Avant-propos

Dans les tout débuts de l'informatique, le fonctionnement « intime » des processeurs décidait toujours, en fin de compte, de la seule manière efficace de programmer un ordinateur. Alors que l'on acceptait tout programme comme une suite logique d'instructions, il était admis que l'organisation du programme et la nature même de ces instructions ne pouvaient s'éloigner de la façon dont le processeur les exécutait : pour l'essentiel, des modifications de données mémorisées, des déplacements de ces données d'un emplacement mémoire à un autre, et des opérations d'arithmétique et de logique élémentaire.

La mise au point d'algorithmes complexes, dépassant les simples opérations mathématiques et les simples opérations de stockage et de récupérations de données, obligea les informaticiens à effectuer un premier saut dans l'abstrait, en inventant un style de langage dit procédural, auquel appartiennent les langages Fortran, Cobol, Basic, Pascal, C... Ces langages permettaient à ces informaticiens de prendre quelques distances par rapport au fonctionnement intime des processeurs (en ne travaillant plus directement à partir des adresses mémoire et en évitant la manipulation directe des instructions élémentaires) et d'élaborer une écriture de programmes plus proches de la manière naturelle de poser et de résoudre les problèmes. Les codes écrits dans ces langages devenant indépendants en cela des instructions élémentaires propres à chaque type de processeur. Ces langages cherchaient à se positionner quelque part entre l'écriture des instructions élémentaires et l'utilisation tant du langage naturel que du sens commun. Il est incontestablement plus simple d'écrire : c = a + b qu'une suite d'instructions telles que : `"load a, reg1"`, `"load b, reg2"`, `"add reg3, reg1, reg2"`, `"move c, reg3"`, ayant pourtant la même finalité. Une opération de traduction automatique, dite de compilation, se charge alors de traduire le programme, écrit au départ dans ce nouveau langage, dans les instructions élémentaires, seules comprises par le processeur. Cette montée en abstraction permise par ces langages de programmation présente un double avantage : une facilitation d'écriture et de résolution algorithmique, ainsi qu'une indépendance accrue par rapport aux différents types de processeur existant aujourd'hui sur le marché.

Plus les problèmes à affronter gagnaient en complexité – comptabilité, jeux automatiques, compréhension et traduction des langues naturelles, aide à la décision, bureautique, conception et enseignement assistés, programmes graphiques, etc. –, plus l'architecture et le fonctionnement des processeurs semblaient contraignants, et plus il devenait vital d'inventer des mécanismes informatiques simples à mettre en œuvre, permettant une réduction de cette complexité et un rapprochement encore plus marqué de l'écriture des programmes des manières humaines de poser et de résoudre les problèmes.

Avec l'intelligence artificielle, l'informatique s'inspira de notre mode cognitif d'organisation des connaissances, comme un ensemble d'objets conceptuels entrant dans un réseau de dépendance et se structurant de manière taxonomique. Avec la systémique ou la bioinformatique, l'informatique nous révéla qu'un ensemble d'agents au fonctionnement élémentaire, mais s'influençant mutuellement, peut produire un comportement émergent d'une surprenante complexité. La complexité affichée par le comportement d'un système observé dans sa

globalité ne témoigne pas systématiquement d'une complexité équivalente lorsque l'attention est portée sur chacune des parties composant ce système et prise isolément. Dès lors, pour comprendre jusqu'à reproduire ce comportement par le biais informatique, la meilleure approche consiste en une découpe adéquate du système en ses parties et une attention limitée au fonctionnement de chacune d'entre elle.

Tout cela mis ensemble : la nécessaire distanciation par rapport au fonctionnement du processeur, la volonté de rapprocher la programmation du mode cognitif de résolution de problème, les percées de l'intelligence artificielle et de la bio-informatique, le découpage comme voie de simplification des systèmes apparement complexes, conduisit graduellement à un deuxième style de langage de programmation, un tout petit peu plus récent, bien que fêtant ses 45 ans d'existence (l'antiquité à l'échelle informatique) : les langages orientés objets, tels Simula, Smalltalk, C++, Eiffel, Java, C#, Delphi, Power Builder, Python et bien d'autres...

