A Software Dual-bus Architecture Suitable for Distributed Real-Time Embedded System

Chenglie Du, Gang Li, Cuiye Song, Bao Zhou
Computer College, Northwestern Polytechnical University
Email:ququ_er2003@mail.nwpu.edu.cn

Abstract

Distributed, real-time and embedded (DRE) systems with high confidence demands can benefit from dynamic reconfiguration technology to adjust and reconfigure system resources at runtime in response to the running state and normal or abnormal events. Reconfiguration is also the key technology for mission computing and the software system dynamic evolution. Monitoring the key objects of the system and taking action accordingly is a feasible mechanism to realize reconfiguration. Software sensors have been used to monitor the events, which helps to guarantee the safety of the DRE system. This paper describes a component-based software dual-bus architecture (SDB), in which one software bus is dedicated to software sensors data transmission. It is expected to guarantee the real-time while supporting the monitoring of the running system information and the dynamic reconfiguration. In the end, we do some simulation of experiment evaluating our SDB architecture in the context of the INS/GNSS software system.

Keywords:

DRE System, Software Dual-Bus, Dynamic Reconfiguration, Software Sensor, Software Architecture

1 Introduction

Mission scheduling and dynamic mode change are growing in importance in distributed real-time and embedded (DRE) systems like airborne command and control systems. Both the functionalities and the scale of DRE systems are getting more complex, as more computing devices are networked together to help committing critical missions. This type of system is characterized by stringent quality-of-service (QoS) requirements, such as reliability, safety, security, survivability, fault tolerance, and real-time, as described in [1]. Safety is especially important, for that big tragedy can be caused in a very short period of time. So system fault must be treated in a timely manner before it becomes a real hazard. When a system fault is detected, it must be treated immediately, so the system can continue working. Reconfiguration is the key technology for mission computing and the software system dynamic evolution. Monitoring the key objects of the system and taking action accordingly is a feasible way to realize dynamic reconfiguration.

Usually, the devices in running system are monitored by software sensors [2]. Software sensors are used in the components of the system to reflect the values of variables in the component with embedded software sensors. As the system grows complicated, software sensors and the communication between them becomes additional load that may affect the real-time of the system. This paper presents a component-based software dual-bus architecture (SDB) that separates the data from normal workflow and software sensors, so that we can treat different data in different scheduling policy and task priorities. Then, we apply the architecture to the INS/GNSS software system, simulate the system in Rhapsody and finally give the simulation result and explain the reconfiguration course of the system.

This paper is structured as follows. Section 1 introduces the whole paper. In section 2, we present a

dynamic reconfiguration model; describe the dynamic reconfiguration process and its restrictions in detail. To meet the restrictions well, we present a software dual-bus architecture and describe its mechanism in section 3. Section 4 offers the application of the SDB, and the simulation of it. Finally, section 5 offers concluding remarks and describes future work on the SDB and its application to DRE systems.

2 Dynamic Reconfiguration

Dynamic reconfiguration is not a new problem [3]. By dynamic reconfiguration we mean a run-time modification of an application's architecture [4]. In Polylith [5], the authors proposed an approach where the reconfiguration can only be achieved at predetermined instants. In [6], in the self-healing system, the reconfiguration is done by Reconfiguration Manager at two levels: component level and connector level between components. The reconfiguration of anomalous objects in a component is performed by Reconfiguration Manager in the healing layer of the component. All these approaches, and many others, are based on non-standard components models. Other methods have been proposed only for one of the standardized component models [7] [8].

We need to modify the configuration of DRE system to adapt to the changes and achieve the dynamic evolution capability [9]. In the traditional way, the reconfiguration work is done off the line, which is called static reconfiguration. It can't be accepted by neither commerce benefit, nor system safety because of the high operation cost and inconvenience. Dynamic reconfiguration is done at run time, which allows the new configuration operation to the part that is not suitable, but not to the whole system [10]. It adds relevant constructs to build and reconfigure applications dynamically by aggregation and replacement of components and objects.

2.1 The Main Idea of the Dynamic Reconfiguration

Dynamic reconfiguration are usually realized through particular architecture [11]. The model of the dynamic reconfiguration includes three levels: Application System, Dynamic Reconfiguration Layer and Configuration Decision Maker. And the Dynamic Reconfiguration Layer includes three parts: Sensors Analyzer and Stimulator module, Components Information Library, and Dynamic Reconfiguration Drive Module, as shown in Figure 1.

