard and open. 2) Software generation system, according to the outside information (or we can be called software requirement from users) searches model library, chooses corresponding functional models, connects various models based on generation rules, cooperation different models. 3) Updating system is responsible for update model library by adding, removing or modifying models and for update generation rules library when rules need to be adjusted, modified. 4) Generation rule library provides the default connection rules among models, the mechanism of the interaction relationship among models and decision supporting function. It consists of knowledge library including many rules and match-reasoning mechanism

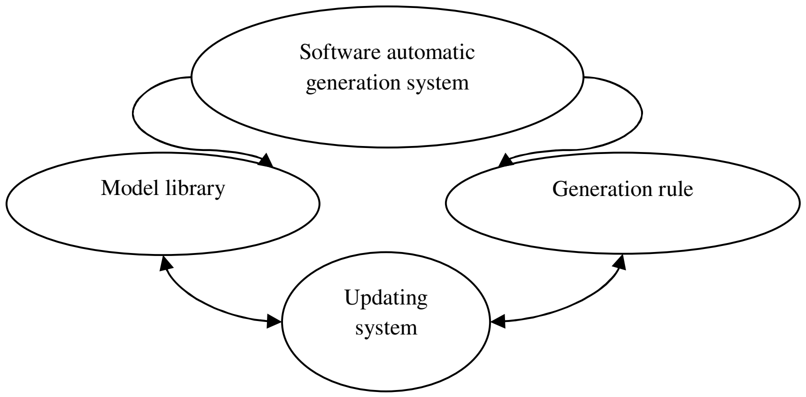

Figure 1. Diagram for the composition of virtual instrument software generation.

Thirdly, we propose the requirement based virtual instrument software generation in our architecture. The basic idea of VI software generations is: getting specific components based on the parsing result from requirement analysis, and using components library to retrieve responding components and using generation rules to assemble components, then making data flow in or flow out among different components. The data inflow or outflow sequence determines the execution order of components and reflects the relationships among components. Furthermore, the final software product is the requirement analysis results from users' viewpoint directly and exactly. The user's requirement analysis result is the core of software and the most critical step in the process of software development. The users' requirement is the ground of software development and taking user's requirement as starting point to generate software can directly reflect the user's real software expectations. Based on this point, how to reflect user's requirement to software is the critical problem.

To meet the reflection of user's requirement, the primary concern of description analysis is which functional components should be chosen for the purpose of generating software described in the process of requirement description. So, the description analysis can be considered as the further explanation. The initial process of description analysis is parsing requirement description and then is the process of building software model. However, one problem we must be faced with is the description may be a vague one, so the parsing result can be divided into two parts: functional part and non-functional part. Functional part means that the description can be converted to specific function components in the software component. Non-functional part is considered mostly as the auxiliary description of software goal means that we cannot do conversion directly. However, non-functional requirement description should be converted in general for the purpose of building the desired software. And if we cannot convert non-function requirement description anymore, we take the conversion process is finished and the description analysis process is over as granted. The software generation process is based on the description analysis result.

Message Mechanism

The message is designed so that different components can cooperate with each other for information sharing. To accomplish this, the first thing we should take into consideration is the composition of message. The figure 2 illustrates the structure of message. In the message structure, the destination component acts as an indicator where the message should go. It typically has a fixed structure, although the Message parameter can be defined with many different structures.

Destination Component
Message Identity
Message Parameter
Source Component

Figure 2. The structure of message.

A place holder for a message is created by setting the composition property of the complex type or group of the element to message parameter. This enables a message to be added within the outer Message, creating a multipart message. Using the Message Identity to hold the pointer of data area, the pointer must be identified earlier in the Message by using a message Identity element. A message identity element is a address element (or attribute) that precedes the message in the model and whose interpret value as property is set to message identity. By default, source component is assumed to be defined within the message which component sends it. This can be overridden by using a message identity, which works in a similar manner to a message identity.

Internal Message Buffer for Component

Taking into account the efficiency of the software and the flow rate of the data stream, internal message buffer is established to avoid a component has become a bottleneck for message traffic flows, as shown in figure 3. Component C is a data display component which has slower processing speed than the component A. Therefore, the component A, which is responsible for data acquisition, has to wait for the notify message from C to make sure there is one free data storage. Therefore, if there is internal message buffer inside the component, the buffer can store the message queue. The message received by component message chronologically sequentially stored into the message queue. In this example, if the component A can apply for buffering multiple memory blocks, the component A can deal with these data blocks to complete data acquisition, so that the data stream can accelerate the flow, there will be no stagnation.

