

## La Programmation Sémantique Opérationnelle pour les Geeks

1.- L'industrie logicielle n'entend pas favoriser l'émergence des environnements de développement qui la dépossèderait du pouvoir de dispatcher de simples missions d'exécution à des armées d'ingénieurs qui ne sont, en fait, que des OS du core-coding (programmation de bas-niveau) où la division des tâches garantit l'illétrisme ontologique de l'analyste-programmeur, formaté, conditionné et parfaitement interchangeable de Paris à Delhi et de 4000 à 200 € / mois.

Ce que tout manager "Métier" d'un secteur d'activité quelconque autre que l'informatique obtiendra, dans ce contexte, d'une SSII ou d'un intégrateur informatique de stature internationale relèvera pour 99% de "l'assurance qualité d'un grand nom, leader sur son marché" (sic) et pour 1% de la qualité de service.

Quand les payeurs le comprendront, ils apprendront à recruter, en direct et sans intermédiaires, les architectes-développeurs de solutions informatiques de haute facture, capables de satisfaire, sans détours, l'ensemble de leurs besoins "Métier".

Ils cesseront, alors, de se laisser vendre ce dont on leur dit que c'est le mieux que l'on puisse obtenir, ce qui, pour être immédiatement vrai en termes de management de projets faiblement structurés, n'en devient pas moins faux, au regard de ce que les praticiens du développement sémantique opérationnel<sup>1</sup> sont, pour leur part, méthodologiquement et pratiquement en capacité de délivrer aux mêmes clients.

En prendre conscience, c'est comprendre que le développement (software) et la modélisation (hardware) informatique nous offrent, dès aujourd'hui, les moyens de produire des solutions de rendement croissant alors que l'industrie informatique privilégie un business anti-concurrentiel et spéculatif pour 99,9% de ce qu'elle distribue sur le marché.

2.- Le client, suffisamment curieux pour s'intéresser aux apports méthodologiques et pratiques du développement sémantique opérationnel<sup>1</sup>, finira invariablement par se rendre compte qu'il n'est rien qu'il ne puisse plus obtenir, aujourd'hui, à partir d'un cahier des charges "métier" solidement structuré :

- ▶ en payant moins de 25% de ce qu'on lui extorque aujourd'hui à coup de projets "industriels" pharaoniques,
- ▶ en cessant de payer des prestations informatiques dont les prix sont calqués sur les seuls modèles dont il a, dans ses propres activités (industrie, services, administrations) la culture : les rendements décroissants,
- ▶ en comprenant que le rendement, au final, authentiquement décroissant des offres proposées par les grands intégrateurs et SSII de l'industrie informatique permet à ces derniers de faire supporter aux payeurs le financement d'une "course à la compétitivité inverse" où moins on en donne et plus ça dure, de cycle en cycle, à l'infini, dans une danse à l'étouffement direct du client et indirecte

des concurrents de moindre surface (ex. : rachat de Sun, propriétaire de JAVA<sup>4</sup>, par Oracle).

Pour prendre ici une image en rapport avec l'actualité, Vettel, Webber et Newey (le concepteur de la RedBull Renault F1) ont fait 1 et 2 au GP de Sepang 2010 en produisant, tout simplement, la meilleure performance du plateau.

Si la course s'était, au contraire, déroulée en application d'un règlement conçu par la fédération internationale de l'automobile dans une logique de compétitivité inverse, le grand prix aurait été remporté par la voiture partie en tête et suffisamment large sur la piste pour qu'il soit matériellement impossible de la doubler sauf à le faire en empruntant les bas-côtés ou un hélicoptère pour jouer à saute-moutons...

3.1.- Sur le principe, la programmation sémantique opérationnelle<sup>1</sup> est une méthode de développement structurée pour produire, en termes de compétitivité directe, la meilleure performance de résultat possible, les modèles de programmation n'en respectant pas le paradigme produisant, eux, en parfaite bonne conscience, si ce n'est toujours délibérément, du vaporlock de compétitivité inverse.

Prenons un exemple fictif : l'exploitant d'un réseau ferroviaire veut changer de système de réservation et de vente en ligne de ses billets "grandes lignes". Il commande deux audits distincts à deux cabinets de conseil concurrents, le cahier des charges, identique dans les deux cas, étant fourni, présenté et explicité par le maître d'ouvrage aux deux maîtres d'oeuvre.

