Qu'est-ce que le "Middleware"

JM Alliot

13 mars 2003

1 But et origine

La définition du Middleware généralement admise est la suivante :

> *Un Middleware est un logiciel de communication qui permet à plusieurs processus s'exécutant sur une ou plusieurs machines d'interagir à travers un réseau.*

La définition est donc assez vague. La notion d'utilisation de machines en réseau remonte au projet Whirlwind, à la fin des années 40. Whirlwind fut un calculateur développé par l'équipe de Jay Forrester (MIT) qui fut utilisé pour le dispositif de défense SAGE (Semi Automatic Ground Environment). SAGE avait pour but de surveiller de façon automatique l'espace aérien américain contre les intrusions des bombardiers soviétiques. SAGE fut une mine d'innovation technologique : stylos lumineux, temps réel et surtout modems furent développés dans le cadre de ce projet. En effet, les 56 calculateurs répartis sur le territoire américain devaient interagir en temps réel. La première application civile de cette technologie fut le système SABRE de réservation développé pour les compagnies aériennes par IBM en 1964.

La deuxième grande étape fut l'arrivée des réseaux rapides locaux (LAN). Le développement d'Ethernet sous l'impulsion de Xerox dans les années 70 permit de diffuser plus largement les systèmes de machines en réseaux locaux. De la même façon, le développement d'ARPANET (qui allait devenir INTERNET) dans les mêmes années posa de manière plus aigü le problème des communications inter-machines et de l'interopérabilité.

Une autre grande étape fut l'apparition formelle, au début des années 80, du modèle client-serveur. Ce modèle, qui consiste à avoir une application (le serveur) qui fournit des services à des clients, a été largement popularisé, par exemple, dans le système graphique X-Windows développé au MIT. Bien d'autres applications répondent aujourd'hui à ce modèle client serveur ; en particulier, les systèmes de montage de fichiers NFS, les serveurs de fichiers FTP, les serveurs WEB (serveurs http), etc.

On distingue aujourd'hui quatre grandes classes de Middleware : l'exécution de transaction (Transactions Processing) qui est une classe de logiciel plutôt orienté "base de données" dont nous ne parlerons pas ici, les RPC (Remote Procedure Calls) qui distribue l'exécution de routines sur un réseau (voir plus loin), les MOM (Message oriented Middleware) qui permettent l'échange de données entre applications (voir également plus loin), et les ORB (Object Request Broker) qui permettent la distribution d'objets sur un réseau de machines (voir la description de CORBA).

2 Le modèle OSI

Le modèle OSI (Open System Interconnection) est une norme ISO (International Organization for Standardization) qui a pour but de séparer les différents niveaux (du niveau physique brut au niveau applicatif) liés à l'interconnexion des système. La norme OSI réalise ainsi un découpage en 7 couches :

couche 1 (physique) : appareillage électrique, cartes

couche 2 (liaison) : contrôle de la couche physique, gestion des erreurs

couche 3 (réseau) : routage des données, recherche du meilleur chemin

couche 4 (transport) : gestion de l'envoi de données

couche 5 (session) : maintien de synchronisation, coordination des communications

couche 6 (présentation) : conversion des données émises/reçues

couche 7 (application) : applications utilisant les propriétés d'interconnexion

Notons cependant que le plus grand réseau du monde (l'Internet) ignore royalement le modèle OSI ; le protocole TCP/IP, par exemple, est à cheval sur les couches 3 (réseau/routage assurée par IP (Internet Protocol)) et la couche 4 (transport, assurée par TCP (Transfer Control Protocol)). Notons également que nous aurons bien du mal à trouver la couche 5 (session) dont les fonctionnalités sont souvent assurés par les applicatifs eux-mêmes (couche 7) : par exemple, le protocole de transfert de fichier FTP s'appuie sur TCP/IP, mais est à cheval sur les couches 5, 6 et 7. Il existe trés peu d'exemple de protocoles normalisés de la couche 5 sur Internet[1].

La norme OSI est donc un modèle commode auquel nous nous réfererons, mais n'est pas un standard de fait. Ainsi, lorsque nous citons une couche OSI pour les différents protocoles présentés ci-dessous, il ne s'agit en aucun cas d'un protocole standardisé pour cette couche, mais plutôt d'un moyen de fixer les idées par rapport à l'échelle OSI.

3 Le transport des données (couche 4 OSI)

Lorsque des machines discutent entre elles, il leur faut choisir un mode de communication (qui serait situé "environ" au niveau de la couche transport de la norme OSI). Nous allons presenter rapidement les deux modes employés sur le réseau Internet : le mode TCP/IP (Transfer Control Protocol over Internet Protocol) et le mode UDP (User Datagram Protocol).

Pour faire simple, le mode TCP est un mode point à point. Une machine parle à une seule autre machine, en privé. La communication est bidirectionnelle.

Le mode UDP ressemble plutôt à un ensemble de personnes réunis dans une pièce qui hurlent au lieu de parler. Tout le monde peut entendre tout le monde. En revanche, il faut savoir trier et organiser l'information, et s'assurer que le message est bien arrivé.

Le format des informations qui transitent que ce soit en mode TCP ou en mode UDP dépend totalement des couches qui se trouvent au dessus dans le modèle OSI. En particulier, c'est la couche présentation, qui définira l'encodage des données en utilisant une des méthodes décrites dans la section suivante.

[1] Les raisons de ce phénomène sont historiques. La norme OSI a été définie à la fin des années 70, et s'est voulu une remise au net des concepts d'interconnexion, alors que l'Internet et ses protocoles existaient depuis le début des années 70.

4 La présentation des données (couche 6 OSI)

Tant qu'un processus s'exécute sur une seule et même machine, la façon dont les données sont représentées intéresse fort peu l'utilisateur. Les choses viennent à changer dès que l'on essaie d'échanger des données entre deux machines.

