STUDIA UNIV. BABEŞ-BOLYAI, INFORMATICA, VOLUME XLIV, NUMBER 1, 1999

TOWARD A FORMAL MODEL FOR COM

IUGA MARIN

Abstract. In this paper is provided a formal notation for the COM's concepts. In this way is possible to describe formally the basic concepts of component programming as interfaces, component classes and component objects. Also, the COM's basic principles are expressed trough six axioms, three regarding IUnknown and three regarding the class factory object. All these notation and axiom are the start in developing a future formal component calculus.

1 A General Overview of COM

COM (abbr. of Component Object Model) is considered now to be the basic layer of the newest Microsoft's technology. Designed to avoid most of the object oriented programming problems (such as reviews of objects in monolithic applications, integration of legacy code, versioning and inter-object language neutral communication) COM is a binary standard which is leading the software development to OLE, considered in [Brockschmidt95] to be "... a unified environment of object-based services with the capability of both customizing those services and arbitrarily extending the architecture through custom services, with the overall purpose of enabling rich integration between components".

COM focuses on components which are reusable piece of code and data in binary form that can be used along with other components from possible other vendors. The concept of component enlarges the concept of object breaking the language barrier. A component could be developed both in procedural or object oriented languages with no constraints to its internal implementation. Since the internal implementation of the component is invisible to the external environment, it exposes a series of interfaces through it can communicate and collaborate with other components. An interface is practically a declaration of a set of methods signature without regarding to any implementation. An interface can be assimilated to a mean for accessing the functionality of a component. All components will be accessed only through its exported interfaces.

As previously seen COM is a model for component programming. In the following paragraphs these components will be called as COM objects. All COM objects exports a set of interfaces, and the interfaces will be browsed using the special interface IUnknown implemented by all COM objects. Similar COM objects are associated with a COM class

identified by a class identifier which is unique for the COM class. Every COM class should have a "class factory " object which creates instances of COM objects associated to that class and implements IClassFactory interface.

It is important to have a formal model for COM because it is possible to express unitary its concepts since the components design is still enough empirical. Also a formal model for COM may improve the performances of software systems based upon components. In this paper will be provided both formal notation for COM components and some axioms for this model.

2 Preliminary Notations

Many of these notations are an adaptation of Cardelli's object calculus, developed in [Cardelli], to component programming theory.

In order to develop a COM formal notation it is necessary to provide a formal notation for interfaces. An interface has its name prefixed with letter I (e.g. I*name*). The interface variables[1] will be denoted using first letters of the alphabet and IA:INTERFACE means IA is an interface. To specify an interface it is necessary to specify its methods as follows:

I*name* :

signature of *method1*

...

signature of *methodn*

A COM class will be referred through its COM class identifier, identifier which has its name prefixed with letter C (e.g. C*classname*). The COM class variables will be denoted using the last letters of the alphabet. The fact that an item CX is a COM class will be expressed using notation X:COMCLASS, and the literal COMCLASS is identified with the set of all COM classes.

If the COM class CX exports the interface IA then this is denoted:

$$CX \rightarrow IA.$$

COM objects can be instantiated from their classes and accessed via their exposed interfaces.

If the COM object X exports the interface IA then this is denoted:

$$X \rightarrow IA.$$

Since all COM object can be accessed only by their interfaces, a COM object may be referred as:

[1] In formal not programatic sense

IA(CX)

where CX is the COM object's COM class and IA is the interface used to access the component.

The set of all COM objects will be denoted with COMOBJECT and if X is a COM object then it can be written as X:COMOBJECT. As previously an object can be instantiate from a COM class using the class factory object for that class. For CX:COMCLASS its class factory will be designated using the operator:

ClassFactory:COMCLASS→COMOBJECT

where the ClassFactory(CX) means the CX's associated ClassFactory.

If the COM object X was instantiated from COM class CX, that will be noted:

X:CX.

The call of a method implemented by an object X:CX using an interface IA is denoted by:

IA(X).method(parameter list).

The practice of COM programming uses widely the HRESULT type as a return type for a method call. The value S_OK means that the call was successful, any other value means failure. Since the success of a method call can be determined by comparing the return value of the call with S_OK it is possible to define predicates SUCCESS and FAIL:

SUCCESS(IA(X).method(parameter list)) when the call was successful

FAIL(IA(X).method(parameter list)) when the call has failed

and consider the return value for all methods as void.

