Design and Implementation of Software Bus Based on Object Model Operating System

RuiLiang Zhou¹, Yongchao Tao¹*, and Songhao Piao¹
¹ Shenzhen Academy of Aerospace Technology, Shenzhen, China

*Corresponding author e-mail: taoyongchao@chinasaat.com

Abstract. At present, the commercial operating system VxWorks is the first choice of leading international intelligent equipment manufacturers. They implement their own equipment controller based on VxWorks. However, VxWorks is not specifically designed for the control field, so there are many deficiencies in the control field, such as: using a dynamic model, dynamically calling various resources at runtime, making the system less predictable and real-time. Aiming at the typical application scenarios of the object operating system, this paper gives the overall software architecture of the object bus. The designed object bus provides a transparent mechanism for communication between components. In view of the highly dynamic adjustment of the objects in the object bus, this article has designed a corresponding set of migration mechanisms for the object bus to solve the various negative effects of cache invalidation after dynamic adjustment. In the last part of the article, the functional verification of the implemented object bus is carried out, and the analysis and summary are carried out.

1. Introduction

With the development of computer technology and software engineering, network interconnected intelligent equipment has been fully applied to all aspects of our lives, but with the crazy increase in the number of application software and hardware equipment, many software development and applications have never been or rarely appeared before. The design problems of the above gradually exposed. In order to enable network-interconnected intelligent equipment to have the ability to negotiate, collaborate and self-organize, the project to develop an object model-based operating system for network-interconnected intelligent equipment was initiated. The innovation of this system is to encapsulate software functions and communication ports as objects.

The traditional IPC is used as the communication port (ports) of the object. In addition to attributes such as priority and scheduling mode (whether it can be preempted), an object also includes ports such as input ports and output ports. The object is the basic unit of scheduling operation, which can be independently addressed on the network and communicate transparently with other objects through its own port.

This article is aimed at the design and implementation of the object bus part. In the traditional development model, a software application or system and its hardware components often constitute a relatively closed whole. A software system realized through demand analysis and design. This type of software system combines fixed functional modules and provides specific interfaces that have been designed to run on specific hardware devices.[1] Many functional modules cannot be individually deleted, upgraded or replaced [2]. For other software systems, even if they use the same language and
run on the same hardware device, it is difficult to use the functional modules of the software system. Different software systems are like strangers, completely isolated, independent of each other, and poor in scalability [3].

The software bus provides a transparent mechanism for communication between components. The bus provides a consistent interface specification for the outside world. Each newly added component must register its functional service information on the bus, and cancel its functional service information when exiting. If a component wants to request the service of another component, the bus is responsible for querying the functional service information database, finding the component that provides this service, and sending this service request [4]. The software bus follows the open design principle and provides "plug and play" services for the software system [5]. Through the communication module in the bus, as long as it follows the "bus" interface standard, any application program, regardless of its function, can be directly integrated into the system environment [6].

2. Design of Soft-Bus in Object Operating System

2.1. The overall function design of the object bus

Through the demand analysis of the object bus in the previous section, we can know that the main function of this project is to provide the positioning and information exchange capabilities between objects for the object operating system. Considering the three basic functions implemented: communication function, caching function, and recording function of the object bus, and their need to cooperate with each other to provide services for the realization of the upper layer, the object bus adopts a hierarchical functional design, as shown in the following figure:

![Object bus functional hierarchy diagram](image)

The function of the upper layer is the function that the object bus provides services to the object operating system, and the realization of these functions requires the support of multiple functions at the bottom.

From the perspective of module division, all functions of the object bus can be divided into three major modules: the recording module, the communication module, and the cache module according to the categories of the basic functions that they rely on.