L'exemple le plus classique et sans doute le plus simple à comprendre est celui de la représentation des entiers. Il existe encore aujourd'hui deux grandes familles de processeurs : ceux qui représentent les entiers en format dit "little endian" et ceux qui les représentent en format "big endian". Pour comprendre la différence, il faut s'intéresser (un peu) à l'architecture physique des machines : l'unité de base en informatique est le bit (0 ou 1). Aujourd'hui, l'unité de base de la mémoire d'un ordinateur est l'octet, qui est un bloc de 8 bits. On ne peut représenter avec 8 bits que les valeurs de 0 à 255 (2^8). Pour représenter des valeurs plus importantes, il faut combiner plusieurs octets entre eux. Avec deux octets, on arrive ainsi à 65536 (256^2), etc. Ces deux octets sont placés physiquement dans la mémoire de la machine l'un derrière l'autre, mais suivant le type du processeur, ils ne seront pas placés dans le même ordre. Ainsi, par exemple, sur les processeurs Intel (PC), le nombre 1026 ($4 \times 256 + 2$) sera représenté par l'octet 00000100 (4) à l'adresse 1 de la mémoire et l'octet 00000010 (2) à l'adresse 0 (technique de codage "little endian" : l'octet dit de poids faible, c'est à dire celui qui a le moins d'importance, est placé à l'adresse la plus faible). L'ordre sera inversé sur un processeur SPARC (Sun) où l'on aura 00000010 suivi de 00000100[2].

Supposons maintenant que notre PC envoie le nombre 1026 "brut de fonderie" à notre machine Sun à travers un réseau quelconque sans se préoccuper de la représentation des données. 1026 qui valait bien $4 \times 256 + 2$ sur notre machine PC va maintenant être interprété comme $2 \times 256 + 4 = 514$ sur notre Sun, puisqu'il est "dans le mauvais sens".

Il existe plusieurs façons de résoudre ce problème, allant du simplissime (mais limité) à la (petite) usine à gaz plus puissante.

4.1 L'encodage ASCII et XML

Les caractères imprimables sont tous représentés sur un octet par un code défini par la norme ASCII (ainsi le A majuscule est représenté par le nombre 65). La norme ASCII étant aujourd'hui extrêmement répandue (il existe cependant encore quelques machines utilisant la vieille norme EBCDIC), l'idée consistant à prendre tout simplement la représentation ASCII d'un nombre, à la transmettre, puis à la récupérer et la recoder à l'autre bout, est venue tout naturellement à l'esprit, d'autant que les procédures de codage/décodage sont disponibles sur toutes les machines, puisqu'elles sont aussi utilisées pour saisir les chiffres rentrés par les utilisateurs, ou pour imprimer les résultats d'un programme. Cette procédure simple peut être suffisante dans nombre de cas, mais présente aussi certains inconvénients :

– Le typage est totalement perdu ; l'application qui décode à l'autre bout doit savoir exactement quelles données vont lui être envoyées (entiers, entiers longs, entiers courts, flottant simple, chaîne de caractère, etc), sous peine de se tromper complètement dans le décodage. On peut certes rajouter un protocole indiquant le type de données transmises (on envoie d'abord un 1, par exemple, lorsque l'on va envoyer un entier codé par un caractère, un 2 pour en entier codés par deux

[2]Nous évitons dans cette exemple tous les problèmes annexes, comme celui de la taille des entiers : sont-ils codés avec 2, 3 ou 4 ou plus d'octets, etc. Nous y revendrons, mais nous avons supposé ici qu'un entier était codé sur 2 octets.

caractères, un 3 pour trois caractères, etc), mais le protocole devient alors plus complexe. On peut aussi rajouter des séparateurs, mais la procédure de décodage est plus difficile, et l'on commence à perdre les avantages de la simplicité du protocole (c'est la solution adoptée avec le balisage XML).
- Pour certains types de données (en particulier les flottants), le codage/décodage ASCII introduit une perte de précision, minime certes, mais suffisante pour entraîner des erreurs de calculs, surtout dans les problèmes mal conditionnés.

Il faut cependant noter que l'encodage ASCII est aujourd'hui très utilisé à travers le protocole XML. XML est un avatar direct de SGML (Standard Generalized Markup Language). SGML est un standard développé au début des années 80 (norme ISO-8879). Il dérive d'un produit IBM (GML) et du célébrissime et encore bien vivant TEX de Donald Knuth. SGML définit ce que l'on appelle les langages à balises (Markup Language). L'idée qui se trouve derrière les langages à balisage (c'est ainsi que l'on traduit généralement Markup Language) est une forme de réaction contre le WYSIWYG (What You See Is What You Get). Plutôt que de représenter la forme d'un document directement sur l'écran, on le décrit par des balises, qui vont indiquer les débuts de chapitre, de paragraphe, les énumérations, etc, mais on ne se préoccupe jamais de la façon dont le document lui-même sera représenté. L'application la plus célèbre de SGML est HTML (Hyper Text Markup Language) qui est le langage standard d'écriture des documents pour le Web.

Le principe d'utilisation de XML (ou SGML) est simple. On commence par écrire un DTD (Definition Type Document) qui décrit la syntaxe admissible pour les futurs documents que l'on écrira. Il s'agit d'une sorte de grammaire que devra respecter le futur document. Une fois cette grammaire écrite, on peut faire transiter les informations dans un format qui la respecte. De façon assez intéressante, le DTD est lui-même un document de type XML. C'est ensuite l'application qui à la charge de relire le document et de l'interpréter. C'est ce qui explique par exemple les différences de visualisation entre les différents navigateurs Internet. La norme HTML (qui est en fait un DTD) décrit simplement la grammaire à respecter, mais pas la façon dont on doit interpréter cette grammaire. On peut à l'aide de XML décrire à peu près tous les types de données possibles (avec les restrictions liées aux données codées en ASCII concernant la perte de précision, comme indiquée dans le XML Schema Part 2), mais on est loin de l'efficacité d'un ASN.1.

4.2 Le mécanisme XDR

XDR (eXternal Data Representation) est un standard de codage développé par Sun au début des années 80. Il est formellement défini par la RFC[3] 1832. Il correspond à peu près exactement à la couche 6 du modèle OSI.

XDR se situe à un niveau d'abstraction plus élevée que le simple codage ASCII. Il s'agit en fait d'un langage de description de données. XDR s'implante sur chaque machine sous la forme d'une librairie (un ensemble de routines) qui assure le codage et le décodage des données. XDR n'entend pas permettre la représentation de tous les types de données ; certains choix ont été faits : on suppose entre autre que les octets ont des représentations uniques, on ne sait pas encoder les champs de bits ou les nombres BCD (Binary Coded Decimals), on code toutes les données en multiples de 4 octets, etc. Cependant, XDR permet d'encoder la plupart des structures de données de langages procéduraux traditionnels (C, ADA, Pascal, etc). Pour utiliser la librairie XDR,

[3]Request For Comments

il suffit que le programmeur décrive dans un fichier ASCII ses structures de données en utilisant des déclarations proches de celles du C ou du Pascal.

XDR a connu un très important développement. Il est utilisé dans de nombreuses applications, par exemple pour le codage des données des RPC (Remote Procedure Calls dont nous reparlerons), et donc dans le protocole NFS (Network File System).