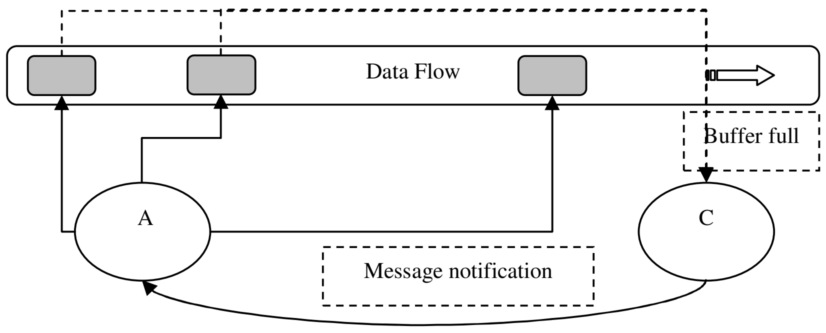

Figure 3. Bottleneck in data stream.

Asynchronous Message Notification among Components

The following example (figure 4) shows the use of the message to share the working state among four components. Assumed that there are four components marked separately as: A, B, C, D; Target software occupies two memory blocks within the data channels form a data stream, the component data acquisition component is C, B of the display component, A is a storage component, D is a processing component. In order to accomplish a predetermined function of the target software, the use of the component type of the component to send a message mechanism. E and F respectively in the data stream of two data blocks located in the message structure the message ID for the data processing P, R to return, then the link between the target software complete message flow of a data stream transmitted by the component shown in figure 4.

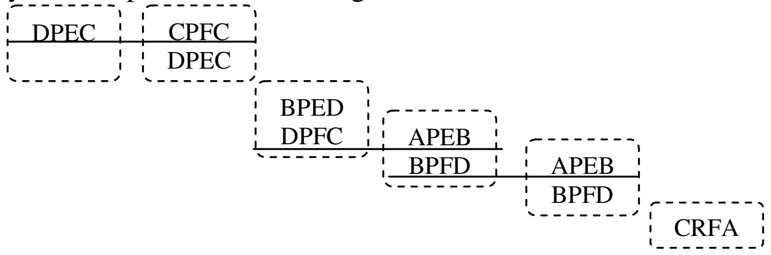

Figure 4. Message based communication among components.

Experiments Works

We use the software generation architecture do some experiments related data channel and message based communication on our self-developed graphic software environment. The test results have displayed in the reference papers [7, 8]. We pick one type of data acquisition component from data acquisition category (data generation component to generate analog signals), a data processing component (to realize FFT transform) under the corresponding category, two data displaying components from data displaying categories (one for the display data generating component generates the analog signal, and the other to show the results after processing data), and then try to generation VI software based on our worked discussed formerly. The experiment results show that the data channel and the message mechanism can work proper for this special field. From the testing experiment, we can conclude that the message based components mechanism works well and efficient, the mechanism can suit for the data stream communication among components.

Conclusions

This paper proposes the need for component and integration frameworks in a special area. Furthermore, the communication among components should not be neglected in order to avoid inconsistent overall the software operation. The generation platform was chosen for its capabilities to integrate components from different providers. The platform was analyzed and proposed to support dynamic components composition. Therefore the communication model and data exchanging model were presented and evaluated. Although the message passing is not fashion thing in terms of architectural design, capabilities, performance and transparency, it helps the platform to compose components together and work well in this special field. This approach results in extensions that do not have any impact on the development of bundles. In fact these extensions are completely transparent to both the providing and the using components. This method can be a reference for other field software generation; however, the achievements are the first step towards the longer term goal of a general shell that common users can develop their own application on the software generation system; further studies are needed as well.

Acknowledgments

This work is supported by Beijing Higher Education Young Elite Teacher Project (YETP0683), Science Foundation of China University of Petroleum (Beijing) (No.01JB0423).

References

[1] Soni N, Jha S K. Component Based Software Development: A New Paradigm [J]. International Journal of Scientific Research and Education, 2014, 2(06).

[2] Ahmad A, Jamshidi P, Pahl C, et al. A pattern language for the evolution of component-based software architectures [J]. Electronic Communications of the EASST, 2014, 59: 1-32.

[3] Herrmann C, Krahn H, Rumpe B, et al. Scaling-Up Model-Based-Development for Large Heterogeneous Systems with Compositional Modeling [J]. arXiv preprint arXiv:1409.6586, 2014.

[4] Goldberg, H., what is virtual instrumentation? IEEE Instrumentation and Measurement Magazine, 2000. 3(4): 10-13.

[5] Horan, S. and R. Wang, Design of a space channel simulator using virtual instrumentation software. IEEE Transactions on Instrumentation and Measurement, 2002. 51(5): 912-917.

[6] Navaee, S. Computing and programming with LabVIEW. 2004. Salt Lake City, UT, United States: American Society for Engineering, Washington, DC 20036, United States.

[7] Fan Yongkai etc., Requirement-driven virtual instrument software automatic generation framework, Journal of Jilin University (Engineering and Technology Edition), v 37, n 3, p 606-610, 2007.

[8] Fan Yongkai etc., Automatic generation of virtual instrument software with orderless automatic connection, Journal of Harbin Institute of Technology, v 41, n 8, p 154-159, 2009.