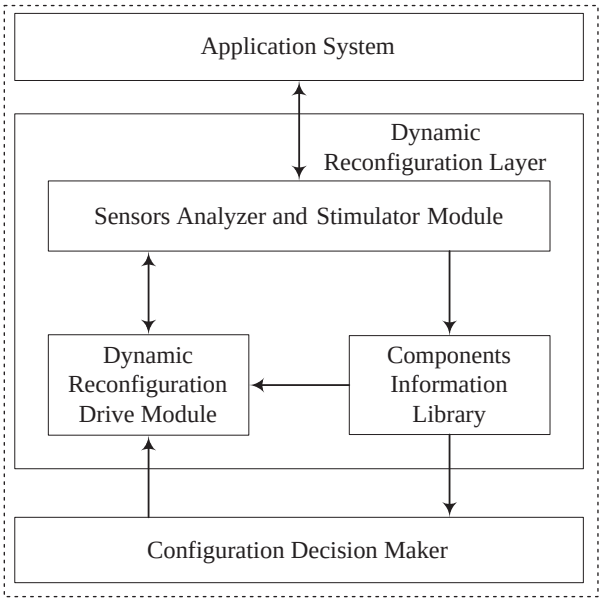

Figure 1: dynamic reconfiguration model

Application system's structure and behavior information is put into the Components Information Library by the Sensors Analyzer and Stimulator Module, so the higher decision-making level Configuration Decision Maker can obtain the whole view of the application system, and make corresponding dynamic reconfiguration decisions based on the analysis of the system and the target of system. These decisions are then sent to the Dynamic Reconfiguration Layer to be carried out. The Dynamic Reconfiguration Drive Module in Dynamic Reconfiguration Layer will attempter the components' behaviors according to the system architecture and semantic information in the Components Information Library at proper time.

To guarantee the consistency and certainty of the reconfiguration, we use independent software sensors as detectors which are associated with the running system's components. Software sensors monitor the system's states, including inner states of the software system and states of the environment. Based on the information, the Configuration Decision Maker makes the

decision whether to do the reconfiguration or not. If a reconfiguration is needed, the Configuration Decision Maker obtains the current configuration information first, and then it queries its subcomponent Blueprint, and finally gets the reconfiguration operation sequences and sends them to the Dynamic Reconfiguration Drive Module.

2.2 The Process of Reconfiguration

When a component fails, the software sensor embedded affirms it; the dynamic reconfiguration model starts the new configuration, and the system continues its normal functions. We describe the reconfiguration process as follows:

(1) Establishing new components. When there are not enough resources for the operation, or the environment changes, new function component will be established.

(2) Deleting old components. When environment gets changed, redundant component will be deleted from system.

(3) Connecting new components. The aim of establishing the connection between components is to permit the communication of them.

(4) Disconnecting old components. When a component is deleted from system, the connection associated with it will also be deleted.

2.3 Dynamic Reconfiguration's Restrictions

There are three restrictions as follows when reconfiguration is done.

(1) Consistency restriction. Consistency restriction can also be called integrity restriction; it expresses the system's correctness: action consistency, architecture consistency and state consistency.

(2) Local restriction. Local restriction means that the process of dynamic reconfiguration is done in a finite range, or it may make the interrelated components work abnormally.

(3) Real-time restriction. The reconfiguration process must be done in a definite period of time.

The restrictions before-mentioned put forward higher demand for DRE system, but single software bus can't satisfy these restrictions well. We should ensure the real-time property in the process of reconfiguration, a more appropriate architecture is needed.

3 The Software Dual-bus Architecture

The concept of software bus comes from the computer' s hardware bus architecture. Complex system consists of components as their functionality module, which may contain subcomponents as their subsystem. As a connector component, software bus connects different components of the system together. The monitor and failure detection process requires mature component technology, proper architecture, effective software state monitoring and reasoning mechanism to perform the information gathering and analysis task. As the scale becomes larger, and the structure more complicated, the amount of sensor data becomes larger, which is a big threat to the system' s real-time property and safety. Effective control of the data transport is very important.