Notons que le typage des données dans XDR est implicite : le passage sur le réseau fait disparaître le type de la donnée et l'application située à l'autre bout doit donc savoir quel type de données elle récupère.

4.3 ASN.1

La norme ASN.1 est apparue en 1984 sous forme d'une norme CCITT (norme X.409) avant de devenir une norme ISO quelque temps plus tard. ASN.1 implante la couche 6 du modèle OSI mais le langage de description de données serait plutôt de la couche 7.

ASN.1 est plus "lourd" mais aussi plus général que XDR, même si le principe de base est le même : il s'agit la aussi d'un langage de description de données qui permet de représenter les structures de données des langages de programmation. L'utilisateur décrit en ASN.1 ses structures de données. Son fichier de description est ensuite compilé par un compilateur ASN.1 qui génère les structures de données adaptées pour le langage de programmation de notre utilisateur, et permet la liaison du programme avec la librairie d'encodage ASN.1. Contrairement à XDR, le typage ASN.1 est explicite : les données sont encodées avec leurs types respectifs : en codage BER par exemple (il y a plusieurs types d'encodages de données en ASN.1), chaque donnée est représentée par un triplet (Tag,Longueur,Valeur), où le Tag identifie le type de données.

ASN.1 est très largement utilisé, depuis les transferts de mail (norme X.400), les stockages d'information (protocole DAP, X.500), le multimédia, la cryptographie, l'échange de données, ou même le futur Aeronautical Telecommunication Network (ATN).

ASN.1 reste lourd à mettre en oeuvre.

5 Procédures et applications distribuées (couche 5 OSI ?)

Nous ne nous sommes intéressés jusqu'ici qu'au simple encodage et transport des données. Ce modèle simple suppose que la machine qui va récupérer les données sait exactement ce qu'elle va devoir en faire. Il est impossible de spécifier à distance l'exécution d'une procédure particulière, à moins que l'on ne décide d'encoder dans le protocole de données des informations sur les opérations à effectuer[4].

Les développeurs ont très tôt souhaités réaliser une distribution du calcul et des procédures de façon plus transparente, ainsi qu'une synchronisation des processus. Là encore, il existe plusieurs modèles allant du plus simple au plus compliqué.

5.1 "Sun" RPC

Sun RPC (Sun Remote Procedure Calls) est un protocole d'abord développé par Xerox, puis employé par Sun, qui s'est aujourd'hui généralisé dans le monde UNIX. Il est défini par la RFC 1831.

[4]La première donnée envoyée peut par exemple indiquer un numéro de procédure à exécuter, etc. Cet exemple est là pour bien montrer une fois de plus qu'il est possible de faire à peu près tout ce que l'on souhaite avec des méthodes simples, mais au prix d'une complexité accrue au niveau des protocoles.

Le principe général est le suivant. Sur une machine, un programme “serveur” se lance. Il indique à un autre programme qui fonctionne en permanence (le Portmapper) quels types de services RPC il fournira aux clients qui souhaiteront se connecter à lui. Le client de son coté, lorsqu'il voudra utiliser un service RPC, se connectera au portmapper de la machine distante, qui lui donnera les “coordonnées” du service à contacter. Avec ces informations, le client fera un pseudo-appel de procédure (la syntaxe est proche d'un appel de procédure standard), et récupérera de façon transparente le résultat.

RPC définit l'encodage des données à employer (XDR), mais en revanche il peut aussi bien utiliser le mode de transport TCP que le mode de transport UDP. La spécification d'une routine RPC se fait dans un langage qui s'appelle le RPC langage. Un compilateur RPC (rpcgen) va, à partir de ce fichier de définition, générer plusieurs fichiers pour le langage C par exemple : un header qui définit l'ensemble des types, un fichier xdr qui contient les routines XDR, et deux fichiers C qui contiennent l'un le squelette des fonctions à implanter par le serveur, et l'autre le squelette des fonctions à implanter par le client. *Ce mécanisme de fichier de définition compilé est trés général, et ressemble d'ailleurs fort au mécanisme de compilation de données ASN.1*. Nous le retrouverons, sous le nom d'IDL (Interface Definition Language) et de compilateur IDL, dans tous les autre middleware présentés ci-dessous (DCE, CORBA, etc). Il faut donc faire extrèmement attention. IDL est un terme générique, et il y a autant d'IDL différents que de middleware différents.

RPC est un protocole de relativement bas niveau, assez simple d'emploi, mais relativement limité par le standard de codage XDR, et la forme de passage des arguments. Il est très largement employé, en particulier au niveau des interfaces des systèmes d'exploitation (montage de systèmes de fichiers par exemple).

5.2 DCE

DCE (Distributed Computing Environment) est un environnement de programmation distribuée (comme son nom l'indique) défini par l'OSF (Open Software Foundation). OSF est une organisation fondée au départ par IBM, DEC, Apollo, HP, Bull, Nixdorf et Siemens. Elle a pour but de créer des environnements “ouverts” (open software) de développement. DCE est également soutenu par X/Open, une autre organisation qui développe un CAE (Common Applications Environment).

DCE est en fait un assemblage de technologies relativement bien connues séparément :

- des *threads* (DCE Threads) pour faire du multitâches à l'intérieur d'un processus (proches des threads Posix)
- des RPC (DCE RPC) assez semblables dans leur principe aux RPCs de Sun, mais utilisant un autre protocole d'encodage de données (NDR au lieu de XDR), et quelques autres aménagements
- un service de répertoire (DCE Directory Service) qui fournit des informations sur les ressources disponibles. Il est compatible avec la norme X500 (et donc proche du “standard” LDAP)
- un service de temps distribué pour la synchronisation des horloges (pas très éloigné dans son principe du “standard” ntp)
- un service de sécurité (DCE Security Service) qui dérive directement du “standard” Kerberos développé au MIT (mais incompatible avec malgré tout)
- un système de fichier distribué (DFS, ou DCE File System) qui dérive de l'Andrew File System développé à Carnegie Mellon (et incompatible avec NFS, bien qu'il existe des passerelles)

DCE est donc un environnement complet de développement. Il est malheureusement incompatible avec les standards “de fait” actuellement utilisés dans le monde UNIX. Cependant, les RPC DCE sont une refonte plus “propre” et plus générale des RPC Sun. La création de RPC se fait également à travers un IDL (Interface Definition Language) qui est lui même compilé par un compilateur IDL pour générer les fichiers C d’interfaçage.

