ARCHITECTURE LOGICIELLE

Concevoir des applications simples, sûres et adaptables

Jacques Printz
Professeur émérite du Cnam, chaire de génie logiciel

Préface de Yves Caseau

3e édition

DUNOD

Toutes les marques citées dans cet ouvrage sont des marques déposées par leurs propriétaires respectifs.

Illustration de couverture : Paris © Beboy-Fotolia.com

ISBN 978-2-10-058342-3

Préface

Ce livre parle, de façon centrale, de l'architecture des logiciels et des systèmes d'information, sous toutes ses facettes. L'architecture logicielle est un sujet scientifique et académique, qui a une histoire, des spécialistes et des références. Il existe d'excellents livres, cités au demeurant par Jacques Printz, et c'est une spécialité qui est enseignée dans la plupart des formations informatiques. L'architecture du système d'information est un sujet plus flou, dont l'importance pratique est évidente, mais dont les fondements sont moins théorisés. Les documents de référence ne sont plus des articles ou des livres, ce sont le plus souvent des schémas. Pour caricaturer, le dessin remplace l'équation et l'expérience remplace le raisonnement. Le lecteur qui s'aventure dans une entreprise découvre des merveilles d'ingéniosité, mais aussi une tradition orale, beaucoup de re-découvertes et de ré-inventions, et une difficulté à capitaliser et à faire émerger une véritable discipline scientifique. Même s'il existe des cours, dans les bons établissements, ou des livres, ils n'ont ni la maturité ni la rigueur de ce qui est consacré à l'architecture logicielle.

Cet état de fait n'est pas un hasard, ni une simple conséquence historique (on a commencé à faire des logiciels avant de faire des systèmes d'information). L'architecture du système d'information est un sujet « en tension ». Cette tension s'exprime entre la vision globale et les propriétés locales, entre les dimensions techniques et les dimensions métiers (ce qui est superbement illustré dans ce livre), entre une échelle de temps courte pour comprendre le fonctionnement et une échelle beaucoup plus longue pour comprendre l'écosystème des acteurs du système d'information. Comprendre l'architecture du système d'information est une gageure, il faut des compétences multiples, une vaste culture scientifique mais également une véritable expérience métier et industrielle. C'est un chemin semé d'embûches, qui est forcément long et demande, le lecteur va s'en apercevoir, une certaine patience.

Il se trouve que Jacques Printz est le guide idéal pour nous conduire sur ce chemin. Il combine une expérience professionnelle et industrielle remarquable et un talent d'enseignant dont témoignent ses précédents ouvrages et les cours qu'il propose au Cnam. Sa culture scientifique est exceptionnelle, et couvre un spectre très large. Le lecteur va s'apercevoir qu'il ne s'agit pas d'agrémenter son exposé par quel-

ques références brillantes, mais bien de donner du sens et de la vie à des principes qui pourraient, autrement, sembler abstraits, voire arbitraires. Jacques Printz a un regard unique sur quarante ans d'histoire de l'informatique et des systèmes d'information, dans lequel la pratique et la théorie sont mêlées de façon indissociable.

Ce livre propose au lecteur une promenade érudite et passionnante. Je parle de promenade car la lecture en continu sera indigeste, sauf pour des lecteurs experts. Ce livre est d'une très grande richesse, il s'agit clairement d'une « somme », issue d'une grande expérience et d'un long travail. Le terme de promenade fait également référence au plaisir qu'il y a de parcourir, de découvrir et de comprendre, de se former un jugement « esthétique » sur les choses. Le lecteur trouvera de nombreuses références à l'élégance et à l'esthétique des constructions.

Il ne s'agit pas, en revanche, d'un ouvrage théorique. Ce livre contient des données chiffrées, de nombreux exemples, des références aux différentes expériences pratiques de son auteur. Jacques Printz connaît bien le monde de l'entreprise, ce qui est essentiel pour bien comprendre le système d'information. La construction de ce système est une aventure humaine, avec de nombreux acteurs et de nombreux rôles. Cette répartition des pouvoirs et des responsabilités est expliquée de façon claire et intuitive, avec l'aide de nombreux schémas.

Il s'agit clairement d'un livre de référence, de ceux qu'on a plaisir à posséder dans sa bibliothèque pour pouvoir trouver la réponse à telle ou telle question. Il va au fond des choses, et possède une bibliographie complète, commentée et très pertinente. C'est la première porte, le lecteur est invité à en pousser d'autres (une évidence compte tenu de l'immensité du sujet). Ce livre est une « bible » de l'histoire des idées en informatique, ce qui en fait un ouvrage fondamental pour les enseignants et les chercheurs.

