Plan

1. **Introduction**
2. **Généralité**
3. Historique de COM et DCOM
4. Définition

* COM
* DCOM

1. Objectifs de COM et DCOM
2. **Architecture des COM et DCOM**
3. Architecture du COM
4. Architecture tu DCOM
5. **Les composants**
6. Contrôle ActiveX
7. DLL ActiveX
8. EXE ActiveX
9. Document ActiveX
10. **Les interfaces**
11. IDL
12. Interface Unknown
13. Interface IclanFactory
14. Interface IDispatc
15. Interface IMoniker
16. **Avantage et inconvénients**
17. Avantage de COM/DCOM
18. Inconvénients de COM/DCOM
19. **Codage d’un objet**
20. COM
21. DCOM
22. **Conclusion**
23. **Introduction**

L’informatique actuelle doit être à l’écoute des utilisateurs et se doit de répondre le plus rapidement possible à leurs besoins.

En conséquence, les applications monolithiques du passé sont progressivement remplacées par des applications distribuées à bases de composants. Une application à base de composants se compose de briques logicielles réutilisables pouvant interagir entre elles.

Ces assemblages de composants logiciels réutilisables et prés à l’emploi permettent de décomposer les applications stratégiques en sous-ensembles de processus coopérant les uns avec les autres : réactivité des architectures informatiques, maîtrise des coûts, évolutivité dans le temps sont alors améliorées.

Pour mettre en place des applications à base de composants, une architecture logicielle est nécessaire: elle définit des spécifications fondamentales que doivent respecter les composants de l’architecture et fournit un ensemble de services.

Deux grands modèles d’architectures logicielles sont sur le devant de la scène : CORBA de l’OMG et COM/DCOM de Microsoft, entre lesquelles EJB de Sun se présente comme un outsider. Le présent document va décrire sommairement l’architecture logicielle de COM/DCOM.

1. **Généralité**

**Historique**

**DDE**

Tout d’abord cela à commencer par le presse-papiers (1987), qui permettait aux utilisateurs de copier des portions de données d’une application à l’autre. Cela fonctionnait bien pour des documents simples, mais lorsque ceux-ci étaient plus complexes, le presse-papiers atteignait ses limites. Plus tard, Microsoft développa la technologie DDE, Dynamic Data Exchange ou Echange dynamique de donnée, une « nouvelle génération » de presse-papiers, qui permet d’insérer des données à partir d'un document d’une autre application. Ces données restaient liées à l'application d'origine. Cette technologie ne fonctionnait pour les applications de Microsoft. A la différence de OLE, où on pourra lancer l'application d'origine en cliquant dans l'objet : feuille Excel dans l’application Word.

**OLE1**

OLE « Object Linking and Embedding », signifie « Intégration d'objets et Lien sur des objets » développé en 1991, le but était de permettre l’insertion des données provenant de plusieurs applications sur un seul document. Et cela soit par intégration complète, soit par référence (ce que l'on appelle une liaison). On peut voir cela dans la plupart des applications où apparaît dans le menu « Coller | Collage spécial ». Les objets intégrés ou liés sont capables de s'afficher dans l'application qui les contient.

**OLE2**

OLE1 était fondé sur des messages Windows mais, cela causait de grosses difficultés d’intégration dans la mesure où toutes les applications n’utilisaient pas les mêmes technologies. C’est donc pendant le développement d’OLE 2 que Microsoft inclus la technologie COM dans son modèle de composant.

**COM**

COM « Component Object Model » ou Modèle d'objet composant est né avec OLE2. C'est une spécification de Microsoft qui décrit comment créer les objets réutilisables. Il a aussi pour but de définir un standard de communication (en fait des méthodes) afin d’accéder aux objets OLE localement. Nous détaillerons cette partie plus loin.