Une caractéristique qu’il faut immédiatement souligner est l’interopérabilité multi-plateformes. Comme pour les RPC Suns, les applications développées autour de DCE sont totalement compatibles entre elles quelle que soit la plateforme où elles sont exécutées. Une propriété qui s’était perdu, comme nous allons le voir, avec le premier CORBA.

5.3 CORBA

DCE est le descendant direct des langages procéduraux. CORBA est une tentative de refondation complète des systèmes de programmation en se basant totalement sur le paradigme de la programmation objet. CORBA couvre les couches 5/6/7 du modèle OSI.

CORBA n’est qu’un élément parmi les concepts développés par l’OMG (Object Management Group). L’OMG travaille sur de nombreux autres projets : MDA (Model Driven Architecture), UML (Unified Modeling Language), OMA (Object Management Architecture), CWM (Common Warehouse Model) etc.

CORBA signifie Common Object Request Broker. Lorsque l’on programme sous CORBA, chaque unité élémentaire accessible est un objet (au sens de la programmation objet) qui peut se trouver n’importe où sur un réseau de machines. Le noyau de CORBA est l’ORB (Object Request Broker) qui établit les relations entre les différents objets distribués sur le réseau. CORBA est donc là bien différent de DCE, puisque c’est l’ORB (un processus central) qui gère l’ensemble du système. En revanche, comme sous DCE, il existe un IDL (OMG IDL) qui permet la définition des types d’objets et de la forme des interfaces (API), et un compilateur d’IDL qui fabrique les fichiers sources pour le langage que l’on va employer (C, C++, Java, ADA, etc).

La première version de CORBA (CORBA 1.0) n’était pas inter-opérable. Des objets utilisant des ORB différents ne pouvaient pas interagir entre eux. La version 2 de CORBA définit un protocole (IIOP Internet to Internet Operating Protocol) qui est censé permettre l’interopérabilité[5]. Cependant, de l’avis même des développeurs, cette inter-opérabilité reste encore délicate.

CORBA possède bien d’autres propriétés intéressantes, comme la possibilité d’utiliser des interfaces dynamiques d’appel de procédure sans passer par la génération statique classique basée sur l’IDL.

Le but de ce document n’est pas de décrire l’ensemble des propriétés de CORBA. Le lecteur intéressé pourra se reporter à l’excellent cours écrit au LIFL par Geib, Gransart et Merle, disponible en ligne.

CORBA présente certes des qualités : passerelle vers JAVA (vu comme middleware), DCE, COM et DCOM (les modèles de répartition de calcul Microsoft), modularité, indépendance complète vis à vis des langages, machines, réseaux, etc.

Mais CORBA est aussi un système lourd dépendant d’un ORB central, parfois difficile à débugger, et délicat à mettre en œuvre. D’autre part, son modèle à objet

[5]En fait, IIOP est une implantation bâtie sur TCP/IP du protocole générique GIOP (General Interoperable Object Representation) qui définit une représentation commune des données (CDR, Common Data Representation, couche 6).

"sent" (malheureusement) le C++, et hérite donc de pas mal des problèmes de ce triste langage (en particulier au niveau du garbage collecting, des problèmes du passage des paramètres en IN/OUT et de leur copie locale, etc) Son utilisation doit rester limité aux domaines qui ont véritablement besoin des fonctionnalités qu'il fournit. D'autre part, son évolution rapide n'est pas toujours un gage de simplicité de maintenance...

5.4 SOAP, Simple Object Access Protocol

SOAP est un protocole léger qui se base complètement sur XML pour la description des données. L'implantation des RPC avec SOAP n'existe pour l'instant que sous la forme de requètes HTTP, bien que le standard permette d'autres implantations. De façon générale, SOAP est "pensé" essentiellement pour être utilisé en conjonction avec le protocole HTTP. SOAP est soutenu par le consortium W3.

6 Les langages vus comme middleware (couche 6)

Pour l'utilisateur qui ne souhaite pas "mélanger" différents langages lors du développement de ses applications la solution middleware-langages peut être une option intéressante. Certains langages fournissent en effet de façon native des services de type middleware. Là encore, les solutions vont du plus simple au plus compliqué. Nous allons rapidement examiner quelques exemples.

6.1 Ocaml

Object CAML est un langage de la famille ML (Meta Language). Il s'agit donc d'un langage de type fonctionnel, mais disposant d'extensions impératives et d'extensions objet. C'est actuellement le standard pour l'enseignement de l'informatique en France, dans les classes préparatoires, les DEUG et aussi beaucoup d'écoles d'ingénieurs. CAML possède la propriété intéressante de reposer sur un modèle théorique "propre" (le lambda calcul typé) qui permet d'employer un typage fort et automatique, invisible pour l'utilisateur qui n'a jamais besoin de déclarer les variables. Le système de typage de CAML permet un codage des structures de données (quel que soit leur type, leur complexité et leur niveau d'imbrication) en chaîne d'octets, de façon totalement transparente. Ces chaînes peuvent être transférées sur le réseau et immédiatement décodées par une autre application CAML de façon tout aussi transparente. Le typage de données est cependant implicite ; comme en XDR, l'application recevant les données doit connaître à l'avance le type de la donnée qu'elle reçoit[6].

6.2 JAVA

JAVA est un autre exemple de langage pouvant s'utiliser comme middleware. Il s'agit d'un langage possédant des caractéristiques procédurales et objets, relativement proches de C++ (en nettement plus "propre", heureusement). En JAVA, le *code* (et pas seulement des données) est portable d'une machine et d'un système d'exploitation à un autre grâce au principe du bytecode et de la machine virtuelle JAVA.

Le bytecode est une technique ancienne, puisqu'elle remonte au moins au Pascal développé à l'UCSD (University of California San Diego) et qui fut utilisé par exemple

[6]Pour une comparaison des différents langages en terme d'efficacité etc, on ne peut que conseiller l'excellent site http ://www.bagley.org/~ doug/shootout

sur l'Apple II au début des années 80. Il n'est peut-être pas inutile de rappeler quelques notions de base sur les langages de programmation ; un langage peut en effet être :

interprété : il s'agit de langages comme les langages de script (sh, awk, perl, php). Un interpréteur de commandes lit le fichier source et l'exécute au fur et à mesure. Le code est donc complètement portable puisqu'il reste en permanence en format source, mais l'exécution est lente, et les erreurs ne peuvent être détectées qu'à l'exécution.

compilé : un programme spécial, le compilateur, transforme le fichier source en code machine directement exécutable par le processeur. Ce code est beaucoup plus rapide, et beaucoup d'erreurs vont être éliminées par le compilateur, mais le code obtenu n'est pas portable d'une machine à l'autre.