3.1 The Design of the Software Dual-bus Architecture

Architecture is the blueprint of the whole course of the system developement [12]. We design two software buses as communication middleware for the components of the system. As shown in Figure 2, four components are introduced: the Sensors Analyzer, the Reconfiguration Operator, the Resource Advisor, and the EmSS-Component representing the System's component with embedded Software Sensor component.

A software sensor is embedded in a software component by putting a sensor macro in the source code just after each assignment sentence to the variable monitored. The macro will be replaced by its corresponding code section when the source code is processed by the preprocessor before compiled. The software sensor is developed separately with the components where its variable exists, just to meet the need of the monitor task. Once the value is changed, the sensor gets it, and

sends the data to the Sensors Analyzer.

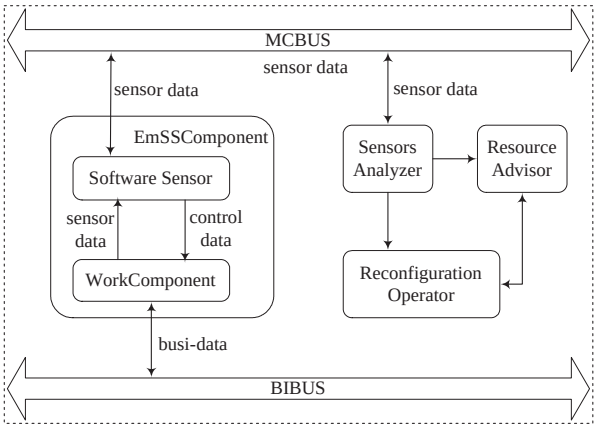

Figure 2: the software dual-bus architecture

The Sensors Analyzer gathers information from sensors, and analysis the data, maybe including the history data, to judge whether an event should be produced and sent to the Reconfiguration Operator. When receiving an event from the Sensors Analyzer, the Reconfiguration Operator takes action as described in last section. The Resource Advisor gets important data from the Sensors Analyzer and the Reconfiguration Operator to maintain the available resource information. It also provides resource information to the Reconfiguration Operator when needed.

MCBUS is the Monitor and Control Bus, while BIBUS is the Business Data Interaction Bus. The MCBUS is devoted to the monitor data transportation, while BIBUS is used for other data transportation. As Qos-enabled component middleware, they shield the communication mechanism to the upper level component; simplify the trustworthy development and reconfiguration. The architecture is expected to adopt tailorable middleware to guarantee safety and real-time properties without restriction to the framework of the existing middleware like CIAO [13]. And, once a fault happens, it is expected to be monitored suddenly, and both the Sensors Analyzer and the Reconfiguration Operator should response immediately.

3.2 The Real-time of the SDB

In real-time system, both the computing and action must be finished before deadline. It should response to the stochastic events timely, and take action immediately. The correctness depends on both the correctness and timeliness of the result [14]. Computer system is a discrete time system, with hard triggers moving forward the time. In such system, events can be treated in two modes: periodic enquiries mode and event/interruption mode, which is also known as time-driven and event-driven. Real-time systems can be event-driven or time-driven, or both.

The event-driven mode can react to the system events timely while saving many CPU periods, but it can't promise the stable running when encountering heavy loads. The time-driven mode is different. It is able to prevent the instabilities, but requires complex control policies. A mixed mode is chosen, absorbing the advantages of the above two modes. We adopt preemptive variable-priority scheduling mixed with time-sharing scheduling as a solution to guarantee the real-time property of the multi-task system. As shown in Figure 3, tasks t1, t2 and t3 with the same priority will share the CPU in turn. Tasks t4 and t5 with higher priority can interrupt the running tasks.

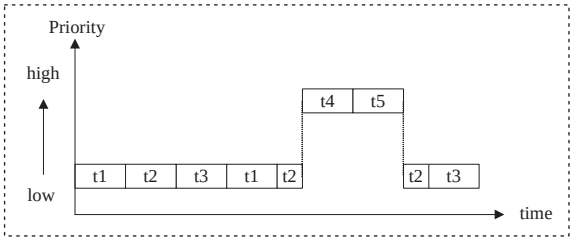

Figure 3: the scheduling policy of the Sensor Analyzer

3.3 The Advantages of the SDB

Complex avionics system consists of many functional components with fewer interactions between them. Resources spreading on different nodes are controlled by components. One component may refer to several computing resources and network resources, but the communication between different components may be shielded by the bus connecting them. In large scale avionics system, there are several advantages using the software dual-bus architecture.