**DCOM**

Microsoft a ensuite fait évolué la technologie COM afin de permettre la répartition des composants sur un réseau. L'aspect distribué a donné le préfixe D pour Distributed COM, qui spécifie toujours les composants, mais cette fois distants et interopérables avec aux systèmes d'exploitation hétérogènes.

**Définition de COM/DCOM**

**COM**

-COM « Component Object Model » ou Modèle d'objet composant est une spécification de Microsoft qui décrit comment créer les objets réutilisables. Pour bien comprendre la notion de COM, il est intéressant de se pencher sur les interfaces.

-Component Object Model (COM) est une norme [d’interface binaire](https://en.wikipedia.org/wiki/Application_Binary_Interface) pour [les composants logiciels](https://en.wikipedia.org/wiki/Component-based_software_engineering) introduite par [Microsoft](https://en.wikipedia.org/wiki/Microsoft) en 1993. Il est utilisé pour permettre la création [d’objets de](https://en.wikipedia.org/wiki/Object_(computer_science)) [communication inter-processus](https://en.wikipedia.org/wiki/Inter-process_communication) dans une large gamme de [langages de programmation](https://en.wikipedia.org/wiki/Programming_languages).

**DCOM**

Le modèle DCOM permet la communication entre des objets s'exécutant sur la même machine. Le modèle de composants distribués DCOM, étend COM afin de supporter les communications avec des objets situés sur des machines distantes, et sur des platesformes différentes. Initialement développé pour Windows, DCOM est maintenant disponible sur des plates-formes UNIX, dont Solaris, Linux et HP/UX. DCOM = COM + RPC (Remote Protocol Call) DCOM supporte les objets distants grâce au protocole ORPC (Object Remote Procedure Call), qui lui, est basé sur RPC (Remote Protocol Call, de DCE : Distributed Computing Environment). Le fonctionnement est détaillé dans la partie architecture globale.

**Les objectifs de COM**

Les Objectifs de COM : Voici les principaux objectifs de COM (Cf. Spécifications de COM) :

• Définir un protocole pour créer des composants.

• Réutilisation des composants

• Microsoft affirme que l’on peut accéder et développer les composants indépendamment du langage de programmation pourvu que le langage permette d’accéder aux pointeurs de fonctions. Il est possible d'utiliser par exemple : C, C++, Visual Basic Etc

**Les Objectifs de DCOM :**

• Les composants DCOM peuvent fonctionner dans des processus différents et sur des machines différentes.

• Possibilité de faire évoluer à tout moment les fonctionnalités d’une application (client, serveur ou composant COM). Cela se fait par QueryInterface que tous les composants COM supportent et qui permet de proposer de nouvelles fonctions à de nouveaux clients sans altérer la compatibilité avec les anciens. Les évolutions se font par de nouvelles Interfaces et un composant COM supporte toujours les vieilles Interfaces.

• Interactions simples et rapides. Les appels in-process se font par pointeurs.

• Réutilisation d’Interfaces.

• Transparence Local/Distant. Le standard binaire autorise le détournement des appels aux Interfaces pour les rediriger vers un objet distant de façon transparente pour l’appelant.

• Microsoft affirme que les composants DCOM sont indépendants du langage de programmation utilisé. A condition, que le langage ait accès aux pointeurs.

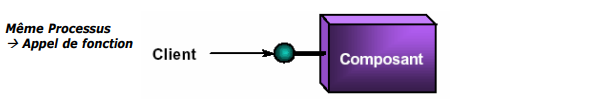
**Différence entre COM et DCOM :**

| Base de | COM | DCOM |
| --- | --- | --- |
| Abréviation | Modèle d'objet composant. | Modèle d'objet de composant distribué. |
| Exécution sur | Environnement côté client. | Environnement serveur. |
| Installation | Requis sur la machine où il est utilisé. | Requis sur le même réseau. |
| Type de modèle | COM est une norme d'interface. | DCOM un modèle conçu pour les applications distribuées. |
| Réutilisabilité des objets | Il permet. | Cela ne permet pas. |
| Autoriser distribué | Non, une telle capacité d'objets distribués. | Oui, il est capable d'objets distribués. |
| Utilisation de la mémoire | Il ne fournit pas une meilleure utilisation. | Il dispose d'un collecteur d'objets distribué qui améliore l'utilisation de la mémoire. |