L'orientation objet en deux mots

À la différence de la programmation procédurale, un programme écrit dans un langage objet répartit l'effort de résolution de problèmes sur un ensemble d'objets collaborant par envoi de messages. Chaque objet se décrit par un ensemble d'attributs (partie statique) et un ensemble de méthodes portant sur ces attributs (partie dynamique). Certains de ces attributs étant l'adresse des objets avec lesquels les premiers collaborent, il leur est possible de déléguer certaines des tâches à leurs collaborateurs. Le tout s'opère en respectant un principe de distribution des responsabilités on ne peut plus simple, chaque objet s'occupant de ses propres attributs. Lorsqu'un objet exige de s'informer ou de modifier les attributs d'un autre, il charge cet autre de s'acquitter de cette tâche. Cette programmation est fondamentalement distribuée, modularisée et décentralisée. Pour autant qu'elle respecte également des principes de confinement et d'accès limité (dit d'encapsulation) que nous décrirons dans l'ouvrage, cette répartition modulaire a également l'insigne avantage de favoriser la stabilité des développements, en restreignant au maximum l'impact de modifications apportées au code au cours du temps. Ces impacts seront limités aux seuls objets qu'ils concernent et à aucun de leurs collaborateurs, même si le comportement de ces derniers dépend en partie des fonctionnalités affectées.

Ces améliorations, résultant de la prise de conscience des problèmes posés par l'industrie du logiciel ces dernières années, complexité accrue et stabilité dégradée, ont enrichi la syntaxe des langages objet.. Un autre mécanisme de modularisation inhérent à l'orienté objet est l'héritage qui permet à la programmation de refléter l'organisation taxonomique de notre connaissance en une hiérarchie de concepts du plus au moins général. À nouveau, cette organisation modulaire en objets génériques et plus spécialistes est à l'origine d'une simplification de la programmation, d'une économie d'écriture et de la création de zone de code aux modifications confinées. Tant cet héritage que la répartition des tâches entre les objets permet, tout à la fois, une décomposition plus naturelle des problèmes,une réutilisation facilitée des codes déjà existants, et une maintenance facilitée et allégée de ces derniers. L'orientation objet s'impose, non pas comme une panacée universelle, mais une évolution naturelle, au départ de la programmation procédurale, qui facilite l'écriture de programmes, les rendant plus gérables, plus compréhensibles, plus stables et réexploitables.

L'orienté objet inscrit la programmation dans une démarche somme toute très classique pour affronter la complexité de quelque problème qui soit : une découpe naturelle et intuitive en des parties plus simples. A fortiori, cette découpe sera d'autant plus intuitive qu'elle s'inspire de notre manière « cognitive » de découper la réalité qui nous entoure. L'héritage, reflet fidèle de notre organisation cognitive, en est le témoignage le plus éclatant. L'approche procédurale rendait cette découpe moins naturelle, plus « forcée ». Si de nombreux adeptes de la programmation procédurale sont en effet conscients qu'une manière incontournable de

simplifier le développement d'un programme complexe est de le découper physiquement, ils souffrent de l'absence d'une prise en compte naturelle et syntaxique de cette découpe dans les langages de programmation utilisés. Dans un programme imposant, l'OO permet de tracer les pointillés que les ciseaux doivent suivre là où il semble le plus naturel de les tracer : au niveau du cou, des épaules ou de la ceinture, et non pas au niveau des sourcils, des biceps ou des mollets. De surcroît, cette pratique de la programmation incite à cette découpe suivant deux dimensions orthogonales : horizontalement, les classes se déléguant mutuellement un ensemble de services, verticalement, les classes héritant entre elles d'attributs et de méthodes installés à différents niveaux d'une hiérarchie taxonomique. Pour chacune de ces dimensions, reproduisant fidèlement nos mécanismes cognitifs de conceptualisation, en plus de simplifier l'écriture des codes, il est important de faciliter la récupération de ces parties dans de nouveaux contextes et d'assurer la robustesse de ces parties aux changements survenus dans d'autres. Un code OO, idéalement, sera aussi simple à créer qu'à maintenir, récupérer et faire évoluer.

