Architecture of Spacecraft Control Software Based on Component Perception

Wenhui Cui[1*], Bo Su[1], Bo Jiang[1], Le Wang[1]
[1]State Key Laboratory of Astronautic Dynamics, Xi'an 710043, China
E-mail: cwenh_80@163.com

Abstract—**The basic principle and characteristics of component-based software system were studied. According to the characteristics of spacecraft control software, a construction method of spacecraft control software based on component perception is proposed. A variety of functional and perceptual components based on control requirements are designed. The design method based on concrete software components realization is given, which solves the problem of rapid reconstruction of spacecraft control software effectively with high reusability and reliability.**

Keywords-Adaptive software; Component perception; Spacecraft control software

I. INTRODUCTION

With the development of software theory and practice, the new software technologies, including software component, software integration, and software development tools, are more and more mature. They provide better technical support for the optimization of the software structure and has been applied in engineering practice successfully. A lot of software is developed based on Component-Based Software Development (CBSD). The CBSD method improves software quality and development efficiency. It is of great significance to enhance the flexibility and robustness of the software.

Traditional spacecraft control software is independent software developed on a high-performance server (mainly minicomputers). Functional components and interface between modules are developed by different teams according to different task requirements independently. The development platforms and environments are selected according to the space mission requirements and human-computer interaction requirements. The advantages of the traditional software include higher real-time performance, lower coupling between different software, and fewer requirements on the networks. The disadvantages of the traditional software include low algorithm reuse rate, heavy and difficult test workload, poor software reliability, and high coupling between the underlying algorithm and the user interface. In this software architecture, the software system is highly dependent on the relevant hardware and development platform, and needs to be reconstructed constantly according to the change of the hardware and the platform.

In historical studies, the study of program language reflection mainly solves the problem of perceiving software state and adjusting behavior on runtime [1][2]. G. Coulson, M. Roman, A. Mukhija, and F. Kon, et al. accomplished adaptive software through middleware and software framework [3-6]. S. W. Cheng, J. Dowling, J. Floch, and I. Ben-Shaul, et al. accomplished software adaptability based on software architecture technology [7-10]. H. Liu, C. Becker, D. Chefrour, and A. Philippe, et al. studied how to construct the component model of adaptive software [11-14]. These studies make automatic software construction based on adaptive components possible.

The functions of spacecraft control software include space mission analysis, spacecraft attitude control, spacecraft simulation, spacecraft fault diagnosis, and other functions. With the increasing demands of space exploration, the software deployed on different platforms and adapted to different functional requirements. They must be rapidly reconstructed and deployed on new platforms according to the new user requirements, while traditional space software that coupled with hardware and operating systems closely can no longer meet these requirements. A design method of spacecraft control software based on functional components and perceptual components is proposed to solve the problems in this manuscript. This method derives adaptive perception and large-scale reuse theory. By establishing the functional component library and perceptual component library of the spacecraft control, the functional components updated and the calculation accuracy improvement continuously on the premise of the stable software framework and the uninterrupted services. New spacecraft control software can be rapidly reconstructed by using existing functional components and perceptual components according to new user requirements through adaptive perception and large-scale reuse. The new method can ensure the safety, flexibility, robustness and continuous improvement of spacecraft control software.

II. SOFTWARE ARCHITECTURE BASED ON COMPONENT PERCEPTION

In the software architecture based on component perception, high-level business logic specifies the type and architecture of components in the abstract layer, and the abstract layer implementation is accomplished by functional components in the basic layer. The abstract layer eliminates the coupling between the high-level business logic and the functional components of the basic layer, which makes the functional components more stable and reusable. The architecture of the software framework depends on the abstract layer rather than the functional components. The implementation of functional components needs to follow the constraints set by the abstract layer. In spacecraft control software development, various requirements determine the different high-level business logic,

978-1-7281-6494-6/19/$31.00 ©2019 IEEE
DOI 10.1109/ITCA49981.2019.00066

user interface, and calculation accuracy requirements, but the main algorithm modules and functional components are the same. The three-layer architecture of the software is shown in figure 1.