3.1.1.- Le cabinet A propose un pilote de réalisation de projet reposant sur un modèle standard (JAVA, PHP,...) de programmation orientée objet<sup>4</sup> :

3.1.1.1.- Il charge un architecte en systèmes d'information de décrire le progiciel attendu en utilisant les méthodes de modélisation les plus académiques (Merise<sup>4</sup>, UML<sup>4</sup>). L'architecte travaille, à ce stade, sur le papier et édite plusieurs diagrammes qui répertorient, structurent et organisent tous les états statiques possibles et interactions nécessaires à son fonctionnement. Sa copie ressemble, en pratique, à une série de puzzles dont les pièces sont des pictogrammes, des flèches et des échelles de temps.

3.1.1.2.- C'est à ce stade que le projet doit être vendu... Si le contrat est signé, l'architecte peut passer à la deuxième et dernière étape de sa mission : découper le modèle papier de son application en presque autant de sous-composants programmables qu'il y a de pièces dans le puzzle de sa copie. L'unité systémique et fonctionnelle du programme initialement pensé dans son ensemble s'efface au profit d'un "éclaté de particules de programme" semblables à des "neurones numériques" (appelés classes en programmation orientée objet<sup>4</sup>) dont chacun aura à produire un calcul très basique, voire simpliste, chaque composant étant relié à ses collègues par des "synapses" (transmission des calculs d'instanciation

de classe, chacune appelant ce qu'elle a besoin d'obtenir de ses voisines pour produire le résultat attendu tel qu'appelé par une instance de classe plus lointaine), l'ensemble étant appelé à constituer, en un énorme "plat de spaghetti" pauvrement structuré, le progiciel final.

3.1.1.3.- L'architecte en système d'information transmet alors le relais à un chef de projet. Ce dernier distribue alors le travail, le plus souvent en offshore, entre des ingénieurs lambdas codant au kilomètre les différentes classes, sans que chacun n'ait jamais à en savoir plus que ce qui sera utile au fonctionnement de sa propre contribution.

3.1.1.4.- Le travail des ingénieurs lambdas est alors inspecté et testé avec navette de retour à l'envoyeur en cas de dysfonctionnement constaté par des ingénieurs de second niveau (ici, rarement recrutés en offshore, quoique...), le chef de projet ayant pour mission de chapeauter ses troupes locales et offshore pour que la mission se termine dans les délais contractualisés (ni avant, ni après !)...

3.1.2.- Le cabinet B propose, pour sa part, un pilote de réalisation de projet reposant sur un modèle de programmation sémantique opérationnelle.

3.1.2.1.- Il charge un architecte-développeur en systèmes d'information de prototyper le progiciel attendu en le lui demandant explicitement de coder la pré-version alpha du logiciel attendu. Cet architecte-développeur travaille en utilisant, dès le départ, le langage de programmation qui permettra de coder la version finale du produit (ou un langage-maître + un ou deux langages accessoires dédiés à des tâches ponctuelles et spécialisées : interaction avec une base de données, impression, envoi de mails, etc...). éditant les grandes lignes des procédures qui répertorient, structurent et organisent tous les états statiques possibles et interactions nécessaires à son fonctionnement.

Sa copie ressemble, à ce stade, à ce que deviendra le programme final, les principales fonctions nécessaires à la mise en place de l'interface utilisateur étant, dès le départ, implémentées pour aider le client de se faire une idée aussi précise que possible de ce que sera le produit fini.

3.1.2.2.- C'est à ce stade que le projet doit être vendu.... Si le contrat est signé, l'architecte-développeur peut passer à la deuxième et dernière étape de sa mission : terminer le codage de l'application à livrer, seul si le dossier le permet et assisté, sinon, par quelques développeurs dont il coordonnerait alors le travail jusqu'à la fin de la mission permettant de terminer, sans étape supplémentaire, la version définitive du progiciel.