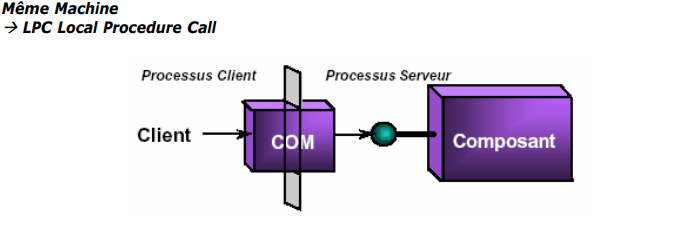
1. **Architecture des COM et DCOM**

En COM/DCOM on distingue essentiellement trois types de composants (On va développer plus les composants dans chapitre suivante  les :

1. Serveur « In-Process » : il s’exécute dans le processus de l’appelant. Tous les composants de ce type sont des DLL (ou des contrôles ActiveX).

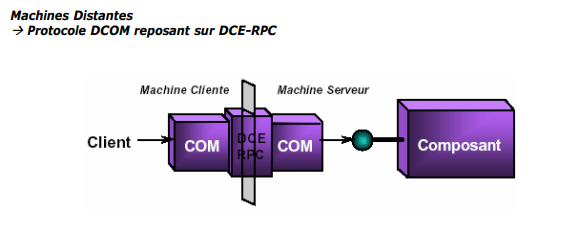


2. Serveur « Local » (ou « Out-of-process) : il s’exécute dans un processus séparé et sur la machine de l’appelant. Un composant de ce type peut être un service ou un exécutable. L'appel atteint d'abord l’objet proxy (voir récapitulatif des architectures) fourni par le modèle COM, qui génère l'appel de procédure à distance vers l'autre processus ou l'autre machine.

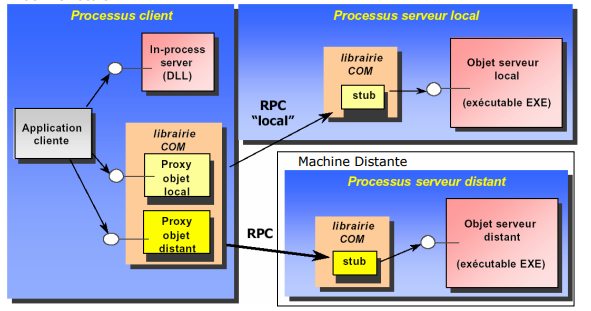


Serveur « Remote » : s’exécute dans un processus séparé, sur une machine distance. Un composant de ce type peut être un service ou un exécutable. Même Processus Æ Appel de fonction Même Machine Æ LPC Local Procedure Call Machines Distantes Æ Protocole DCOM reposant sur

DCE-RPC



**Recaptulatif de l’architecture**



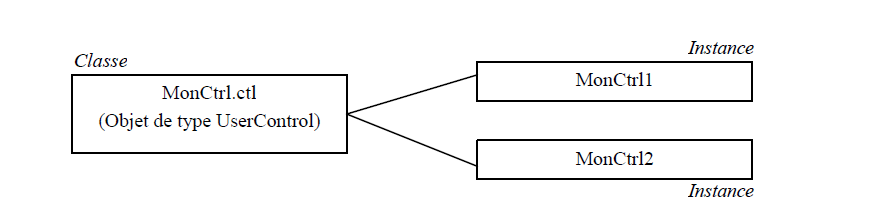
**Les composants**

1. **Contrôle ActiveX**

Les contrôles ActiveX (d’extension .ocx pour OLE Control Extension) sont des éléments logiciels indépendants et réutilisables qui incluent des éléments visuels et du code. La boîte à outils de Visual Basic renferme l'ensemble des contrôles intégrés; elle facilite et accélère ainsi considérablement le développement des applications. Les contrôles doivent être placés dans un type de conteneur quelconque, comme une feuille ou une application. Ils communiquent entre eux via un ensemble de normes et d’interfaces communs.

