International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 3 Mar 2022, pp: 1264-1266 www.ijaem.net ISSN: 2395-5252

Component model based on COM extension optimization

Mohammed Layth Talal [1], Bahaa Abdul qader Thabit [2]
College of Administration and Economic, University of Diyala, IRAQ [1]
Department of Computer Science, Basic Education College, University of Diyala, IRAQ [2]

Submitted: 20-03-2022 Revised: 27-03-2022 Accepted: 30-03-2022

ABSTRACT: component approach. The basic tools used to build programs are today the most important and main component in various development teams in both small and large projects. The basic toolkit is of particular importance for the latter, since the inertia in such projects is very high and architectural flaws and weaknesses of the basic tool can lead to serious problems in the future. Since the scope of application is quite wide, the basic toolkit must be flexible, high speed, require a minimum of resources of the final operating system, in addition, extensibility is required for the introduction of new links [1].

Keyword:COM,object-oriented, GUID,scheme.

INTRODUCTION

When developing applications, most software developers use component technology, which is a development of object-oriented design technology [2]. It combines the flexibility in component selection inherent in developing an application in-house with the code reliability and functional completeness proven by repeated use typical of commercial software products. Moreover, component technology allows you to quickly make changes to an existing application without violating its performance [1]. At the same time, new applications can work with new modules, and old ones with the old ones that remain in the system. The problem of "legacy" systems is removed (there is no need to completely replace them to change or expand functionality, which means

Component model Microsoft COM (Component Object Model). In COM, an object is characterized by its class. A class is an implementation of some set of interfaces. Multiple inheritance of interfaces is not supported, instead an object can have multiple interfaces at the same time. In COM, an interface can be defined by inheriting from another interface. All interfaces have a base interface, IUknown. To navigate from an interface of a base type to an inherited interface, or from one of an object's interfaces to another, the client must call the QueryInterface method defined in the IUknown [5] base interface.

COM uses a 16-byte GUID to identify classes and interfaces. An object in COM is an instance of a class. The client accesses the object using a pointer to one of its interfaces. The relationship between a class and the set of interfaces it supports is rather arbitrary. There is no predefined relationship between a class identifier and a particular set of interfaces, and different instances of a class may support different subsets of interfaces. The class identifier refers to a particular implementation, and the actual set of interfaces for a given implementation is only finally known at runtime.

Borrowed from C++, the integration model at the binary level using function tables has its advantages. The ability to replace object implementations, invisible to the user, and the high efficiency of method calls within the same address space, similar to calling a virtual function in C++, are positive aspects of this model.

A significant and, perhaps, the only drawback of COM is the overhead associated with creating class instances. Creating an instance of a class in COM involves searching for an implementation module by the object identifier GUID in the registry in which the class implementation is located [4], loading the module and allocating memory for the class instance. As you know, registry operations in a Windows system are relatively long, memory allocation is an expensive procedure, as is loading a module. The advantages of the component approach are negated when building on its basis, for example, various parsers of structured data formats, such as archives, mail messages, HTML pages, OLE documents, etc. Parsing such structured data entails constant multiple creation of object instances, where significant performance loss will occur. Loading a module is a necessary step when creating an object, and as for searching the registry and allocating memory, you can try to do without them.

Optimization and expansion of COM. The author has developed a component model that eliminates the listed disadvantages of the COM model. The main ideas in building the model were:

The ability to create objects in memory provided from the outside;

· the possibility of explicitly specifying the implementation module file when creating an object.

The point of this approach is the ability to provide memory to an instance of the class, allocated in a special way, optimal for work in each use case. For example, you need to quickly create an instance of an object of a certain class. Stack memory can be used for this, since memory allocation on the stack is very fast compared to the allocation of small blocks of memory by standard operating system tools. However, stack memory may not be sufficient in situations where you need to create many instances of different classes at the same time, for example, when parsing compound container objects. In this case, memory can be allocated in large blocks as needed, and space can be reserved from each block for each instance created. It will be much faster

The ability to access the class of an object by explicitly specifying a module reduces the time it takes to create an object, since there is no comparison between the object identifier and its implementation module.