Il est parfaitement inconséquent d'opposer le procédural à l'OO car, in fine, toute programmation des méthodes (c'est-à-dire la partie active des classes et des objets) reste totalement tributaire des mécanismes procéduraux. On y rencontre des variables, des arguments, des boucles, des arguments de fonction, des instructions conditionnelles, tout ce que l'on trouve classiquement dans les boîtes à outils procédurales. L'OO ne permet en rien de faire l'économie du procédural, simplement, il complémente celui-ci, en lui superposant un système de découpe plus naturel et facile à mettre en œuvre. Il n'est guère surprenant que la plupart des langages procéduraux comme le C, Cobol ou, plus récemment, PHP, se soient relativement aisément enrichi d'une couche dite OO sans que cette addition ne remette sérieusement en question l'existant procédural. Cependant, l'impact de cette couche additionnelle ne se limite pas à quelques structures de données supplémentaires afin de mieux organiser les informations manipulées par le programme. Il va bien au-delà. C'est toute une manière de concevoir un programme et la répartition de ses parties fonctionnelles qui est en jeu. Les fonctions et les données ne sont plus d'un seul tenant mais éclatées en un ensemble de modules reprenant, chacun, une sous-partie de ces données et les seules fonctions qui les manipulent. Il faut réapprendre à programmer en s'essayant au développement d'une succession de micro-programmes et au couplage soigné et réduit au minimum de ces micro-programmes. En substance, la programmation OO pourrait reprendre à son compte ce slogan devenu très célèbre parmi les adeptes des courants altermondialistes : « agir localement, penser globalement ». Se pose alors la question de stratégie pédagogique, question très controversée dans l'enseignement de l'informatique aujourd'hui, sur l'ordre chronologique à donner au procédural et à l'OO. De nombreux enseignants de la programmation, soutenus en cela par de très nombreux manuels de programmation, considèrent qu'il faut d'abord passer par un enseignement intensif et une maîtrise parfaite du procédural, avant de faire le grand saut vers l'OO. Quinze années d'enseignement de la programmation à des étudiants de tout âge et de toute condition (de 7 à 77 ans, issus des sciences humaines ou exactes) nous ont convaincu qu'il n'y a aucun ordre à donner. De même qu'historiquement, l'OO est né quasiment en même temps que le procédural et en complément de celui-ci, l'OO doit s'enseigner conjointement et en complément du procédural. Il faut enseigner les instructions de contrôle en même temps que la découpe en classe. Tout comme un cours de cuisine s'attardant sur quelques ingrédients culinaires très particulier parallèlement à la manière dont ces ingrédients doivent s'harmoniser, ou un cours de mécanique automobile se focalisant sur quelques pièces ou mécanismes en particulier en même temps que le plan et le fonctionnement d'ensemble, l'enseignement de la programmation doit mélanger à loisir la perception « micro » des mécanismes procéduraux à la vision « macro » de la découpe en objets. Aujourd'hui, tout projet informatique de dimension conséquente débute par une analyse des différentes classes qui le constituent. Il faut aborder l'enseignement de la programmation tout comme débute la prise en charge de ce type de projet, en enseignant au plus vite la manière dont ces classes et les objets qui en résultent opèrent à l'intérieur d'un programme.

L'orienté objet s'est trouvé à l'origine ces dernières années, compétition oblige, d'une explosion de technologies différentes, mais toutes intégrant à leur manière les mécanismes de base de l'OO : classes, objets, envois de messages, héritage, encapsulation, polymorphisme... Ainsi sont apparus une multitude de langages de programmation, qui intègrent ces mécanismes de base à leur manière, à partir d'une syntaxe dont les différences sont soit purement cosmétiques, soit légèrement plus subtiles. Ils sont autant de variations sur le ou les thèmes créés par leurs trois principaux précurseurs : Simula, Smalltalk et C++.

L'OO a également permis de repenser trois des chapitres les plus important de l'informatique de ces deux dernières décennies. Tout d'abord, le besoin d'une méthode de modélisation graphique débouchant sur un niveau d'abstraction encore supplémentaire (on ne programme plus, on dessine un ensemble de diagrammes, le code étant généré automatiquement à partir de ceux-ci) (rôle joué par UML 2) ; ensuite, les applications informatiques distribuées (on ne parlera plus d'applications distribuées mais d'objets distribués, et non plus d'appels distants de procédures mais d'envoi de messages à travers le réseau) ; enfin, le stockage des données qui doit maintenant compter avec les objets. Chaque fois, plus qu'un changement de vocabulaire, un changement de mentalité sinon de culture s'impose.