3.2.- Deux stratégies de communication avant-vente : la sécurité du discours théorique formel du cabinet A et la présentation d'un prototype en forme de "proof of concept" par le cabinet B

Toute l'intelligence ajoutée propre aux offres des deux cabinets A et B concurrents relève, à ce stade, de l'activité propre aux alinéas 3.1.1.- et 3.1.2.-

de chacune des propositions. Les deux architectes produisent, à ce stade le meilleur effort possible, en termes de compétitivité directe.

Pour parvenir à produire son prototype dans le même temps limité que celui que l'architecte de l'équipe A consacre à éditer de simples diagrammes de modélisation, l'architecte de l'équipe B doit s'appuyer sur une méthodologie de double formalisation - sémantique et opérationnelle<sup>1</sup> - des processus qu'il doit prendre en charge. Il sera ici essentiel qu'il choisisse des outils de développement bien adaptés à cette exigeante gymnastique, explicitement conçus pour simplifier le caractère protéiforme de sa réflexion :

- ▶ centralisation de la logique "métier" du futur progiciel donnant au programme la structure d'un serveur de données, au choix, accessible en mode local ou distant.
- ▶ association de cette structure "serveur" avec une logique de traitement obéissant au paradigme élémentaire du système-expert<sup>5</sup> (données + règles impératives = résultats calculés).
- ▶ modelage rédactionnel (code source) de la logique "métier" par codage direct des principales procédures organisant la logique fonctionnelle du progiciel
- ▶ utilisation d'un langage de programmation dynamiquement typé<sup>3</sup> permettant à l'architecte de rester concentré sur sa réflexion logique sans avoir à s'interrompre à chaque instant pour déclarer le type de chacune des valeurs (constantes d'initialisation des traitements à opérer, variables de changement d'état des données manipulées au fil des-dits traitements) qu'il manipule.
- ▶ utilisation d'un langage interprété permettant de tester instantanément chaque nouvelle procédure sans avoir à se préoccuper de la compiler systématiquement au préalable.

En résumé, c'est à la rencontre de l'intelligence artificielle (datas, règles, moteur d'inférence), des langages fonctionnels et des langages de programmation logique<sup>3</sup> (Lisp, SmallTalk, Python, Rexx, Runtime Revolution,...) que tout se joue et permet au cabinet B de proposer un modèle de développement directement centré sur la sémantique opérationnelle<sup>3</sup> des processus à mettre en oeuvre et, où tout est fait pour favoriser l'économie de moyens, le contrôle de qualité et la performance de résultat.

3.3.- Contrat en poche, les stratégies des cabinets A et B prennent des chemins opposés...

Pour le cabinet A, le démembrement du modèle unifié de départ ouvre la voie à la taylorisation du dossier (modèle industriel, systématiquement contre-productif en matière de prestation intellectuelle)...

Pour le cabinet B, la recherche de la performance optimale se poursuit et les meilleures méthodes de développement rapide (XP Programming, SCRUM,...) renforcent rapidement la lisibilité de l'état d'avancement du projet.

3.4.- La méthode du développement sémantique opérationnel n'a pas jusqu'ici donné lieu à une mise en forme théorique permettant aux maîtres d'ouvrages de

s'interroger utilement sur son bien-fondé. Elle reste, à ce jour, une pratique de terrain connue et mise en oeuvre par trois catégories d'acteurs :

- ▶ Les éditeurs de langages fonctionnels et de programmation logique<sup>3</sup>, prioritairement dédiés au développement rapide universel et spécialisé et où l'on a pu voir se succéder des offres toujours plus performantes au fil du temps (Cobol, Prolog, Rexx, Lisp, Metacard, Perl, Python, Runtime Revolution,...).
- ▶ Les maîtres d'ouvrages qui comprennent ne pas pouvoir obtenir de solution informatique adaptée à leurs attentes en programmation impérative<sup>2</sup> ou orientée objet<sup>4</sup> (domaines de l'intelligence artificielle, de l'aide à la décision adaptative, du reporting adaptatif,... ; secteurs de l'industrie aérospatiale, de l'automobile, de l'éducation, des ressources humaines, de l'ingénierie financière,...).
- ▶ les architectes-développeurs sollicités par les précédents pour agir comme maîtres d'oeuvre.
- ▶ Les concepteurs de solutions logicielles inédites qui deviendraient trop vite des usines à gaz difficiles à finaliser et à industrialiser en programmation non sémantique.