(1) Scale extensibility. When a new functional component is introduced, it can be easily added to the system architecture. The bus is just an abstract communication protocol, so several components connected by

the bus can form one complex component.

(2) Qos-enabled Control. We only care about data related with safety in this paper, but actually, the Qos of DRE system include many aspects. In the SDB architecture, since data about Qos can be treated separately, more aspects of Qos needs can be considered.

(3) Real-time guarantee. Two software buses can separate Qos data and business data, and thus resolve the interaction of huge amount of data elegantly by scheduling according to the priorities of events.

4 The Application of Software Dual-bus Architecture

4.1 The Instantiation of Software Dual-bus Architecture

In the basis of interrelated work above, we take an INS/GNSS system for example. We collect monitor information via software sensors which are embedded in the operation modules, and send these data to system's dynamic reconfiguration module to achieve dynamic reconfiguration.

Integrate the event-driven style and design idea of double bus architecture, we get an instance of the architecture. The instantiation of this architecture embodiment in operation component and software sensor data transaction component.

The instantiation of operation component behaviors as: adding software sensor in the INS/GNSS system to form an operation component embedded software sensor.

The instantiation of software sensor data transaction component behaviors as: in monitor data transaction with Pipe-filter Style, one module's output due to next module's input. This component acts as a filter, and connects between modules act as pipeline.

4.2 The Simulation & Validation of the Dynamic Reconfiguration in INS/GNSS System

We take Rhapsody as the test environment. Rhapsody follows the UML standard, and it's a real-time embed oriented tool, software engineers can complete their work from requirement to outline design, even to code [15] [16]. Figure 4 shows part of the simulation & validation of INS/GNSS system's dynamic reconfiguration.

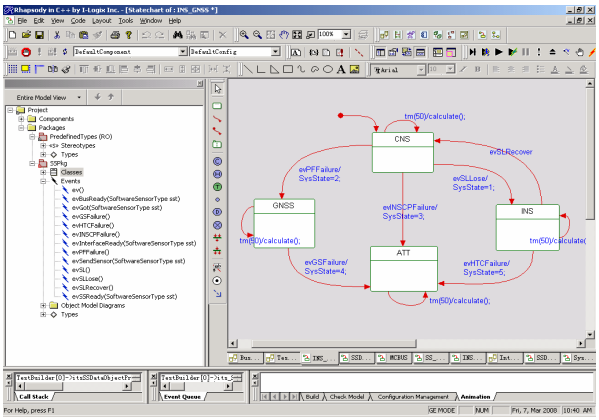

Figure 4: the state chart of the INS/GNSS

When creating the project of INS/GNSS in Rhapsody, we abstract the software dual-bus architecture as: INS/GNSS class, software sensor class, interface class, bus class and data transaction class. We add the cell-detect with time-driven mode to the INS/GNSS state chart in the condition of 50ms, it springs software sensor period. There is a message queue in the event manager, IN_PORT(ssget)–GEN(event) is a receive accept buffer, it sends message to event instantiation object through OUT_PORT(sssend)–GEN(event), the latter fetches the monitor data by ReadSSData(), and last, we realize the information transfer between classes.

As for the system level failure inject, we use the abnormal spring mechanism. Table 1 shows the INS/GNSS systems' simulate failure event at run-time.

Table 1: Running failure events in INS/GNSS

Event Name	Description
evSLLose	satellite signal lost event
evPFFailure	INS/GNSS platform failure
evINSCPFailure	INS/GNSS main program error
evGFFailure	satellite software platform fault
evHTPFailure	aviation computing program error

In the simulation experiment, we injected 50 failures of each kind described in Table 1, and all the fail-

ures were successfully monitored and sent to the Sensors Analyzer. Information from MCBUS is managed by the Sensors Analyzer and then the Reconfiguration Operator. The Reconfiguration Operator obtains the reconfiguration request event from the the Sensors Analyzer, then it refers to the Resource Advisor to execute corresponding arithmetic and configuration strategy.

In the dynamic reconfiguration process, mode switch signal is obtained from interface in scheduler class, and corresponding state switch event is sent to GNSS class, but if this event is not defined in the GNSS class, the dynamic reconfiguration process will be started. At last we will disconnect the scheduler class and INS class, delete the INS class, create the GNSS class and connect the scheduler class and GNSS class.