Aujourd'hui, force est de constater que l'OO constitue un sujet d'une grande attractivité pour tous les acteurs de l'informatique. Microsoft a développé un nouveau langage informatique purement objet, C#, a très intensément contribué au développement d'un système d'informatique distribuée, basé sur des envois de messages d'ordinateur à ordinateur, les services web, et a plus récemment proposé un nouveau langage d'interrogation des objets, LINQ, qui s'interface naturellement avec le monde relationnel et le monde XML. Tous les langages informatiques intégrés dans sa nouvelle plate-forme de développement, Visual Studio .Net (aux dernières nouvelles, ils seraient 22), visent à une uniformisation (y compris les nouvelles versions de Visual Basic et Visual C++) en intégrant les mêmes briques de base de l'OO. Aboutissement considérable s'il en est, il devient très simple de faire communiquer ou hériter entre elles des classes écrites dans des langages différents. Quelques années auparavant, Sun avait créé Java,une création déterminante car à l'origine de ce nouvel engouement pour une manière de programmer qui pourtant existait depuis toujours sans que les informaticiens dans leur ensemble en reconaissent l'utilité et la pertinence. Depuis, en partant de son langage de prédilection, Sun à créé RMI, Jini, et sa propre version des services Web, tous basés sur les technologies OO. Ces mêmes services Web font l'objet de développements tout autant aboutis chez HP ou IBM. À la croisée de Java et du Web, originellement, la raison sinon du développement de Java du moins de son succès, on découvre une importante panoplie d'outils de développement et de conception de sites Web dynamiques.

IBM et Borland, en rachetant respectivement Rational et Together, mènent la danse en matière d'outil d'analyse du développement logiciel, avec la mise au point de puissants environnements UML, technologie OO comme il se doit. Au départ de développements chez IBM (qui soutient et parie sur Java plus encore que SUN ne le fait), la plate-forme logicielle Eclipse est sans doute, à ce jour, l'aventure Open Source la plus aboutie en matière d'OO. Comme environnement de développement Java, Eclipse est aujourd'hui le plus prisé et le plus usité et gagne son pari « d'éclipser » tous les autres. Borland a rendu Together intégrable tant dans Visual Studio.Net que dans Eclipse comme outil de modélisation UML synchronisant au mieux et au plus la programmation et la réalisation des diagrammes UML. Enfin, l'OMG, organisme de standardisation du monde logiciel, n'a pas comme lettre initiale de son acronyme la lettre O pour rien. UML et Corba sont ses premières productions : la version OO de l'analyse logicielle et la version OO de l'informatique distribuée. Cet organisme plaide de plus en plus pour un développement informatique détaché des langages de programmation ainsi que des plates-formes matérielles, par l'utilisation intensive des diagrammes UML. Au départ de ces mêmes diagrammes, les codes seraient générés automatiquement dans un langage choisi et en adéquation avec

la technologie voulue. Par le nouveau saut dans l'abstraction qu'il autorise, UML se profilerait comme le langage de programmation de demain. Il jouerait à ce titre le même rôle que jouèrent les langages de programmation au temps de leur apparition, en relèguant ceux-ci à la même place que le langage assembleur auquel ils se sont substitués jadis : un pur produit de traduction automatisée. Au même titre qu'Unix pour les développements en matière de système d'exploitation, l'OO donc apparaît comme le point d'orgue et de convergence de ce qui se fait de plus récent en matière de langages et d'outils de programmation.

Objectifs de l'ouvrage

Toute pratique économe, fiable et élégante de Java, C++, C#, Python, .Net ou UML requiert, pour débuter, une bonne maîtrise des mécanismes de base de l'OO. Et, pour y pourvoir, rien de mieux que d'expérimenter les technologies OO dans ces différentes versions, comme un bon conducteur qui se sera frotté à plusieurs types de véhicule, un bon skieur à plusieurs styles de ski et un guitariste à plusieurs modèles de guitare.

Plutôt qu'un voyage en profondeur dans l'un ou l'autre de ces multiples territoires, ce livre vous propose d'explorer plusieurs d'entre eux, mais en tentant à chaque fois de dévoiler ce qu'ils recèlent de commun. Car ce sont ces ressemblances qui constituent en dernier ressort les briques fondamentales de l'OO, matière de base, qui se devrait de perdurer encore de nombreuses années, y compris sous de nouveaux déguisements. Nous pensons que la mise en parallèle de C++, de Java, C#, Python, PHP 5 et UML est une voie privilégiée pour l'extraction de ces mécanismes de base.