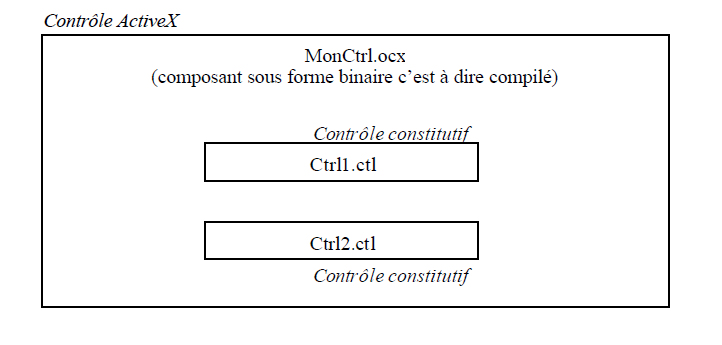
**Exemple** :

Tout contrôle créer dans Visual Basic est reconnu comme une classe de contrôle, laquelle lui sert de modèle. Lorsqu’on place un contrôle sur une feuille, une instance de ce contrôle est créée :



Un projet de contrôles ActiveX Visual Basic contient un ou plusieurs fichiers .ctl, définissant chacun une classe de contrôle distincte. Lorsqu’on compile un projet de contrôles, un fichier .ocx est créé pour le composant de type contrôle.

Par ailleurs, un fichier .ocx unique peut compter plusieurs contrôles, comme illustré ci-dessous.



1. **DLL ActiveX**

Une DLL ActiveX (d’extension .dll) est un composant de code compilé qui s ‘exécute dans le même processus que l’application cliente. En d’autres termes, une DLL ActiveX expose à la fois des fonctionnalités à d'autres applications et s'exécute dans le même espace mémoire que l’application cliente.

1. **EXE ActiveX**

Un EXE ActiveX (d’extension .exe) est une application fournissant des classes d’objets. Elle s’exécute out-of-process.

En d’autres termes, un EXE ActiveX expose à la fois des fonctionnalités à d'autres applications et s'exécute comme

une application autonome.

1. **Document ActiveX**

Un document ActiveX (d’extension .vbd) est un type spécifique d'objet ActiveX qui peut être placé et activé dans un conteneur de document ActiveX, tel que Microsoft Internet Explorer. Les documents ActiveX constituent une extension de la technologie de documents composés (liaison et incorporation d'objets encore appelé OLE).

Le principal aspect fonctionnel des documents ActiveX est leur effet sur l'interface utilisateur. Un document ActiveX remplit la zone d'affichage d'un conteneur et donne ainsi à ce dernier la possibilité de se comporter comme l'application serveur à travers une interface utilisateur semblable.

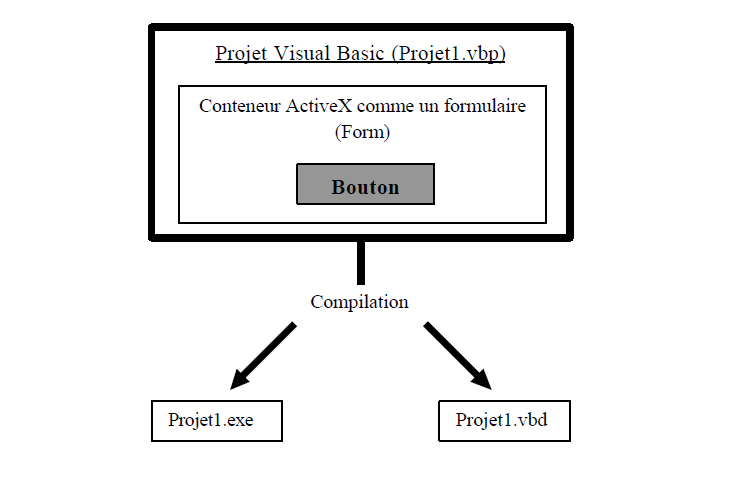
**Exemple** : lorsqu’on exécute Microsoft Word, on lance en fait l'application Word comme conteneur de son objet document. Lorsqu’un objet document Word est ouvert à partir d'Internet Explorer, Word est lancé comme un serveur fournissant le document à un autre conteneur, Internet Explorer ici.