La programmation sémantique opérationnelle reste, surtout et encore, une pratique suspecte pour les intégrateurs et les SSII qui ont appris à se nourrir grassement des stratégies de compétition inverse. Quand certains prétendent rien en connaître ou, même, qu'elle participerait d'une vision erronée des choses, d'une forme d'informatique "molle" qu'ils croient pouvoir opposer à une noblesse présumée des "sciences dures" où l'informatique pourrait apparaître comme une discipline scientifique, il conviendrait de leur rappeler que l'informatique n'est en rien une science mais un environnement continuellement évolutif et nourri de techniques et de modèles trouvant avantage à entretenir une grande complicité avec de nombreux domaines scientifiques, le plus souvent transdisciplinaires mais pas seulement. Il peut aussi y avoir de l'Art dans la manière de designer un programme informatique...

Least but not last : Oracle a publié de 1989 à 1994 l'un des meilleurs langages de programmation sémantique opérationnelle du marché (Oracle Media Object). Il serait certainement édifiant d'entendre les explications "OFF" qui ont amené Larry Ellison à se laisser convaincre que ce produit devait disparaître de son catalogue et on n'ose imaginer ce que serait, aujourd'hui, le coeur de l'ingénierie informatique mondiale s'il avait, plutôt, fait le choix stratégique d'utiliser Oracle Media Object pour écraser le "toujours vagissant Java" de Sun (qu'il a de toute façon avalé depuis).

Une remarque encore : les principaux langages de programmation impérative<sup>2</sup> et leurs dérivés orientés POO<sup>4</sup> sont si peu souples "dans les coins" que tous les grands serveurs de bases de données du marché (Oracle : PL/SQL, PostgreSQL : PL/pgSQL, SQLServer et Sybase : Transact-SQL, IBM DB2 : SQL PL, etc...) comportent, en interne, leur propre langage de programmation orienté sémantique opérationnelle.

4.1.- Avant de poursuivre, précisons, pour éviter tout malentendu, le cadre de référence de la réflexion engagée :

Il n'est aucunement question, ici, de dénier le rôle et la place qui est et restera celle des principaux langages impératifs<sup>2</sup> compilés, actuels et futurs, dans le monde du développement logiciel. En ce sens, il est essentiel de rappeler que la plupart des logiciels serveurs (démons web, bases de données SQL multi-utilisateurs, serveurs de streaming audio-video, démons mail, FTP, etc,... sont habituellement codés en C, C++ ou Objective C, pour ne citer que les langages impératifs<sup>2</sup> compilés les plus répandus. Il en va de même pour toutes les applications de bureautique traditionnelle, de traitement d'image, d'encodage audio-video, de jeux d'arcade, etc... où la vitesse d'exécution du code conditionne, de manière significative le confort d'utilisation, la performance et l'économie en moyens de calcul. Dès lors que la vitesse d'exécution redevient essentielle, quel que soit la qualité du prototypage initial, le logiciel devra être compilé avant utilisation.

Ceci étant dit, rappelons encore que :

- ▶ nous sommes passé, en moins de trois générations de l'ABC (Atanasoff Berry Computer) qui stockait 60 mots de 50 bits dans ses deux tambours et réalisait 30 additions par seconde (1939) au Jaguar XT5-HE (Cray, 2009), le premier supercalculateur à avoir officiellement dépassé la puissance de 1.75 pétaFLOPS.
- ▶ la démocratisation de l'internet haut-débit, la puissance de calcul disponible à très bas prix, la virtualisation des infrastructures (serveurs, data centers, cloud-computing) conditionnent une migration presque globale et généralisée des données et des applications qui quittent le poste de travail pour se déporter vers des serveurs capables de les sécuriser et de les partager entre des milliers, voire, des millions d'utilisateurs.

Sans s'étendre sur les détails, tout le monde y gagne : de la chute des coûts qui naît de la mutualisation des ressources partagées jusqu'à la maintenance automatisée des services en ligne en passant par la redondance des infrastructures de calcul et de stockage, les centres de coût de l'informatique professionnelle et personnelle quittent le poste de travail qui devient, hors bureautique de première nécessité, un simple terminal de connexion aux ressources distantes, accessibles en mode loué (services aux professionnels) ou en contre-valeur de la publicité en ligne (services aux particuliers).