Il nous a paru pour cette raison indispensable de discuter et comparer la façon dont ces cinq langages de programmation gèrent, par exemple, l'occupation mémoire par les objets ou leur manière d'implémenter le polymorphisme, pour en comprendre, *in fine*, toute la problématique et les subtilités indépendamment de l'une ou l'autre implémentation. Rajoutez une couche d'abstraction, ainsi que le permet UML, et cette compréhension ne pourra s'en trouver que renforcée. Chacun de ces cinq langages offrent des particularités amenant les praticiens de l'un ou l'autre à le prétendre mordicus supérieur aux autres : la puissance du C++, la compatibilité Windows et l'intégration XML de C#, l'anti-Microsoft et le leadership de Java en matière de développement Web, les vertus pédagogiques et l'aspect « scripting » de Python, le succès incontestable de PHP 5 pour la mise en place de solution Web dynamique et capable de s'interfacer aisément avec les bases de données. Nous nous désintéresserons ici complètement de ces guerres de religion (qui partagent avec les guerres de langages informatiques pas mal d'irrationalité), a fortiori car notre projet pédagogique nous conduit bien davantage à nous pencher sur ce qui les réunit plutôt que ce qui les différencie. C'est leur multiplicité qui a présidé à cet ouvrage et qui en fait, nous l'espérons, son originalité. Nous n'allons pas nous en plaindre et défendons en revanche l'idée que le choix définitif de l'un ou l'autre de ces langages dépend davantage d'habitude, d'environnement professionnel ou d'enseignement, de questions sociales et économiques et surtout de la raison concrète de cette utilisation (pédagogie, performance machine, adéquation Web ou base de données, …). De plus, le succès d'UML, assimilable à un langage universel OO à l'intersection de tous les autres et automatiquement traduisible dans chacun, ou des efforts, tels ceux de Microsoft, d'homogénéisation des langages OO, rend ces discordes quelque peu obsolètes et un peu dérisoires, tant il va devenir facile de passer de l'un à l'autre. Enfin, nous souhaitions que cet ouvrage, tout en étant suffisamment détaché de toutes technologies, couvre l'essentiel des problèmes posés par la mise en œuvre des objets en informatique, y compris le problème de leur stockage sur le disque dur et leur interfaçage avec les bases de données, de leur fonctionnement en parallèle, et leur communication à travers Internet. Un ouvrage donc qui découvrirait l'OO de très haut, ce qui lui permet évidemment de balayer très large, et qui accepte ce faisant de perdre un peu en précision, perte dont nous il apparaît nécessaire de mettre en garde le lecteur.

Plan de l'ouvrage

Les 23 chapitres de ce livre peuvent se répartir en cinq grandes parties.

Le premier chapitre constitue une partie en soi car il a pour importante mission d'introduire aux briques de base de la programmation orientée objet, sans aucun développement technique : une première esquisse, teintée de sciences cognitives, et toute en intuition, des éléments essentiels de la pratique OO.

La deuxième partie intègre les quatorze chapitres suivants. Il s'agit pour chacun d'entre eux de décrire, plus techniquement cette fois, ces briques de base que sont : objet, classe (chapitres **2** et **3**), messages et communication entre objets (chapitres **4**, **5** et **6**), encapsulation (chapitres **7** et **8**), gestion mémoire des objets (chapitre **9**), modélisation objet (chapitre **10**), héritage et polymorphisme (chapitres **11** et **12**), classe abstraite (chapitre **13**), clonage et comparaison d'objets (chapitre **14**), interface (chapitre **15**).

Chacune de ces briques est illustrée par des exemples en Java, C#, C++,Python, PHP 5 et UML. Nous y faisons le pari que cette mise en parallèle est la voie la plus naturelle pour la compréhension des mécanismes de base : extraction du concept par la multiplication des exemples.

La troisième partie reprend, dans le droit fil des ouvrages dédiés à l'un ou l'autre langage objet, des notions jugées plus avancées : les objets distribués, Corba, RMI, Services Web (chapitre **16**), le multithreading ou programmation parallèle (ou concurrentielle, chapitre **17**), la programmation événementielle (chapitre **18**) et enfin la sauvegarde des objets sur le disque dur, y compris l'interfaçage entre les objets et les bases de données relationnelles (chapitre **19**). Là encore, le lecteur se trouvera le plus souvent en présence de plusieurs versions dans les quatre langages de ces mécanismes.