Il existe donc deux types de documents ActiveX :

EXE Document ActiveX, pour lequel le serveur est un .exe

DLL Document ActiveX, pour lequel le serveur est une .dll

L'illustration suivante montre les fichiers créés par la compilation d'un projet EXE Document ActiveX.



La compilation d'un projet DLL ActiveX engendrerait la création du fichier Projet1.dll.

1. **Les Interfaces**

Une interface permet d’utiliser un composant, c’est une collection de fonctions (services) qui définissent le comportement des objets. Une Interface n’est ni une classe, ni un composant. C’est une liste qui contient les signatures des méthodes et non pas leur implémentation.

Les objets ne sont vus que par leur interface. Les objets clients invoquent les interfaces des objets serveur (qui fournissent des services). Et pour accéder à un objet, il faut avoir accès à une interface associée à cet objet, autre remarque : un objet peut avoir plusieurs interfaces.

**IDL**

IDL, acronyme d'Interactive Data Langage, est un [langage de programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation) [propriétaire](https://fr.wikipedia.org/wiki/Logiciel_propriétaire) apparu à la fin des années 1970, et qui est rapidement monté en puissance dans les domaines de la  télédétection et de l'[astronomie](https://fr.wikipedia.org/wiki/Astronomie). IDL est un langage vectoriel de traitement de données et de visualisation très répandu dans l'industrie et dans la recherche.

**Avantages :**

Une syntaxe claire

* un apprentissage très rapide en venant de [Pascal](https://fr.wikipedia.org/wiki/Pascal_(langage)), [Fortran](https://fr.wikipedia.org/wiki/Fortran), [Caml](https://fr.wikipedia.org/wiki/Caml) ou [C](https://fr.wikipedia.org/wiki/C_(langage)) ;
* Une rapidité de programmation ;
* La main sur les données (la commande « help » donnant accès à toutes les variables connues).

**Inconvénients :**

Un langage propriétaire dont les licences sont coûteuses

* Une évolution un peu chaotique des Widgets ;
* Un délicat passage à une approche objet *artificielle* (non demandée par les utilisateurs).

**Interface Unknown**

L’interface IUnknown est l'interface fondamentale dans le Component Object Model (COM). La spécification COM stipule que les objets COM doivent implémenter cette interface. De plus, toutes les autres interfaces COM doivent être dérivées de IUnknown. IUnknown expose deux fonctionnalités essentielles de tous les objets COM : la gestion de la durée de vie des objets via le comptage des références et l'accès aux fonctionnalités des objets via d'autres interfaces.

Une interface IUnknown (ou dérivée de IUnknown) consiste en un pointeur vers une table de méthodes virtuelles qui contient une liste de pointeurs vers les fonctions qui implémentent les fonctions déclarées dans l'interface, dans l'ordre dans lequel elles sont déclarées dans l'interface. La surcharge d'appel d'invocation in-process est donc identique aux appels de méthode virtuelle en C++.

**Interface IClassFactory**

Tous les serveurs COM/DCOM suivent le modèle de conception d'usine. La classe IClassFactory est la norme utilisée dans COM/DCOM pour réaliser cette conception. L'interface IClassFactory contient deux méthodes destinées à traiter une classe entière d'objets, et est donc implémentée sur l'objet de classe pour une classe spécifique d'objets (identifiée par un CLSID).