3 IUnknown axioms

All COM objects must implement the interface IUnknown and all interfaces exposed by a COM object must have the IUnknown's methods. The behavior of these methods is identical in any other interfaces. The IUnknown interface can be specified as in [MSDN6]:

Interface IUnknown:

QueryInterface(IA:INTERFACE, X:CX) where CX:COMCLASS

AddRef()

Release()

The last two methods are strongly related to implementation, ensuring the reference counting mechanism. These methods succeeds always so the SUCCESS predicate is always true.

The QueryInterface, by far one of the most important method of all interfaces, allows the browsing of interfaces exposed by a COM object. For a COM object, the call:

IUknown(X).QueryInterface(IA,Y)

will make, in case of success, the Y object variable to be the COM object X accessed via interface IA. If X does not expose interface IA the call will fail.

There can be three axioms related to IUnknown:

Axiom 1: IUnknown implementation

$$\frac{CX : COMCLASS}{CX \rightarrow IUnknown}$$

Axiom 2: Interface browsing

$$\frac{X : CX \quad CX : COMCLASS \quad CX \rightarrow IA}{SUCCES(IUnknown(X).QueryInterface(IA, Y)) \; where \; Y : CX}$$

Axiom 3: Independent interface browsing

$$\frac{X : CX \quad CX : COM \quad CX \rightarrow IA \quad CX \rightarrow IB}{SUCCES(IA(X).QueryInterface(IB, Y)) \; where \; Y : CX}$$

4 ClassFactory axioms

Normally, every COM class can be instantiated using a COM object called the class factory object for that class. Its role is to make instance of a class, eventually setting a default interface for every new instance. The main interface implemented by a class factory is IClassFactory which can be described as follows [MSDN6]:

Interface IClassFactory:

CreateInstance(outer: IUnknown(aggr:COMOBJECT),

interface:INTERFACE, object:COMOBJECT);

LockServer(BOOL flock);

Since LockServer is strong related to software implementation (it is a way to control server unloading), the relevant method from IClassFactory is CreateInstance. The CreateInstance's fundamental role is to create the *object* COM object accessed with interface *interface* and inside the aggregate *aggr* COM object. The *aggr* object may be a null object so the new created instance is not aggregatable.

First axiom concerned to ClassFactory objects is:

Axiom 4: ClassFactory universal definition

$$\frac{\forall \, CX : COMCLASS}{\exists \, Y : COMOBJECT \quad Y = ClassFactory(CX)}$$

This means that for every COM class CX it's class factory object always exists.

The second axiom can be formulated as:

Axiom 5: IClassFactory universal implementation

$$\frac{\forall CX : COMCLASS \qquad Y = ClassFactory(CX)}{Y \rightarrow IClassFactory}$$

which means that every class factory object must implement IClassFactory .
Finally, the last axiom is:

Axiom 6: Instance creation

$$\frac{\forall CX : COMCLASS \quad CX \rightarrow IA \quad Y = ClassFactory(CX)}{SUCCESS(IClassFctory(Y).CreateInstance(IUnknown(aggr), IA, object))}$$

where aggr can be a null object.

This axiom specifies that the class factory operator for class CX which exports IA interface should always create new instances of CX accessed with IA interface (depending eventually on nullity or non-nullity of aggr).

5 Conclusions

In this paper a simple model for formal notations and some axioms regarding COM was given. It is practically a way to describe COM in a formal way which avoids any empirical and informal expressions. Using the provided set of axioms it is possible to conceive COM in an unitary manner.

Maybe the main usefulness of this model is that it can be the start for a COM object calculus. This calculus could be used to describe aggregation or delegation inside the COM model, creating thus a formal method for component design.

REFERENCES

[Brockschmidt95] Kraig Brockschmidt, Inside OLE 2nd Edition, Microsoft Press (A Division of Microsoft Corporation), 1995

[Cardelli] Luca Cardelli & Martin Abadi, An Imperative Object Calculus, DEC Systems Research Center, date unknown

[MSDN6] Microsoft Developers Network, Visual Studio 6, Platform SDK: COM and ActiveX Object Services: COM, 1998

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. M. Kogalniceanu 1, România.