Cette tendance ne présente qu'un inconvénient qu'il convient de noter avant de poursuivre : quid, pour l'entreprise française E., de la confidentialité de ses données "Métier" hébergées dans un data-center A. en Asie-Pacifique ou aux USA ? Attention, ici, à ne pas se laisser aveugler par le fait que la sécurité matérielle des données conditionnerait aussi leur inviolabilité... Si l'entreprise E. ne veille pas, très attentivement, à la protection de son savoir-faire, elle en paiera, tôt ou tard, invariablement le prix.

4.2.- Hors bureautique de base (word, excel, powerpoint), 90% des besoins informatiques "Métier" de l'entreprise communicante font appel à des solutions partagées en réseau (internet, intranet, extranet, réseau virtuel privé over IP). Or, la structure même des échanges de données en réseau (sockets UDP et TCP) fait que la latence des communications et le délai d'acheminement des données échangées (quelques millisecondes à quelques dixième de secondes au plus) prend plus de temps que celui qui est nécessaire à la mise en forme des données transmises du poste client vers le serveur et réciproquement.

En d'autres termes, la programmation des applications réseau de l'entreprise peut, aujourd'hui, faire appel à des langages dont le code est compilé à la volée au moment de son exécution, sans impacter négativement les performances. Ce constat est valable pour tous les langages utilisés pour coder ces applications réseau (Perl, PHP, Python, JAVA, Javascript, etc...). Aucun obstacle ne s'oppose donc à ce que les langages interprétés les mieux adaptés à la programmation sémantique opérationnelle<sup>1</sup> soient préférés à ce qui le sont moins, chaque fois que l'on peut partir d'une feuille blanche pour designer une nouvelle application réseau.

Une bonne application réseau doit réunir deux caractéristiques essentielles : elle doit être structurellement fiable et pouvoir évoluer à l'infini !

4.2.1.- Une bonne application réseau doit être structurellement fiable (y compris, parfaitement résistante aux tentatives d'intrusion malveillantes) :

Designer une application réseau structurellement fiable consiste à lui interdire de prendre en compte tous les traitements non explicitement autorisés (possibles effets de bord ou tentatives d'intrusion malveillantes) :

4.2.1.1.- En termes de logique "Métier",

ce qu'il faut définir avant de commencer le codage, c'est un sommaire des traitements à prendre en charge, permettant d'organiser la logique applicative du centre vers la périphérie, en étoile, où un authentique moteur d'inférence représentera le cœur du système d'information, le gendarme organisant la circulation des messages, tandis que les données d'entrée, les règles impératives applicables et les résultats calculés constitueront, pour leur part, le carburant, le comburant, et les résultats d'exploitation. Cette structure modulaire propre à la programmation sémantique opérationnelle permet d'assurer une cohérence systémique forte des programmes en cours de développement, quand bien même l'ensemble des modules périphériques de gestion des traitements ne seraient pas intégralement qualifiés dès le départ.

Cette approche permet, mieux que beaucoup d'autres, de limiter le temps nécessaire à la vérification du code fraîchement édité, seules les opérations explicitement listées par les règles impératives<sup>5</sup> utilisables par le moteur d'inférence<sup>5</sup> ayant à être vérifiées une à une, toute interaction autre, susceptible de parasiter la logique applicative principale, ne disposant, à contrario, d'aucun moyen d'accès propre à lui permettre d'interagir avec le code validé.