Le bytecode est une technique intermédiaire entre interprétation et compilation. Comme pour un langage compilé, le code source est transformé par un compilateur, qui le vérifie et élimine les erreurs. Mais le résultat n'est pas du code machine directement exécutable par le processeur, mais un code intermédiaire, le bytecode, qui sera lui interprété par une machine virtuelle. Cette machine va dépendre du processeur, du système d'exploitation, etc. En revanche, le bytecode va lui rester indépendant des architectures physiques.

Il existe ainsi une machine virtuelle JAVA par type de machine et de système d'exploitation, mais le code JAVA peut se "déplacer" sur le réseau de façon totalement transparente. C'est ce qui en a fait le langage de choix pour la programmation des applications du WEB. Le développeur du code place sur son site un code (l'applet JAVA) qui va être téléchargé dans le navigateur et exécuté localement par la machine virtuelle JAVA.

JAVA peut donc être utilisée aussi comme middleware.

7 Un middleware spécifique pour les simulations : HLA

HLA (High Level Architecture) est un standard de description de données et d'interopérabilité pour la simulation. La définition de base de HLA remonte à 1996 et le standard a été définitivement fixé par l'IEEE en 2000. HLA a été développé par le département de la defense américain (DoD) et est désormais imposé comme standard pour tous les nouvelles évolutions de simulateur au DoD. La MITRE corporation a eu sa part dans l'évolution de HLA, et il est également adapté au développement de simulations ATC.

HLA est inspiré des modèles à objets, mais, suivant la description même du standard HLA, il réduit trés fortement la sémantique des modèles à objets, contrairement à CORBA, par exemple. HLA ne permet que la description des attributs des objets, et la description des interactions entre objets. Un objet est forcément "réel" (un char, un avion, etc). HLA repose en partie sur un mécanisme d'abonnements pour informer les "fédérés" (c'est ainsi que l'on appelle les différents acteurs opérants à l'intérieur d'un environnement) des interactions qui sont susceptibles de les intéresser (ce mécanisme ressemble à la notion d'évènements dans les modèles à objets). Pour éviter un effondrement du système, le mécanisme d'abonnements doit être géré de façon "intelligente" : les clients ne doivent pas s'abonner à tous les évènements, mais seulement à ceux susceptibles de les intéresser.

Au niveau de l'architecture, le standard HLA repose sur un logiciel, le RTI (Real Time Infrastructure) qui gère l'ensemble de la simulation : nommage des objets, etc.

Le RTI est souvent décomposé en un serveur central le RTIG, et des RTI locaux gérant chacun un fédéré. Le RTI gère l'ensemble des couches "basses" et fournit donc une interface au niveau applicatif seulement. Il existe plusieurs implantations de RTI. Le DoD a longtemps fourni une implantation gratuite de RTI, ce qui n'est plus le cas aujourd'hui. Il existe cependant des implantations plus ou moins complètes du standard dans le domaine public (dont une fournie par l'ONERA/CERT en C++ (CERTI), et une en ADA (yaRTI)). Il existe aussi des implantations commerciales, dont certaines utilisent CORBA pour la gestion des couches basses et des services de nommage. Ce choix (CORBA) est assez largement critiqué par la communauté universitaire (on peut se reporter aux articles publiés par le CERT sur ce sujet) pour l'implantation de RTI, en raison de sa lourdeur (on n'utilise qu'une fraction de CORBA pour implanter le RTI), mais aussi en raison des risques de conflits avec des logiciels utilisant simultanément le RTI et CORBA pour d'autres besoins.

Le standard HLA est relativement simple dans son principe. La maturité des RTI est plus discutable.

8 Et ce n'est pas tout…

Il est difficile de terminer ce document sans évoquer des systèmes aussi répandus que PVM (Parallel Virtual Machine), MPI (Message Passing Interface) ou encore des technologies autour du "Grid computing" comme Globus ou Legion.

Tous ces projets sont plutôt utilisés dans le monde du calcul massivement parallèle, mais MPI ou surtout PVM sont des alternatives très agréables et simples d'emplois pour le passage de données entre applications. MPI et PVM sont très employés, et Globus est aujourd'hui soutenu par IBM qui y voit un middleware utilisable sur ses super-calculateurs comme l'ASCI White (12.3 Teraflops) qui n'est finalement qu'un cluster de 512 calculateurs RS/6000.

9 Conclusion

L'idée force que nous souhaitons voir se dégager de ce document est à la fois la grande variété des middleware disponibles et surtout la grande diversité des fonctionnalités qu'ils peuvent fournir. Il faut donc bien comprendre que le choix d'un middleware, ou d'un type de middleware, doit se faire à partir des besoins exprimés. Employer un protocole UDP avec un codage ASCII pour déclencher des procédures différentes avec des types de données variables conduira certainement à des problèmes de développement et d'évolution des applications. Mais, de la même façon, utiliser un système de distributions d'objets pour échanger quelques messages en format fixe entre applications sera tout aussi inutile et surtout coûteux en terme de développement et de maintenabilité. Il n'y a pas de solution unique, mais des solutions adaptées aux besoins.

Il faut aussi se méfier d'idées toutes faites et techniquement fausses comme "il suffit d'employer le même middleware pour être interopérable", ou "l'emploi de middlewares différents interdit l'interopérabilité", etc. La mise en relation de logiciels conçus par des équipes différentes ne sera simple que si les APIs et/ou les IDLs des procédures ou objets à mettre en commun ont été identifiés dès l'origine et de façon commune. Dans le cas contraire, il faudra de toute façon repenser la connexion en terme de middleware, et en fonction du couplage, choisir une mode de connexion adapté qui ne sera pas nécessairement celui employé pour le développement interne, surtout si le niveau

de couplage est faible. Les arguments développés par les vendeurs de middleware dans ce domaine à grand coups de buzzwords sont tendancieux, et largement dementis par l'expérience. Il faut là aussi se garder des effets de mode et savoir baser ses choix sur une solide connaissance technique et un ordinaire bon sens.