2.2. Structure Design of Object Bus

After completing the overall functional design of the object bus, the next step is to design the structure of the object bus. The structure design of the object bus is to abstract the overall object bus and provide corresponding guidance and ideas for the realization of each part of the object bus. It divides the functions of the object bus into those components of the object bus structure, and then describes the connections, interactions, and constraints between the components. The object structure
design embodies the correspondence between the software functions and structure of the object bus, thereby providing some support for the overall design decision of the object bus. The overall structure design of the object bus is as follows:

![Object bus structure diagram](image)

**Figure 2. Overall framework of object bus**

As shown in the figure above, the overall structure of the object bus is divided into six components: object services, object bus records, receiving ports, sending ports, object records on the bus, and communication layer. Outside the object bus structure is a component named "object creation/management", which is a module of the object operating system and the target of the service provided by the object bus. The communication layer provides the underlying implementation for the receiving port and sending port components. The receiving port, sending port and object record components provide the basis for the realization of the communication function of the object service component. Object service is a component that the bus provides external services, and provides bus-related services for the object creation/management module in the object system. The object creation/management component uses the corresponding functions of the object bus by calling the interface provided by the object service component.

1. **Communication layer components:**

   The main function of the communication layer component is to use the corresponding communication protocol to send the encapsulated information to the specified address. Provide sending and receiving related APIs to call sending and receiving components, establish connections between different buses, and realize the most basic communication functions. Its creation and use are completely controlled by the sending port component and the receiving port component.

2. **Send port component:**

   The main function of the sending port component is to play a link between the previous and the next; the link provides the corresponding interface service of the sending function for the object service, and obtains the related data; the next calls the communication layer interface to play the communication function, and the obtained data is processed Encapsulate, and then send the encapsulated data to other designated buses through the communication layer. There are two sending methods, single sending and group sending. Single sending refers to sending to a designated bus or object; group sending refers to sending to all bus members connected to the bus network. The design of the sending port component adopts the principle of isolation, which embodies the hierarchical design concept, isolates the upper-layer component from the bottom-layer implementation, and only exposes the necessary interfaces to provide corresponding services.

3. **Receiving port components:**

   The receiving port component is a component corresponding to the sending port component. There are many functions between them that correspond to each other. Therefore, the main function of the receiving port component also plays a role of connecting the link between the above and the following: providing the corresponding receiving function for the object service component Interface service, start the receiving module and feed back the processing result of the received data; then call the
communication layer interface, exert the communication function, obtain the received data, then analyze the data, and perform corresponding operations according to the analyzed structure.

(4) Object record components on the bus
The main function of the object recording component on the bus is to record the object information on the bus, to cache the position information of the object queried through object positioning, and to record the information of other objects that have a dependency relationship with the object. There are two kinds of dependencies here, one is active communication, that is, the object that the object actively establishes a connection through the sending component; the other is passive communication, that is, the source object that the object receives messages through the receiving component.

(5) The record component of the object bus
The main function of the recording component of the object bus is to record some information of the object bus in the object operating system. These features and information mainly include: other bus address information and task information for recording and maintenance of the object bus itself.

(6) Object service components
The main function of the object service component is to provide all the services of the object bus for the object creation/management module of the object operating system, and to provide the object positioning and message exchange functions for the object operating system. The function of the object service module is realized by the combination of the sending port component, the receiving port component, the object recording component on the bus, and the recording component of the object bus.

3. Implementation of Soft-Bus in Object Operating System

3.1. The overall function realization of the object bus
The main function of the object bus is to provide the positioning and information exchange capabilities between objects for the object operating system. There are three major functional modules: recording module, cache module, and communication module. These three functional modules are internally composed of three basic functions including recording function, cache function, and communication function, as well as more than a dozen that mainly rely on these three basic functions. It consists of several functions, these functions include: bus initialization, object registration, entering the bus network, exiting the bus network, object positioning, data port communication, event port communication, signal port communication, output port communication, and object migration.

3.2. Implementation of Object Bus Recording Module
The record function of the object bus is mainly composed of three parts: object record, task record and bus record.

(1) Object registration
```c
void sbus_regist_object(struct object_regist *o)
{
    Get the pointer of the object linked list head;
    While(The pointer does not point to the end of the linked list)
    {
        Pointer moves back;
    }
    Add object record information;
}
```

(2) Find and add tasks
```c
struct task_findIPbyName* searchTask(struct object_regist* o, char name[20])
{
    Get the head pointer of the task list;
    Traverse the task list to compare tasks;
    If(Task already exists)
    {
        Return point;
    }
```
3.3. Implementation of Object Bus Cache Module

The cache uses the structure of an array and a linked list. For ease of use, the blank cache space is uniformly placed at the end of the array, which makes it easier to obtain when adding.