In the developed model, as in COM, there are classes, classes have a list of supported interfaces. Each object interface has a reference count. All interfaces inherit from IUnknown. One module may implement one or more classes. Ensuring the data protection of a class instance from multi-threaded access is assigned to the component itself, which makes it possible to optimally implement protection against parallel access and not lose unnecessarily the speed of object execution.

The concepts of static and dynamic class interfaces were introduced into the component model. Static interfaces are designed to solve the problem of implementing functionality that does not require the creation of class instances. This saves time because instantiation takes extra time. Dynamic interfaces, on the other hand, belong to class instances and solve the same tasks as in COM.

The ability to explicitly specify parameters when creating classes was introduced, among which there must be a pointer to external memory for solving problems of creating on external memory. In addition, the method of obtaining information about a class from an arbitrary interface, including its unique identifier, has been standardized. Class and interface identifiers are the same as in COM, that is, it is a well-known GUID. This provides the possibility of identifying the object class, which is required when passing parameters of pointers to interfaces. For example, when you need to convert an arbitrary interface pointer to some specific type of C++ implementation class. Such tasks can also be solved by implementing a separate interface and then trying to request it, but this is not very convenient for programming, and the code is more cumbersome and confusing. After all, the concept of interfaces is designed to provide encapsulation of data and implementation, and in this case it will be redundant.

Each class has its own IEntryPoint_… entry point interface, which implements methods for creating a class instance with arbitrary parameters and methods for obtaining the required external memory size. The class creation method can return a predefined interface to make it easier to use when programming. You can also receive static interfaces through the entry point interface. An analogue of the entry point interface in COM is the class factory interface, which can also be obtained through it. Thus, when describing a class, its own entry point interface is described with its own creation method with parameters.

The scheme for creating a class instance is as follows: first, the module is loaded into memory and a pointer to the DllGetClassObject function is obtained, similarly to COM; then, through this function, a pointer to the interface of the entry point of the desired class or a pointer to the static interface is obtained, and already through the interface of the entry point, by calling the method, an instance of the object is created.

The author has developed and implemented a component model that makes it easier and more efficient to use all the advantages of the component approach in software systems of any complexity. The technology allows you to create instances of classes as quickly as possible, make descriptions of interfaces and their relationships with classes more understandable, and simplify the operation of interfaces. The technology is an extension of Microsoft's COM model.

REFERENCES

[1]. Dubenetsky V.A., Sovetov B.Ya., Tsekhanovsky V.V. Models of information technologies of organizational management. // Sat. elected. work on grants in the region. informatics and control systems. - Regional Center for Scientific and Technical. examinations at SPGETU. - 1996.

[2]. Dubenetsky V.A., Ilyin V.P., Lachinov E.S., Tsekhanovsky V.V. Object-oriented technology for computer-aided design of distributed information systems. // V intl. Conf.: "Regional Informatics-96". - St. Petersburg, 1996.

[3]. Dubenetsky V.A., Ilyin V.P., Tsekhanovsky V.V. Object-oriented technology for studying and researching the subject area of system engineering. // V intl. Conf.: “Regional Informatics-96”. - St. Petersburg, 1996.

[4]. Dubenetsky V.A., Sovetov B.Ya., Tsekhanovsky V.V. Technology of formalization of the structure of the concepts of the subject area and their functional relationships. // Mater. VI intl. Conf.: "Regional Informatics-98". - St. Petersburg, 1998.

[5]. Dubenetsky V.A., Tsekhanovsky V.V. Construction of a conceptual model of the subject area on the example of discrete manufacturing. // Mater. VII intern. Conf.: "Regional Informatics-2000". - St. Petersburg, 2000.

[6]. Dubenetsky V.A., Sovetov B.Ya., Tsekhanovsky V.V. Representation of subject areas in a computer environment. // VII Intern. Conf.: "Modern learning technologies". - SPb., 2001.