**Interfaces IDispatch**

IDispatch est l'interface qui expose le protocole OLE Automation. En étendant IUnknown, c'est l'une des interfaces standard qui peuvent être exposées par les objets COM. COM distingue trois types d'interface : les interfaces personnalisées IUnknown basées sur VTABLE, les interfaces IDispatch prenant en charge l'introspection et les interfaces doubles prenant en charge les deux types.

L'interface Automation (IDispatch) permet à une application cliente de savoir quelles propriétés et méthodes sont prises en charge par un objet au moment de l'exécution, c'est-à-dire implémente le concept de RTTI. Il fournit également les informations nécessaires pour appeler ces propriétés et méthodes. Les applications clientes n'ont pas besoin de connaître les membres de l'objet lorsqu'elles sont compilées. Cela permet aux objets COM et ActiveX d'être appelés par des plates-formes de programmes de script telles que le serveur ASP et JavaScript sur Internet Explorer, où les conventions d'appel n'étaient pas connues au moment de la construction d'IIS ou d'IE. En revanche, une simple bibliothèque d'objets est compilée et liée dans un programme, par ex. un appel DLL doit connaître un nom de fonction et des paramètres au moment de la compilation.

**Interface Imoniker**

L'interface IMoniker est utilisée pour nommer les objets COM. Un objet qui implémente IMoniker est appelé un « moniker », qui est un nom qui identifie de manière unique un objet COM. Vous pouvez considérer un surnom comme une généralisation d'un nom de chemin ; de la même manière qu'un nom de chemin identifie un fichier dans le système de fichiers, un surnom identifie un objet COM. Par exemple, supposons que vous ayez un objet représentant une plage de cellules dans une feuille de calcul qui est elle-même incorporée dans un document texte. Les monikers sont plus largement applicables que les noms de chemin ; alors que les noms de chemin ne peuvent identifier que des fichiers, les surnoms peuvent identifier n'importe quel type d'objet, qu'il s'agisse d'un fichier, d'une incorporation, d'une sélection dans un document ou autre chose.