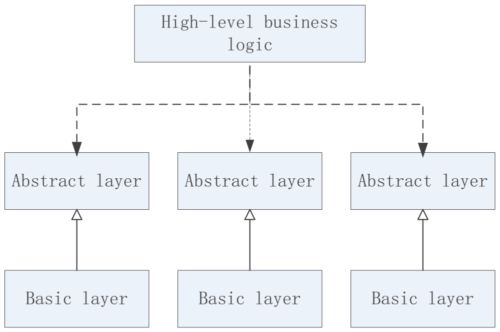

Figure 1. The three-layer architecture of the software

Cheng elaborated that domain software design mainly includes adaptive underlying module design and system external performance adaptive framework design. Adaptive underlying modules that provide general functional support for software are easy to reuse, while adaptive software frameworks are difficult to reuse [7]. In component-based software architecture, the high-level business logic and abstract layer define the software framework together. Since the high-level business logic of potential users is uncertain, the appropriate software framework can be constructed according to different high-level business logic without multiple interventions of users and software designers if the abstract layer is adaptive. The abstract layer based on perceptual components and connectors can accomplish this function well.

The input of the framework based on perception is environment and state perception, and the output is parameter and structure adjustment. The convergence of the environment and state perception form the group perception together, and the group decision and collaboration based on group perception are the main mechanisms of the adaptive perception software. The component-based adaptive software model is shown in figure 2.

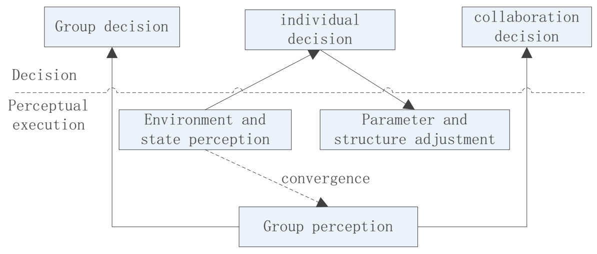

Figure 2. Component-based software adaptive model

Based on the component model of support software environment adaptability fine-grained online adjustment, the key items of software environment adaptability include perception, decision, and execution were encapsulated as independent perceptual components and connectors [7]. The connector is an independent package of the interaction mechanism between the components [15]. The online reconfiguration of the components and connectors are supported by the dynamic software architecture technology. The components were selectively updated when necessary to adjust the ability of the software to adapt.

III. A METHOD TO GENERATE THE SOFTWARE ARCHITECTURE MODEL BASED ON COMPONENT PERCEPTION

The main calculation modules of spacecraft control software include orbit calculation module, attitude calculation module, satellite sensor output calculation module, actuator output calculation module, space environment calculation module, signal transmission error calculation module and so on. The basic functional modules of the software also include basic mathematical calculations and signal processing calculations. The calculation modules calls different functional components according to different business logic and provides different outputs that meet the different precision requirements. According to the perception of different business logic and development environments, perceptual components adaptively select different functional components to compose appropriate software with different functions. This is the basis of spacecraft control software component design.

In spacecraft control software design, model construction can be used to convert specific high-level business logic from natural language to modeling language. There are three steps to generate a software architecture model based on formal specification:

1) The application requirements are input into the software construction system. The perceptual components of the abstract layer identify and analyze these requirements, and realizes the query, matching, selection, and assembly of functional components through the strategy connector to build the adaptive software system model.

2) The component assembly protocol is set up by the existing model. When the protocol conforms to the high-level business logic and makes it effective, the protocol forms the basis of the component syntax matching and assembles the component into service according to the protocol.

3) In the update and reconstruction of software systems, the functional components were queried, matched, selected, deployed, assembled, released, removed, and replaced by perceptual components through strategy connectors based on changes of external requirements. A specific example is that in the assembly of spacecraft control software, the perceptual components match and connect different perturbation and integral models according to different precision requirements to meet different orbital calculation and research requirements of users. The relationship and interaction process between perceptual component, functional component and strategy connector is shown in figure 3.

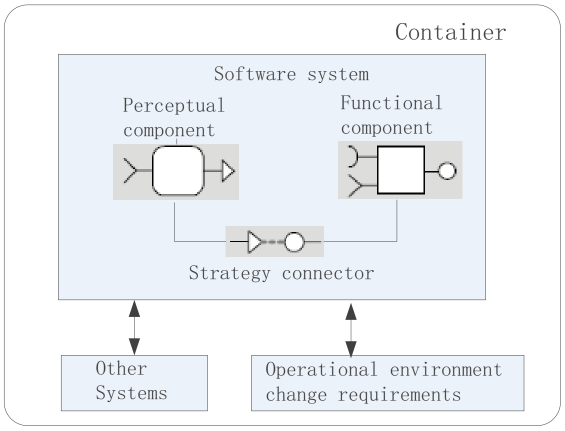

Figure 3. Component perceptual model

The perceptual component and strategy connector are used as the concrete implementation method of the abstract layer between the functional components and the high-level business logic in the method. In this kind of software design system, the component perception layer automatically selects the optimal functional components and realizes different software frameworks according to different high-level business logic.