4.2.1.2.- En termes de modélisation technique,

la transposition du principe énoncé à l'alinéa précédent du "tout ce qui n'est pas explicitement autorisé est, de fait, interdit." doit aussi s'appliquer à la structure technique gouvernant les transactions réseau de l'application distribuée<sup>6</sup>. Il suffit, ici, de s'en tenir à une règle simple trop souvent négligée : l'utilisation d'un protocole de contrôle automatique de l'identité du client<sup>6,7</sup> et de l'intégrité de la requête qu'il adresse au serveur<sup>6</sup> chargé de la traiter, ce dernier ayant ordre de la tenir pour nulle et non avenue si le contrôle d'authentification échoue. Cette précaution permet, dans 98% des cas, d'éviter, tant l'usage de sessions cryptées qu'une trop grande redondance des opérations d'authentification forte, toutes deux fortement consommatrices de ressources, l'usage d'un simple message HTTP<sup>7</sup> "POST" en lieu et place du trop commun "GET" étant généralement suffisant, très discret (et donc difficilement repérable par les pirates) et parfaitement neutre en termes de trafic réseau.

4.2.2.- Une bonne application réseau<sup>6</sup> doit être conçue de manière à pouvoir évoluer continuellement en souplesse.

A cet effet, il est essentiel, qu'elle soit, dès le départ, architecturée en un système logiciel structurellement modulaire. Il ne suffit plus ici de dire que la programmation sémantique opérationnelle se prête idéalement à ce type d'organisation pour en déduire que les méthodes qui lui sont propres y suffisent.

Avant que le codage du progiciel ne puisse débuter, il faudra encore que la distribution fonctionnelle décrite à l'alinéa 4.2.1.1. trouve son pendant en termes de distribution technique des traitements associés. Il importe ici de préciser qu'une parfaite indépendance des différents gestionnaires de données (serveur d'application des règles d'inférence, stockage ACID-SQL centralisé des données, publication de l'interface utilisateur sur le poste client, affichage synchrone/asynchrone des résultats calculés dans l'interface utilisateur) contribuera, d'une manière essentielle, à la cohérence systémique du produit fini.

4.3.- En résumé, les performances de résultat de la programmation sémantique opérationnelle<sup>1</sup> reposent entièrement sur la mise en oeuvre d'une architecture de démembrement de la complexité en unités fonctionnelles (et techniques) modulaires administrées comme des périphériques indépendants et complémentaires sous le contrôle d'un moteur d'inférence<sup>5</sup> en charge de distribuer et de coordonner les rôles élémentaires qui conditionnent la performance globale du progiciel.

Un parfait respect des règles de modularité du code garantit qu'il sera, à l'avenir, toujours simple et évident de faire évoluer les caractéristiques du logiciel, de manière à pouvoir lui adjoindre de nouvelles options, voire, en modifier ou en supprimer d'autres sans avoir à repartir tous les cinq ans d'une feuille blanche, là où c'est une norme habituelle en matière de programmation non sémantique.

4.4.- Quelques situations concrètes où la programmation sémantique opérationnelle nous permet de déployer de meilleurs logiciels réseau, de meilleurs sites professionnels, un meilleur Internet

- ▶ tous types d'applications d'entreprise louées,
- ▶ toutes solutions personnalisées et adaptatives d'aide à la décision (reparamétrables par les opérateurs),
- ▶ tous systèmes de gestion de contenus fortement évolutifs (e-learning, jeux en ligne, salles de marchés,...),
- ▶ tous systèmes d'enquêtes en ligne et/ou de reporting administratif et financier à autorité,
- ▶ tous web-services personnalisés dédiés à l'interconnexion dynamique de solutions réseau non conçues pour l'être,
- ▶ tous systèmes de veille concurrentielle personnalisés,
- ▶ tous systèmes d'authentification forte deux-niveaux compatibles HTTP(S)

Alors, pour conclure, les concepts méthodologiques qui définissent la programmation sémantique opérationnelle nous apportent-ils une assurance de résultat au format "AAA" propre à nous permettre de développer de meilleures solutions réseau d'entreprise ? Certainement, pour autant que nous prenions conscience que les méthodes et les outils adéquats sont disponibles et n'attendent de nous qu'une seule chose : que nous ne perdions jamais de vue les quelques fondamentaux permettant d'ancrer le service informatique à forte valeur ajoutée dans une logique de compétitivité authentiquement dédiée à une distribution "axelrodienne", plus équitable en somme,... des rendements croissants entre fournisseurs et clients.

Le développement sémantique opérationnel pourrait bien, dès lors, y gagner ses "trois étoiles".

Qu'y gagneront, de leur côté, en termes de qualité perçue, les utilisateurs ?