Lexique

ADA : langage de programmation procédural fortement typé, multi-tâches, résultat d'un processus de spécifications très poussé demandé par le Department of Defense des Etats Unis, développé au début des années 80. Il devait servir au codage de l'ensemble des applications du DoD. Pour des raisons essentiellement liées, à ses débuts, à l'immaturité des compilateurs, à leur mauvais interfaçage avec UNIX, mais aussi à la réticence de programmeurs peu enclins à changer leurs (mauvaises ?) habitudes, ADA n'a jamais réussi à s'imposer complètement face au C, puis à ses avatars objets (C++, JAVA, etc). ADA souffre aussi d'une grande complexité (le manuel de référence est monstrueux), liée à l'absence d'un modèle théorique "propre"

ALGOL : ALGOrithmic Language, langage apparu en 1960, à l'origine duquel on trouve John Backus (pour ALGOL60) mais auquel participeront Nicklaus Wirth, John Hoare, etc ALGOL60 est la première tentative de programmation structurée typée. ALGOL68 (1968) sera l'objet de nombreux débats en raison de sa grande complexité, et verra le départ de Wirth qui créera Pascal

AFS : Andrew File System, système de partage de fichiers à travers un réseau, développé par l'Université de Carnegie Mellon, base du DFS

API : Application Programming Interface, définition des interfaces de programmation d'un système quelconque. L'utilisation d'API permet de rendre accessible à l'utilisateur les fonctionnalités du système tout en lui masquant l'implantation interne

ARPA : Advanced Research Program Agency, agence dépendant du DoD en charge des programmes de recherches avancées

ARPANET : ARPA Network, réseau de communications développé par l'ARPA au début des années 70, qui est l'ancêtre d'Internet

ASCI White : super calculateur atteignant 12 Teraflops en pointe, développé par IBM. Il s'agit en fait d'un cluster de 512 processeurs RS/6000

ASCII : American Standard Code for Information Interchange, norme de codage sous forme d'un octet des caractères imprimables

ASN.1 : Abstract Syntax Notation One, norme de description de données (couche 7), mais aussi middleware couche 6, normalisée par l'ISO

ATN : Aeronautical Telecommunication Network, futur réseau de télécommunication aéronautique

Apollo : fabricant d'ordinateurs, pionnier dans la fabrication des stations de travail, disparu aujourd'hui

BCD : Binary Coded Decimal, format de codage des nombres où chaque chiffre est codé séparément en binaire. Par exemple 27 en représentation binaire "classique" s'écrira 11011 ($1 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$), mais s'écrira 100111 (codage de 2 (10) suivi du codage de 7 sur 4 bits (0111)) en BCD

BER : Basic Encoding Rule, format d'encodage de base des données en ASN.1

Broadcast : technique consistant à diffuser à l'ensemble des acteurs présents les données. Opposé de connexions point à point

Bull : fabricant d'ordinateurs français qui n'en finit pas de disparaitre

Buzzword : définition précise : "a usually important-sounding word or phrase used primarily to impress laypersons", traduction française libre : mot qui fait tilt (buzz) dans l'esprit de certains décideurs qui croient que l'informatique se résume à la lecture du Monde Informatique. Je conseille fortement à ceux qui s'ennuient en réunion l'utilisation de grilles de Business Buzzword Bingo (http ://isd.usc.edu/˜ karl/Bingo/about.html). Notre collègue Nicolas Dubois en développa d'excellentes pour l'Air Traffic Management

C : langage procédural faiblement typé développé par Brian Kernighan et Dennis Ritchie au début des années 70, pour servir de langage de programmation au système d'exploitation UNIX de Ken Thompson, et descendant des langages B et BCPL (non typé). C était un langage intéressant pour l'époque, mais qui porte les stigmates de son âge (une des première version de C devait fonctionner avec 24Ko de mémoire sur un PDP-11...). La grande force de C est son intégration parfaite avec UNIX. On dit souvent du C qu'il s'agit d'un assembleur indépendant de la machine...

C++ : langage verrue de type objet défini par Bjarne Stroustrup au dessus de C pour implanter les concepts objets tout en conservant la compatibilité avec C. C++ est certainement l'exemple de tout ce qu'il faut éviter (pas de garbage collecting, gestion de la mémoire "à la C", surcharge d'opérateurs faite n'importe comment, utilisation des infâmes constructions pragma, etc)...Son succès est une nouvelle démonstration du fait que les meilleurs ne sont pas nécessairement ceux qui réussissent (voir DOS et CP/M, Unix et VMS, etc)

CAE : Common Applications Environment

CAML : version du langage ML développé par l'INRIA, et dont l'exécution était basée sur une machine abstraite, appelée la CAM (Categoric Abstract Machine). Voir ML

CCITT : Consultative Committee for International Telephony and Telegraphy, organisme de normalisation

CDR : Common Data Representation, un des protocoles niveau 6 de CORBA

COM : middleware Microsoft

CORBA : Common Object Request Broker Architecture, voir définition détaillée dans le texte

CWM : Common Warehouse MetaModel, une des usines à gaz de l'OMG, dont la définition même contient 3 buzzwords par ligne (*The Common Warehouse Metamodel (CWM) is a specification that describes metadata interchange among data warehousing, business intelligence, knowledge management and portal technologies*). Fermez le ban

DAP : Directory Access Protocol, protocole d'accès à des carnets d'adresses, de noms, de protocoles, etc. Une version légère du protocole DAP est LDAP

DCE : Distributed Computing Environment, environnement de programmation distribuée (comme son nom l'indique) défini par l'OSF

DCOM : middleware Microsoft. Voir COM

DEC : Digital Equipment Corporation, fabriquant d'ordinateurs (essentiellement de mini ordinateurs, la série des PDP, puis des VAX) depuis les années 60, qui finit par disparaitre (absorbé par Compaq) dans les années 90, victime du succès des stations de travail, puis surtout des PC. Champion de VMS

DFS : DCE File System, système de partage de fichiers développé dans le cadre de DCE, et basé sur AFS

Dod : Department of Defense, ministère de la défense des Etats Unis

DTD : Definition Type Document, document écrit suivant la norme SGML définissant la syntaxe que devront suivre d'autres documents qui doivent justement respecter ce DTD. Cela parait compliqué mais ça ne l'est pas. Par exemple, la norme HTML est un DTD

EBCDIC : Extended Binary Coded Decimal Interchange Code, code de représentation des caractères longtemps utilisé par IBM

Ethernet : protocole réseau (couche 2 OSI) inventé par Metcalf au centre de recherches Xerox Parc au début des années 70. Révolutionnaire pour sa vitesse, il reste encore aujourd'hui la base de la majorité des réseaux locaux (LAN)

FTP : File Transfer Protocol, protocole de transfert de fichiers sur Internet

GIOP : General Interoperable Object Representation, protocole général de représentation des objets pour CORBA, censé garantir l'interopérabilité des différents ORB

GML : Generalized Markup Language, ancêtre de SGML

Globus : middleware utilisé surtout dans le monde du calcul scientifique hautes performances, de type Grid Computing

Grid computing : modèle général de parallélisme qui s'intéresse surtout à l'utilisation de clusters de machines reliées par réseau spécialisé à très haut débit. L'ASCI White d'IBM est un bon exemple de ce type de calculateurs

HLA : High Level Architecture, standard définissant une architecture générale pour la simulation, définie par le DoD. L'implantation de HLA repose sur un RTI.