(1) Increasing the cache is the most basic function of the cache module, but it is also the core function, because it involves the specific implementation of the cache update strategy. The performance, stability, and availability of the cache largely depend on the algorithm to increase the cache.

(2) Lookup cache is a part of the service function of the cache module. According to the key information, the cache is searched for whether there is a cache that meets the requirements, and the cache information is returned.

(3) There are two main parts in the way of deleting the cache. One is to find the cache address to be deleted according to the content to be deleted; the other is to find the cache address that should be released in the order of the array, and delete the cache after swapping the contents of these two parts.

(4) Cache migration is a service function provided by the cache module for the object migration process. The purpose is to improve the availability of the cache, mainly when the object is migrated, the corresponding cache state is marked as the cache state, which can deny and delay access, and wait for the update.

3.4. Implementation of Object Bus Communication Module

The basic communication function is the foundation of the buffer module, which contains two parts: the sending and receiving of information.

(1) Sending of information:

```c
int sendDataLine(SOCKET sclient, struct dataLine dl, void* defaultTarget)
{
    Send the information in dl;
    If (send failed)
    {
        Return the failure value -1;
    }
    While (processing is not over)
    {
        Receive feedback information;
        Processing feedback information;
    }
}
```

(2) Receipt of information

```c
void* service_run(void *null)
```
4. Experimental data and summary

Test environment: The object bus is written in visual studio, so the same platform is used to design and write test cases. The host system used for testing is windows 10 x64.

Test content: The program creates an object bus, and then simulates the construction of some objects and information. Through some external operations, the following functions are used to test:

1) Recording module function: including the initialization of the object bus, the object bus entering the network and exiting the network, observing whether the corresponding parameters are correct, and whether the access and exit procedures can be executed correctly.

2) Cache module function: It includes functions such as adding cache, deleting cache, and searching cache of the object bus, such as handling situations such as preemption, and whether the handling of situations such as cache space exhaustion is correct.

3) Communication module function: including various tests of object bus communication, such as object positioning, information transfer of four ports, and object migration realization. During the process of object migration realization, cache migration will also be observed.

The function of the recording module of the object bus is operating normally, providing support for the behavior of the object bus, the initialization of the object bus, the operations of accessing and exiting the object bus network are also normal, can correctly handle the error address information, reflecting the safety of the object bus And robustness considerations. The positioning of the object, the information transmission of the four ports, and the realization of the migration of the object are all performed accurately.

5. Acknowledgments

This work was financially supported by Shenzhen Special Fund for Strategic Emerging Industry Development JCYJ20170816151055158 and Key Basic Research Projects of the Military Science and Technology Commission Basic Strengthening Plan 2019-JCJQ-ZD-097-00.

References

[1] V P Mochalov, N Y Bratchenko, S V Yakovlev. Analytical model of object request broker based on Corba standard[J]. Journal of Physics: Conference Series, 2018, 1015(2).

[2] YOSHINO Daishi, WATANOBEG Yutaka, YAGUCHI Yuichi, NAKAMURA Keita, NARUSE Keitaro. Application possibility of OpenRTM-aist-based integrated robot systems using CORBA interfaces and brokered Pub/Sub messaging interfaces[J]. The Proceedings of JSME annual Conference on Robotics and Mechatronics (Robomec), 2017, 2017(0).

[3] Dong W. Kim, Ho-Dong Lee, Clarence W. de Silva, Jong-Wook Park. Service-provider intelligent humanoid robot using TCP/IP and CORBA[J]. International Journal of Control, Automation and Systems, 2016, 14(2).

[4] Fang, Liu. Analysis on the Distributed Computer Software Bus Architecture[C]/ 2018.

[5] Zhao Q, Gu Z, Zeng H. Design Optimization for AUTOSAR models with Preemption Thresholds and Mixed-Criticality Scheduling[J]. Journal of Systems Architecture, 2016: S1383762116300984.

[6] Chun Xi Zhao. Design of Extended Event Service Model Based on CORBA[J]. Applied Mechanics and Materials, 2014, 3468.