- ▶ des systèmes d'information fiables et sécurisés qui feront, simplement et durablement, ce que l'on attend d'eux,
- ▶ des systèmes d'information faciles à faire évoluer dans le temps sans risque de dérapage des coûts.
- ▶ des systèmes d'information bien adaptés à la prise en charge des normes (RDF, OWL, SPARQL, SVG,...) énoncées par le W3C pour favoriser l'émergence du web sémantique<sup>8</sup>.

Pierre-Clément Sahores, à Jurançon le 30 avril 2010

## Notes

1.- En informatique, la sémantique opérationnelle est l'une des approches qui servent à donner une signification aux programmes informatiques d'une manière rigoureuse, mathématiquement parlant (voir Sémantique des langages de programmation).

Une sémantique opérationnelle d'un langage de programmation particulier décrit comment chaque programme valide du langage doit être interprété en termes de suite d'états successifs de la machine. Cette suite est la signification du programme. Dans le cas d'un programme fonctionnel, l'état final d'une suite qui termine donne la valeur de retour du programme. (Dans le cas général, il peut y avoir plusieurs suites de calculs et plusieurs valeurs de retour pour un seul programme, parce que celui-ci pourrait être non-déterministe.)

Un des moyens les plus courants pour définir rigoureusement une sémantique opérationnelle est de fournir un système de transition d'états rendant compte du comportement attendu du langage considéré. Une telle définition autorise une analyse formelle du langage, permettant l'étude de relations entre les programmes. Parmi les relations importantes, on trouve : les pré-ordres de simulation et les bisimulations, qui sont très utiles dans le cadre du parallélisme.

Définir une sémantique opérationnelle au travers d'un système de transition se fait habituellement en donnant une définition inductive de l'ensemble des transitions possibles. Habituellement, cela prend la forme d'un ensemble de règles d'inférence définissant les transitions valides du système.

La sémantique opérationnelle est reliée à la sémantique dénotationnelle au travers du concept d'abstraction.

source : [http://fr.wikipedia.org/wiki/Sémantique\\_opérationnelle](http://fr.wikipedia.org/wiki/Sémantique_opérationnelle)

2.- En informatique, la programmation impérative est un paradigme de programmation qui décrit les opérations en termes de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme.

L'implémentation de la quasi totalité des processeurs qui équipent les ordinateurs est de nature impérative : ils sont faits pour exécuter du code écrit sous forme d'opcodes (pour operation codes), qui sont des instructions élémentaires exécutables par le processeur. L'ensemble des opcodes forme le langage machine spécifique au processeur et à son architecture. L'état du programme à un instant donné est défini par le contenu de la mémoire centrale à cet instant, et le programme lui-même est écrit en style impératif en langage machine, ou le plus souvent dans une traduction lisible par les humains du langage machine, dénommée assembleur.

Les langages de plus haut niveau utilisent des variables et des opérations plus complexes, mais suivent le même paradigme. Les recettes de cuisine et les vérifications de processus industriel sont deux exemples de concepts familiers qui s'apparentent à de la programmation impérative ; de ce point de vue, chaque étape est une instruction, et le monde physique constitue l'état modifiable. Puisque les idées basiques de la programmation impérative sont à la fois conceptuellement familières et directement intégrées dans l'architecture des microprocesseurs, la grande majorité des langages de programmation sont impératifs.

source : [http://fr.wikipedia.org/wiki/Programmation\\_impérative](http://fr.wikipedia.org/wiki/Programmation_impérative)

3.- Les langages de programmation impératifs doivent être distingués d'autres types de langages, les langages fonctionnels et les langages de programmation logique. Les langages fonctionnels, tels que Haskell ou ML, ne sont pas des suites d'instructions et ne s'appuient pas sur l'idée d'état global, mais au contraire tendent à s'extraire de ce modèle pour se placer à un niveau plus conceptuel (qui a ses fondations dans le lambda-calcul). Les langages de programmation logiques, tels que Prolog, se concentrent sur ce qui doit être calculé, et non comment le calcul doit être effectué.

source : [http://fr.wikipedia.org/wiki/Programmation\\_impérative](http://fr.wikipedia.org/wiki/Programmation_impérative)

4.- La programmation orientée objet (POO), ou programmation par objet, a été élaborée par Alan Kay dans les années 1970. C'est un paradigme de programmation informatique qui consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. Il possède une structure interne et un comportement, et il sait communiquer avec ses pairs. Il s'agit donc de représenter ces objets et leurs relations ; la communication entre les objets via leur relation permet de réaliser les fonctionnalités attendues, de résoudre le ou les problèmes.