HP : Hewlett-Packard, firme fabricant essentiellement des ordinateurs et des machines à calculer (HP fabriqua la première machine à calculer portable programmable), et qui réussit à survivre encore aujourd'hui

HTML : Hyper Text Markup Language, langage à balisage, qui est en fait un DTD de SGML

HTTP : Hyper Text Transfer Protocol, protocole de transfert pour les fichiers HTML, base de fonctionnement des clients/serveurs sur le Web

IBM : International Business Machine, dit aussi Big Blue, firme fondée par Herman Hollerith, qui fabriqua d'abord des machines pour cartes perforées, avant de commencer à fabriquer des ordinateurs à la fin des années 40 (les modèles 701,702, 704). Contrairement à une idée reçue, c'est UNIVAC qui réalisa les premiers ordinateurs commerciaux, et non IBM

IDL : Interface Definition Language, langage permettant de définir les types de données et les squelettes des procédures de façon indépendante du langage de programmation qui sera utilisé par la suite. Il y a autant d'IDL que de middleware

IIOP : Internet to Internet Object Protocol, implantation du protocole GIOP sur TCP/IP

IP : Internet Protocole, couche 3 OSI pour Internet évidemment

ISO : International Organization for Standardization, autre organisme de standardisation

Internet : ensemble de machines interconnectées et utilisant le même protocole de niveau 3 (IP). Le père d'Internet est Arpanet

JAVA : langage de programmation de type objet (mais acceptant les constructions procédurales) développé par Sun et reprenant une syntaxe proche de celle du C. JAVA est cependant nettement plus "propre" que son cousin C++, sans atteindre toutefois l'élégance du aujourd'hui presque défunt langage Eiffel de Bertand Meyer. La légende dit que JAVA fut inventé/développé par une équipe de Sun qui travaillait près d'une machine à café (JAVA=KAWA pour nous français)

Kerberos : protocole de sécurité développé dans les années 80 au MIT, permettant de contrôler les accès à tout un ensemble de services, du login au partage de fichier. Réutilisé dans DCE, tire son nom de Cerbère (Kerberos en grec, le chien qui gardait la porte des enfers dans la dite mythologie)

LAN : Local Area network (réseau local), réseau de machines interconnectées dans un périmètre "réduit" (une entreprise, une école, etc). Ethernet est un support physique traditionnel pour ce type de réseau. Opposé de WAN (Wide Area Network)

LaTeX : ensemble de macros (étendant le système TeX) développé par Leslie Lamport (plus connu dans le monde informatique pour ses travaux sur les problèmes d'horloges synchrones, connues d'ailleurs sous le nom d'horloges de Lamport)

LDAP : Lightweight Directory Access Protocol, système "léger" d'accès aux carnets d'adresses et aures repertoires définis par la norme X.500. Petit cousin de DAP

Legion : un des middleware faisant du Grid Computing

LISP : : LISt Processing, langage dévelopé par John McCarthy au MIT dans les années 50. Langage fonctionnel, non typé, qui fut largement utilisé par les programmes d'intelligence artificielle

MDA : OMG Model Driven Architecture, principes de design architecturaux basés sur les produits de l'OMG

MIT : Massachussets Institute Of Technology, grosse université américaine (Boston)

ML : Meta Language ; famille de langages apparue à la fin des années 70 et basée sur un modèle formel, le lambda calcul typé. ML est un langage de type fonctionnel, mais contenant des extensions procédurales, fortement typé, mais de façon implicite (le programmeur n'a jamais besoin de déclarer une variable). Les implantations les plus récentes de ML (OCAML et SML) sont extrèmement rapides (voir http ://www.bagley.org/~ doug/shootout), et le manuel de référence du langage est réduit à une vingtaine de pages, en raison de son extrème simplicité, bien qu'il implante la quasi-totalité des concepts modernes de programmation. Son seul défaut est qu'il oblige le programmeur habitué aux langages procéduraux à repenser totalement sa façon de faire. . .

MOM : Message Oriented Middleware, middleware s'occupant essentiellement de passages de messages entre applications. Exemples : MPI, PVM

MPI : Message Passing Interface, middleware réalisant du passage de messages entre applications. Cousin évolué (et plus complexe) de PVM

NDR : Network Data Representation, système de présentation des données pour les RPC de DCE. Cousin des XDR de Sun

NFS : Network File System, protocole de partage de fichiers le plus répandu dans le monde Unix

OMA : Object Management Architecture, voir OMG, MDA, CWM, etc

OMG : Object Management Group, association ayant pour but de développer et promouvoir tout un ensemble d'outils basés sur la programmation objet (MDA, UML, OMA, CWM, CORBA). Semble avoir une affection marquée pour les buzzwords

ORB : Object Request Broker, élément central de l'architecture CORBA, assurant les communications et les synchronisations entre objets. Depuis la norme Corba 2, les ORB peuvent eux-même communiquer entre eux (en principe)

OSF : Open Software Foundation ; organisation fondée au départ par IBM, DEC, Apollo, HP, Bull, Nixdorf et Siemens. Elle a pour but de créer des environnements "ouverts" (open software) de développement

OSI : Open System Inteconnection, norme ISO qui a pour but de séparer les différents niveaux (du niveau physique brut au niveau applicatif) liés à l'interconnexion des systèmes

Ocaml : Object CAML, version étendant le langage CAML par des concepts objets de façon élégante

PC : Personal Computer, terme employé par IBM lors du lancement de l'IBM PC à la fin des années 70. Devenu un terme générique depuis, recouvrant en général toute machine équipé d'un processeur de type Intel 80x86 ou compatible, et dont le BIOS émule le BIOS original du PC d'IBM

PVM : Parallel Virtual Machine, middleware de communication par messages, simple d'emploi

Pascal : langage de programmation inventé par Niklaus Wirth sur les ruines des échecs des différents langages ALGOL ; Pascal est un langage procédural fortement typé, qui a surtout été employé pour l'enseignement de la programmation

Portmapper : processus au centre du fonctionnement des RPC de Sun. Il assure l'enregistrement des différents services RPC disponibles sur une machine et assure la diffusion de l'information aux clients