IV. THE ARCHITECTURE FOR SPACECRAFT CONTROL SOFTWARE BASED ON COMPONENT PERCEPTION

There are three layers in the architecture for spacecraft control software based on component perception: the bottom layer is the basic functional component layer to realizes the development and deployment of various spacecraft control functional components, including messaging management, transport protocol management, performance management, time management, application run management, calculation components and so on. The second layer is the component perception layer, which realizes the matching and assembly of functional components. Its functions are realized by the perceptual component and the strategy connector. The layer includes data management, component release, component matching, component query, component selection, component department, applied assembly and so on. The application construction layer is on the top. The layer includes service framework code generator, user management, service description file generator, applied data representation, service information representation, and process editing and so on. In this architecture, the application construction layer transforms the user requirements into software requirements. The component perception layer reads the software requirements and translate the software requirements to component perception requirements. The component perception layer queries and selects the appropriate functional components, and constructs them to complete software. Component-based spacecraft control software architecture is shown in figure 4.

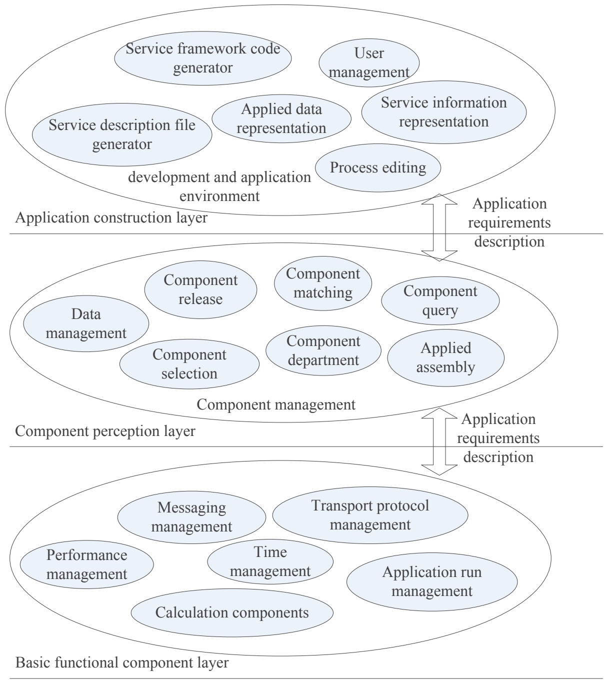
Service framework code generator
User management
Service description file generator
Applied data representation
Service information representation
Process editing
development and application environment
Application construction layer
Application requirements description
Component release
Component matching
Component query
Data management
Component selection
Component department
Applied assembly
Component management
Component perception layer
Application requirements description
Messaging management
Transport protocol management
Performance management
Time management
Application run management
Calculation components
Basic functional component layer

Figure 4. Component-based spacecraft control software architecture

The granularity and the interface of the functional components are the core problems of the component-based model. Appropriate functional component granularity and interfaces support the selection and assembly of perceptual components efficiently, while inappropriate granularity and interfaces lead to redundancy, insecurity and interface mismatch of software functions [16]. The main functional components used to support computing services is listed in table 1:

TABLE I. THE MAIN CALCULATION COMPONENTS OF SPACECRAFT CONTROL SOFTWARE

Module	**Calculation component**	**High-level business logic**
Vector matrix operation	real matrix operation, real matrix inverse of all the selected pivot elements Gauss-Jordan elimination method, Gaussian elimination method	orbit prediction, attitude calculation, attitude determination, orbit determination, orbit control, attitude control
Time conversion	The mutual conversion of various time systems, such as the Julian day, the Gregorian day, the true solar time, the mean solar time, the local time, and the universal time	
Space coordinate system conversion	The mutual conversion of all kinds of space coordinate systems, such as the Earth-centered inertial reference Frame, Earth-centered fixed coordinate Frame, Moon-center inertial reference Frame, Moon center fixed coordinate Frame, Mars-center inertial reference Frame, Mars-center fixed coordinate Frame, etc	
Numerical integrals	Runge-Kutta method, Adams integral method, Chebyshev integral method, Gaussian method for multiple integrals, Monte Carlo method for multiple integrals	

Interpolation fitting	Least-square interpolation method, two-dimensional Lagrange interpolation, Least-square curve fitting	
Perturbation calculation	non-spherical gravitational perturbation, atmospheric damping perturbation, third problem of gravitational perturbation, solar pressure perturbation, tide perturbation	
Control calculation	orbit control prediction, GEO satellite position keeping control, sun-synchronous orbit descending node control, spin stabilization satellite attitude adjustment control	
Signal processing	Kalman filter, Wiener filter, classical time-frequency analysis technology, modern time-frequency analysis technology, power spectrum estimation	attitude determination, orbit determination

In the architecture for spacecraft control software based on component perception, the software can be constructed or reconstructed quickly and adaptively. The focus of software development shifts from how to process data to how to achieve user interaction. Software testing can also skip over the testing of functional components and enter the user interaction, data interface, and other high-level testing directly. This will greatly improve the development efficiency and stability of spacecraft control software.