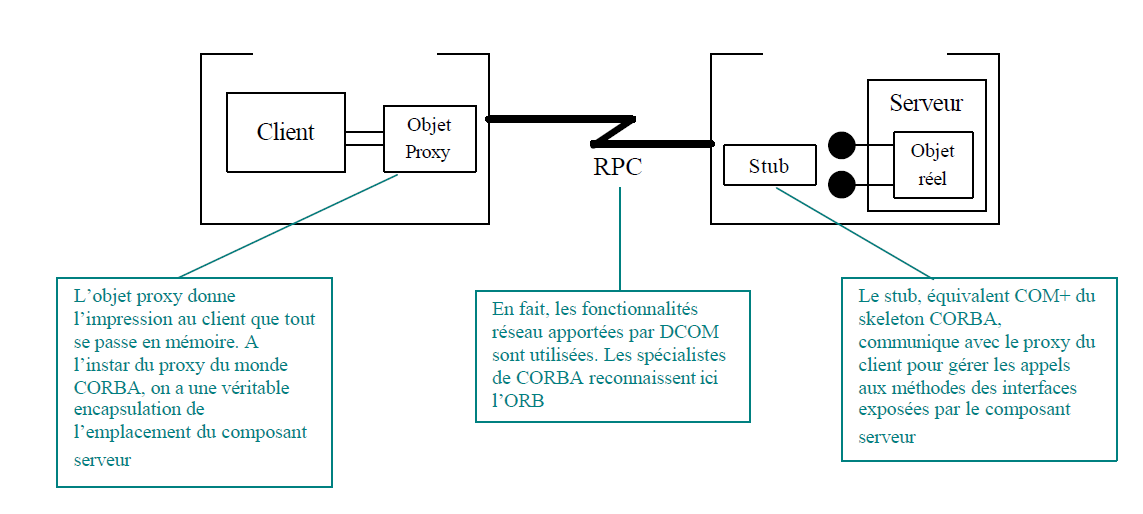
**LA COMMUNICATION ENTRE COMPOSANTS**

**Proxy et stub**

Comme mentionné précédemment, COM+ fournit des services permettant à des composants de dialoguer entre eux, quel que soit leurs emplacements sur un réseau. Pour que les composants puissent dialoguer, COM+ utilise deux objets spécifiques appelés proxy et stub. En pratique, un compilateur compatible COM+ insère du code proxy à l'intérieur d'une application cliente afin de gérer les appels du serveur et les valeurs renvoyées. Du coté composant serveur compatible COM+, du code stub est inséré pour gérer les appels à partir du code proxy du client ainsi que toutes les valeurs renvoyées aux clients.

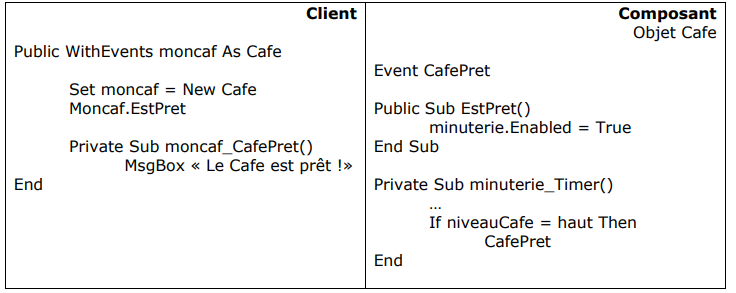
Ce traitement des paramètres et des valeurs renvoyées constitue ce que l'on appelle le **marshalling**. Le marshalling peut donc se définir comme le conditionnement de l'ensemble des paramètres et valeurs renvoyées, et leur va-et-vient entre les processus.

L'illustration ci-après représente la circulation des informations entre un client et un serveur par l'intermédiaire de code proxy et stub.



1. **Codage (Un exemple de développement)**

**A faire bien**

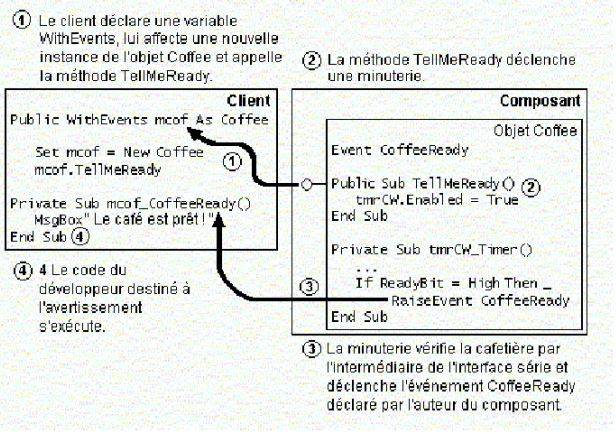


1. Le client déclare une variable WithEvents, lui affecte une nouvelle instance de l’objet Cafe et appelle la méthode EstPret

2. La méthode EstPret déclenche une minuterie

3. La minuterie vérifie la cafetière par l’intermédiaire de l’interface série et déclenche l’événement CafePret déclaré par l’auteur du composant

4. Le code du développeur destiné à l’avertissement s’exécute



1. **VII – CONCLUSION**

La complexité toujours croissante des applications, l’intégration devenue de plus en plus difficile ont propulsé les applications à base de composants sur le devant de la scène. Leurs aptitudes à la réutilisation, à la modularité en feront peut - être les fondements même des entreprises dans le futur.

De plus, la difficulté de conception d’applications stratégiques totalement réparties sur un réseau est considérablement réduite grâce aux architectures logicielles objet. En conséquence, COM et DCOM constituent les bases de l’informatique de demain.