RFC : Request For Comments, document qui, comme son nom ne l'indique pas, définit les standards de fait utilisés sur l'Internet

RPC : Remote Procedure Call, modèle consistant à distribuer les procédures sur un réseau, et à permettre leur appel depuis n'importe quelle machine. Il en existe deux grandes implantations, celle faite par Sun et celle de DCE

RTI : Real Time Infrastructure, le RTI est le logiciel qui permet le fonctionnement d'un système basé sur le standard HLA. le RTI gère l'ensemble des couches basses, le système de nommage et d'abonnements aux évènements, etc

SABRE : système de réservation développé par IBM pour les grandes compagnies aériennes américaines dans les années 60. SABRE était basé sur l'expérience de SAGE, et existe encore aujourd'hui

SAGE : Semi Automatic Ground Environment, système informatique de surveillance de l'espace aérien américain contre les intrusions des bombardiers soviétiques développé dans les années 50. Véritable mine d'innovations technologiques (modems, réseaux d'ordinateurs, stylos lumineux), les machines Whirlwind-II qui furent conçues par Jay Forrester au MIT puis développées par IBM sont celles

qui eurent la plus grande longévité de l'histoire de l'informatique. Mises en route à la fin des années 50, la dernière fut arrétée en 1983

SGML : Standard Generalized Markup Language (normalise en 1986), standard des langages à balisage dont les ancêtres furent GML et TeX. SGML est un méta-langage qui permet d'écrire des DTD. Les DTD définissent la syntaxe qui sera autorisée pour un langage donné. On peut definir non seulement le jeu de balises mais aussi la syntaxe de base (Reference Concrete Syntax), ce qui fait qu'un programme Lisp peut apparaitre comme des donnees SGML. SGML est apparu comme très (voire trop) complexe ce qui a limite son utilisation. Voir DTD

SOAP : Simple Object Access Protocol, protocole léger basé sur XML pour la description des données, et s'articulant autour de HTTP.

Sun : constructeur d'ordinateurs fondé par des dissidents d'Apollo au début des années 80, et fabriquant essentiellement des stations de travail. Sun a également introduit un certain nombre d'innovations technologiques : RPC, JAVA, etc

TEX : Typesetting Language développé par Donald Knuth à la fin des années 60. Ancêtre des langages à balisage, conçu pour être indépendant des supports physiques de représentation, en association avec son langage de construction/définition de fonts, Metafont. TEX est encore bien vivant aujourd'hui dans le monde universitaire, et a connu de nombreuses extensions dont LATEX. Un fichier TEX des années 60 est encore compilable et utilisable aujourd'hui. Donald Knuth est bien connu dans le monde de la recherche pour le "fameux" théorème de Knuth et Bendix, et pour son encyclopédie "The Art Of Computer programming".

Typage : le typage est l'opération qui consiste à assigner un type à un objet et à vérifier que l'on ne "mélange" pas des objets de types différents. Les langages de programmation peuvent implanter différents niveaux de typage ; par exemple le LISP est non typé (on peut mettre dans une même liste des entiers, des flottants et des caractères, c'est à la charge du programmeur de vérifier que tout se passera bien), le C est faiblement typé (on n'est pas censé pouvoir mélanger des objets de type différents, mais il est très simple de le faire), ADA et ML sont des exemples de langage fortement typé (de deux façons différentes : explicite pour ADA, implicite pour ML) qui interdisent strictement le mélange d'objets de types différents

UCSD : University of California San Diego

UDP : User Datagram Protocol, protocole de broadcast au dessus de IP

UML : Unified Modeling Language, langage de modélisation censé améliorer la qualité du développement...Un des produits de l'OMG. Attention à ne pas confondre UML et IDL...

UNIX : système d'exploitation développé par Ken Thompson dans les laboratoires de recherche Bell au début des années 70. Unix est le fils spirituel de Multics, un des premiers systèmes d'exploitation multi-tâches, développé en collaboration par le MIT (qui avait déjà l'expérience Concurrent Time Sharing System, CTSS, le premier exemple de système d'exploitation multi-tâches), General Electrics et Bell. Thompson développa Unix lorsque Bell se retira de Multics. Le premier Unix fut développé sur un DEC PDP-7 avec 8Ko de mémoire, écrit en langage B. La seconde version, développée sur PDP-11, fut écrite en C

VMS : système d'exploitation développé par DEC pour ses mini-ordinateurs. Probablement supérieur à Unix, il mourut victime d'un mauvais marketting, et de la disparition de DEC

W3C : le World Wide Web Consortium s'occupe de la gestion des "standards" du Web (spécifications, logiciels, outils)

WEB : si vous ne savez pas ce que c'est, inquiétez-vous...

WYSIWYG : What You See Is What You Get, modèle de traitement de texte développé par Xerox à la fin des années 70. L'idée était d'avoir une représentation graphique immédiate de l'aspect final du document au fur et a mesure qu'on le tapait

X-Windows : système graphique développé essentiellement pour les stations de travail au début des années 80 au MIT, suivant un modèle client-serveur. Le serveur X assure la gestion de l'ensemble des périphériques graphiques (clavier, écran, souris, etc) tandis que les clients (les applications graphiques) utilisent uniquement ses API. Les clients peuvent se trouver en un point quelconque du réseau, et pas nécessairement sur la machine où s'exécute le serveur. X-Windows est toujours bien vivant aujourd'hui dans le monde Unix/Linux. C'est une forme de middleware...

XDR : eXternal Data Representation, couche 6 OSI utilisée par les RPC de Sun. Cousin de NDR de DCE

XML : eXtended Markup Language (1998) ; la finalité de XML est comparable à celle de SGML : c'est un meta-langage pour fabriquer des langages de donnees à balises ; XML a simplifié la logique SGML et figé la syntaxe de base pour écrire les donnees. XML devrait devenir le langage d'échange des données sur le Web, en remplaçant progressivement HTML.

Xerox : plus connu pour ses photocopieurs, le centre de recherche Xerox de Palo Alto a pourtant été à la base d'un très grand nombre d'avancées dans le domaine informatique : Ethernet, les premières vraies interfaces graphiques, les premiers systèmes WYSIWYG

X.500 : recommendation CCITT décrivant la structure, les modèles, les concepts et les services que doit fournir un répertoire, de quelque type que ce soit. Voir DAP et LDAP

Remerciements

Merci à tous ceux qui ont bien voulu relire et commenter les versions successives de cette note, et tout particulièrement Pascal Brisset, Dominique Colin de Verdière, Didier Pavet, Jean-Marc Pomeret et Muriel Preux.