La quatrième partie décrit plusieurs projets de programmation objet totalement aboutis, tant en UML qu'en Java. Elle inclut d'abord le chapitre **20**, décrivant la modélisation objet d'un petit flipper et les problèmes de conception orientée objet que cette modélisation pose. Le chapitre **21**, lié au chapitre 22, décrit la manière dont les objets peuvent s'organiser en liste liée ou en graphe, mode de mise en relation et de regroupement des objets que l'on retrouve abondamment dans toute l'informatique. Le chapitre **22** marie la chimie et la biologie à la programmation OO. Il contient tout d'abordla programmation d'un réacteur chimique générant de nouvelles molécules à partir de molécules de base, et ce, tout en suivant à la trace l'évolution de la concentration des molécules dans le temps. La chimie – une chimie élémentaire acquise bien avant l'université – nous est apparue être une plate-forme pédagogique idéale pour l'assimilation des concepts objets. Nous ne surprendrons personne en affirmant que les atomes et les molécules sont deux types de composants chimiques, et que les secondes sont composées des premiers. Dans ce chapitre, nous traduisons ces connaissances en UML et en Java. Dans la suite de la chimie, nous proposons aussi dans le chapitre une simulation élémentaire du système immunitaire, comme nouvelle illustration de combien l'informatique OO se prête facilement à la reproduction informatisée des concepts de science naturelle, tels ceux que l'on rencontre en chimie ou en biologie.

Enfin la dernière partie, se ramène au seul dernier chapitre, le chapitre 23, dans lequel sont présentés un ensemble de recettes de conception OO, solutionnant de manière fort élégante un ensemble de problèmes récurrents dans la réalisation de programme OO. Ces recettes de conception, dénommées Design Pattern, sont devenues fort célèbres dans la communauté OO. Leur compréhension accompagne une bonne maîtrise des principes OO et s'inscrit dans la suite logique de l'enseignement des briques et des mécanismes de base de l'OO. Elle fait souvent la différence entre l'apprenti et le compagnon parmi les programmeurs OO. Nous les illustrons en partie sur le flipper,la chimie et la biologie des chapitres précédents.

À qui s'adresse ce livre ?

Cet ouvrage ayant pour objet de traiter de nombreuses technologies, nul doute qu'il est destiné à être lu par un public assez large. En clair, il s'adresse à tous les adeptes de chacune de ces technologies, industriels, enseignants et étudiants, qui pourront le confronter utilement à l'état de l'art en la matière. La vocation première de cet ouvrage n'en reste pas moins une initiation à la programmation orientée objet, prérequis indispensable à l'assimilation de nombreuses autres technologies.

Ce livre sera un compagnon d'étude utile et, nous l'espérons, enrichissant pour les étudiants qui comptent la programmation objet dans leur cursus d'étude (et toutes technologies s'y rapportant : Java, C++, C#, Python, PHP, Corba, RMI, Services Web, UML). Il devrait les aider, le cas échéant, à évoluer de la programmation procédurale à la programmation objet, pour aller ensuite vers toutes les technologies s'y rapportant.

Nous ne pensons pas, en revanche, que ce livre peut seul prétendre à une même porte d'entrée dans le monde de la programmation tout court. Comme dit précédemment, nous pensons qu'il est idéal d'aborder les mécanismes OO en même temps que procéduraux. Pour des raisons évidentes de place et car les librairies informatiques déjà en regorgent, nous avons fait l'impasse d'un enseignement de base des mécanismes procéduraux : variables, boucles, instructions conditionnelles, éléments fondamentaux et compagnons indispensables à l'assimilation de l'OO. Nous pensons, dès lors, que ce livre sera plus facile à aborder pour des lecteurs ayant déjà un peu de pratique de la programmation dite procédurale, et ce, dans un quelconque langage de programmation. Aujourd'hui, l'informatique est un sujet si vaste, existant à tant de niveaux d'abstraction, et pour tant de raisons différentes, qu'il n'est pas étonnant qu'il faille l'aborder muni de plusieurs guides. Ce livre en est un. Il n'a rien d'exhaustif, ne se spécialise dans aucune des technologies évoquées, mais fournit les bases nécessaires à l'assimilation d'un grand nombre d'entre elles et de celles à venir.