Il est possible de concevoir par objet une application informatique sans pour autant utiliser des outils dédiés. Il n'en demeure pas moins que ces derniers facilitent de beaucoup la conception, la maintenance, et la productivité. On en distingue plusieurs sortes :

- les langages de programmation (Java, C#, Vala, Objective C, Eiffel, Python, Ruby, C++, PHP, Smalltalk...)
- les outils de modélisation qui permettent de concevoir sous forme de schémas semi-formels la structure d'un programme (Objectteering, UMLDraw, Rhapsody, DBDesigner...)
- les bus distribués (DCOM, CORBA, RMI, Pyro...)
- les ateliers de génie logiciels ou AGL (WinDev et son langage le WLangage)

Il existe actuellement deux catégories de langages à objets : les langages à classes et ceux à prototypes, que ceux-ci soient sous forme fonctionnelle (CLOS), impérative (C++, Java...) ou les deux (Python, OCaml, ...).

source : [http://fr.wikipedia.org/wiki/Programmation\\_orientée\\_objet](http://fr.wikipedia.org/wiki/Programmation_orientée_objet)

5.- Un système expert est un logiciel capable de répondre à des questions, en effectuant un raisonnement à partir de faits et de règles connus. Il peut servir notamment comme outil d'aide à la décision. Le premier système expert est DENDRAL. Il permettait d'identifier les constituants chimiques.

Un système expert se compose de 3 parties :

- une base de faits,
- une base de règles et
- un moteur d'inférence.

Le moteur d'inférence est capable d'utiliser faits et règles pour produire de nouveaux faits, jusqu'à parvenir à la réponse à la question experte posée.

La plupart des systèmes experts existants reposent sur des mécanismes de logique formelle (logique aristotélicienne) et utilisent le raisonnement déductif.

source : [http://fr.wikipedia.org/wiki/Système\\_expert](http://fr.wikipedia.org/wiki/Système_expert)

6.- L'architecture d'un environnement informatique ou d'un réseau est dite distribuée quand toutes les ressources ne se trouvent pas au même endroit ou sur la même machine. On parle également d'informatique distribuée. Ce concept s'oppose à celui d'architecture centralisée dont une version est l'architecture client-serveur.

Internet est un exemple de réseau distribué puisqu'il ne possède aucun nœud central. Les architectures distribuées reposent sur la possibilité d'utiliser des objets qui s'exécutent sur des machines réparties sur le réseau et communiquent par messages au travers du réseau.

source : [http://fr.wikipedia.org/wiki/Architecture\\_distribuée](http://fr.wikipedia.org/wiki/Architecture_distribuée)

7.- Le HyperText Transfer Protocol, plus connu sous l'abréviation HTTP, littéralement le « protocole de transfert hypertexte », est un protocole de communication client-serveur développé pour le World Wide Web. HTTPS (avec S pour secured, soit « sécurisé ») est la variante du HTTP sécurisée par l'usage des protocoles SSL ou TLS.

Les clients HTTP les plus connus sont les navigateurs Web permettant à un utilisateur d'accéder à un serveur contenant les données. Il existe aussi des systèmes pour récupérer automatiquement le contenu d'un site tel que les aspirateurs de site Web ou les robots d'indexation.

Ces clients se connectent à des serveurs HTTP tels qu'Apache HTTP Server ou Internet Information Services.

source : [http://fr.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](http://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

8.- Le Web sémantique désigne un ensemble de technologies visant à rendre le contenu des ressources du World Wide Web accessible et utilisable par les programmes et agents logiciels, grâce à un système de métadonnées formelles, utilisant notamment la famille de langages développés par le W3C.

source : [http://fr.wikipedia.org/wiki/Web\\_sémantique](http://fr.wikipedia.org/wiki/Web_sémantique)