5 Conclusion

DRE systems are getting more and more complex, the dynamic reconfiguration of the DRE system has advantages of high adaptability and persistent usability, but for DRE system, there are tremendous challenges to achieve the dynamic reconfiguration. We present a software dual-bus architecture on the basis of component technology, and also we embed software sensors in the system components, collecting the monitoring information of the running system, thus reducing the influence on the DRE system's real-time property. Software sensors have been used to monitor the events, which help to guarantee the safety of the DRE system. To protect the real-time of the DRE system against the additional cost by software sensors and their communication, SDB can guarantee the real-time while supporting the monitoring of the running system information and the dynamic reconfiguration. In the end, we take the INS/GNSS for example, instancing the software dual-bus architecture. It proves that this architecture is conducing to the DRE system's safety.

DRE system has many characteristics, such as reliability, safety, security, survivability, fault tolerance and real-time etc, we should satisfy all the requirements simultaneously, or the DRE systems will loss its advantages. This paper makes some contributions to part of them—safety and real-time. The SDB not only provides a solution to the state monitoring of the DRE systems, bus also suggests a helpful model to the DRE system information communication. We will keep on researching on it and other high confidence characteristics of DRE systems.

References

[1] H. Chen, J. Wang, and W. Dong, "High Confidence Software Engineering," vol. 31, no. 12A, Dec. 2003.

[2] H. Mary, "Using software sensors for migrating from classical simulation systems towards virtual worlds," in *Engineering of Computer-Based Systems, IEEE Conference Proceedings*, 1997, pp. 105–112.

[3] R. Fabry, "How to design system in which modules can' be changed," in *Proc. 2 nd Int, Conf. on soft. Eng.*, 1976, pp. 470–476.

[4] P. Hnetynka and F. Plasil, "Dynamic reconfiguration and access to services in hierarchical component models," in *Proceedings of CBSE 2006*, Jun. 2006, pp. 352–359.

[5] J. M.Purtilo, "The polylith software bus," *ACM TOPLAS*, no. 1, pp. 151–174, 1994.

[6] M. E. Shin and J. H. An, "Self-Reconfiguration in Self-Healing Systems," in *Proceedings of the Third IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASE'06)*, Mar. 2006, pp. 89–98.

[7] J. Almeida, M. Wegdam, M. V. Sinderen, and L. Nieuwenhuis, "Transparent Dynamic Reconfiguration for CORBA," Rome, Italy, Sep. 2001.

[8] J. T. Rutherford, K. Anderson, A. Carzaniga, D. Heimbigner, and D. A. Wolf, "Reconfiguration in the Enterprise Java Bean Component Model," Department of Computer Science, University of Colorado, Tech. Rep. CU-CS-92S-OI, Dec. 2001.

[9] C. Li, Ph.D. dissertation, ZheJiang University, 2005.

[10] Antonio • D, Chritiaan • J, and Pradeep • K, "Organization and Selection of Reconfigurable Models," in *Simulation Conference Proceedings*, vol. 1, Dec. 2000, pp. 386–393.

[11] E. Alden and J. C., "Assured Reconfiguration of Embedded Real-Time Software," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2004, pp. 367–376.

[12] M. Shaw and D. Garlan, *Software Architecture Perspectives on an Emerging Discipline*. New Jersey: Prentice Hall, 1996.

[13] V. Subramonian, G. Deng, and C. Gill, "The design and performance of component middleware for Qos-enabled deployment and configuration of DRE systems," *The Journal of Systems and Sofware*, no. 80, pp. 668–677, 2007.

[14] Z. D. Kaynar, N. Lynch, R. Segala, and F. W. Vaandrager, "Timed I/O automata: a mathematical framework for modeling and analyzing real-time systems," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, 2003, pp. 166–177.

[15] I. Schinz, T. Toben, C. Mrugalla, and B. Westphal, "The Rhapsody UML Verification Environment," in *Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM'04)*, Sept. 2004, pp. 174–183.

[16] A. Egyed and D. S. Wile, "Statechart simulator for modeling architectural dynamics," in *Proceedings. Conference on Software Architecture, Working IEEE/IFIP*, Aug. 2001, pp. 87–96.