In the life cycle of software, many changes will lead to the updating or redevelopment of functional components. For example, the mathematical models of signal processing and numerical integration will achieve new development, the accuracy of various basic astronomical data will be improved, and astronomical parameters will also change with time. Since each functional component is relatively independent, the above problems can be solved by releasing the component on the premise of maintaining the relationship of components and interfaces. The reconstruction of the software relies on the perceptual component that does not need the intervention of software users and developers at all. The testing of the above components focus on the component implementation, and the software interface stability can be maintained.

V. CONCLUSION

The architecture of spacecraft control software architecture based on component perception is deduced in the manuscript. The main functional components and perceptual components are defined and described according to the characteristics of components. The application practice of spacecraft control software shows that the development system based on component perception can meet the requirements of the rapid reconstruction and maintain the flexibility and stability of the spacecraft control software effectively based on the new requirements. Since the communication protocols of the architecture for spacecraft control software based on component perception between layers support cross-platform development thoroughly, this software architecture not only supports software developed based on minicomputer but also supports control software design based on the cloud platform and service-oriented software architecture.

ACKNOWLEDGMENT

Thanks to Bo Su for his help in writing this article and editing the computing component, and to Bo Jiang and Le Wang for their contributions to this article.

REFERENCES

[1] B. C. Smith, "Reflections and Semantics in a Procedural Language". Cambridge, MA: Massachusetts Institute of Technology, 1982.

[2] P. Maes, "Concepts and Experiments in Computational Reflection," ACM Sigplan Notices, vol 12, pp. 147-155, 1987.

[3] G. Coulson, G. Blair, P. Grace, F. Taiani, et al. "Generic Component Model for Building Systems Software," ACM Transactions on Computer Systems, vol. 1, pp. 1-42, 2008.

[4] M. Roman, C. K. Hess et al. "A Middleware Infrastructure to Enable Active Spaces,". IEEE Pervasive Computing, vol. 4, pp. 74-83, 2002.

[5] A. Mukhija, Glinz M. "Runtime Adaptation of Applications through Dynamic Recomposition of Components," Proceedings of International Conference on Architecture of Computing Systems, pp. 124-138, 2005.

[6] F. Kon, M. Román, et al. "Monitoring, Security, and Dynamic Configuration with the Dynamic TAO Reflective ORB," Proceedings of Middleware Conference, pp. 121-143, 2000.

[7] S. W. Cheng, "Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation," Pittsburgh: Carnegie Mellon University, 2008.

[8] J. Dowling, V. Cahill. "The K-Component Architecture Meta-model for Self-Adaptive Software," Proceedings of International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, pp. 81-88, 2001.

[9] J. Floch, S. Hallsteinsen et al. "Using Architecture Models for Runtime Adaptability," IEEE Software, vol. 2, pp. 62-70, 2006

[10] I. Ben-Shaul, O. Holder, "Dynamic Adaptation and Deployment of Distributed Components in Hadas," IEEE Transactions on Software Engineering, pp. 769-787, 2001.

[11] H. Liu, M. Parashar et al. "Component-Based Programming Model for Autonomic Applications," Proceedings of International Conference on Autonomic Computing, pp. 10-17, 2004.

[12] C. Becker, M. Handte et al. "PCOM-A Component System for Pervasive Computing," Proceedings of International Conference on Pervasive Computing and Communications, pp. 67-76, 2004.

[13] D. Chefrour. "Developing Component Based Adaptive Applications in Mobile Environments," Proceedings of ACM Symposium on Applied Computing, pp. 1146-1150, 2005.

[14] A. Philippe, L. Jér Me. "CompAA: A Self-Adaptable Component Model for Open Systems," Proceedings of International Conference and Workshop on the Engineering of Computer Based Systems, pp. 19-25, 2008.

[15] N. R. Mehta, N. Medvidovic, S. Phadke. "Towards a Taxonomy of Software Connectors," Proceedings of International Conference on Software Engineering, pp. 178-187, 2000

[16] K. K. Lau, Z. Wang, "Software Component Models," IEEE Transactions on Software Engineering, vol 10, pp. 709-724, 2007.