Il serait vain, en conséquence, de vouloir aborder les « principaux thèmes » de ce livre. Pour n'en citer que quelques-uns, je vais m'arrêter sur ceux qui me semblent les plus importants et qui sont encore trop ignorés dans les formations sur le système d'information. Par exemple, ce livre fait la part belle au thème de l'architecture de données, pris sous une forme extrêmement vaste, qui va de la modélisation jusqu'aux transactions distribuées. Les explications fournies par Jacques Printz sont tout à la fois claires et détaillées, et font de ce livre une contribution significative au domaine. Il est rare, en effet, de voir les sujets de distribution de données traités de « bout en bout », depuis le point de vue sémantique et conceptuel jusqu'à l'architecture interne du moniteur transactionnel.

De la même façon, la troisième partie sur l'architecture fonctionnelle propose une « reconstruction complète du génie logiciel » en partant de ses finalités (dans le cadre du système d'information), c'est-à-dire les processus de l'entreprise. Le lecteur qui suit ce chemin va redécouvrir, dans une vision globale et cohérente, tous les principes de l'urbanisation du système d'information. Cette partie permet d'ailleurs de souligner la continuité remarquable entre la vision métier et la vision technique qui est présentée dans ce livre.

Une autre contribution formidable de ce livre est la vulgarisation des « patrons de conceptions » du système d'information. C'est la bonne réponse à cette tendance néfaste à la réinvention et la répétition des mêmes erreurs qui trahit parfois l'immaturité de la discipline. Jacques Printz n'est pas le premier à faire émerger des « *patterns* » de l'architecture du système d'information, mais sa présentation est lumineuse. On y trouvera, en particulier, les descriptions « intelligentes » des concepts tels que le client-serveur, l'architecture en couches (ou en tiers) ou l'utilisation de bus logiciels. L'intelligence ici est ce qui permet de simplifier, de limiter la complexité, de faire les généralisations qui s'imposent, ou de replacer l'évolution des idées dans un contexte historique. La maîtrise de la complexité est probablement le sujet central de l'architecture du système d'information, et il reçoit cette place dans le livre de Jacques Printz. Il est l'occasion, une fois de plus, de profiter de l'érudition de l'auteur et de replacer certaines questions informatiques dans un cadre philosophique plus large.

Pour conclure, je noterai que ce livre contient l'essentiel de ce qu'il faut savoir pour comprendre le fonctionnement d'un système d'information complexe d'une grande entreprise, ce qui en fait un ouvrage unique. En tant que DSI, je pourrais ajouter, à titre de clin d'œil, que c'est le livre que je souhaiterais que tous mes fournisseurs de logiciel aient lu (et compris). Il s'agit sans nul doute d'un livre qui comble un vide, qui va jouer un rôle de référence pour de nombreuses années. Il fait partie de ces livres qu'on regrette de ne pas avoir lu plus tôt lorsqu'on les referme.

Yves CASEAU
Directeur général adjoint de Bouygues Telecom,
Membre de l'Académie des Technologies

Table des matières

Partie 2 – Analyse de deux chefs-d'œuvre d'architecture

Partie 3 – Architecture fonctionnelle logique

Partie 4 – Propriétés d'une bonne architecture

Avant-propos

Jusqu'à une date récente l'architecture logicielle était une notion qui ne dépassait pas le cercle très restreint des concepteurs de systèmes d'exploitation, de progiciels système comme des SGBD (Système de gestion de base de données), ou des logiciels de télécommunication ou celui des concepteurs de systèmes temps réel pour la défense, le spatial ou le contrôle aérien. C'est ce domaine technologique qui a donné naissance, dès les années 1950-1960 à l'ingénierie système[1].

Dans un système d'information « traditionnel », disons dans les années 1970-1980, l'architecture de ce qu'on appelle encore « application de gestion » se résume à l'organisation du modèle de données, avec des méthodologies comme MERISE ou Warnier-Orr, très populaires en France, mais quasiment ignorées des pays anglo-saxons, méthodologies fondées sur le langage logique du modèle ENTITÉS – RELATIONS – ATTRIBUTS.

Sur une machine centralisée, le « *mainframe* » des années 1970-1980, le système d'exploitation prend complètement à sa charge tout le contrôle de ce qui se passe dans la machine ainsi que les diverses régulations nécessaires à une répartition équitable des ressources disponibles pour les applications que la machine héberge. Le concepteur d'application est donc déchargé d'un travail difficile de gestion de ressources, qui met en œuvre des algorithmes dont certains ont mis 10 ou 15 ans à trouver leur point d'équilibre, comme par exemple les algorithmes de gestion de la mémoire virtuelle qui donnent au programmeur l'apparence d'une mémoire sans limite, de son point de vue.

Avec l'arrivée des stations de travail, des micro-ordinateurs et des architectures clients-serveurs distribuées ce cadre relativement simple avec un point de contrôle unique vole en éclats. Les architectes-concepteurs de systèmes d'information se retrouvent confrontés aux mêmes problèmes que leurs collègues des systèmes technologiques, mais bien souvent sans avoir un « *back ground* » scientifique suffisant, car limité aux données. Toutefois, avec des systèmes d'information comme les systè-

1. Sur ce sujet, voir les deux ouvrages de J-P. Meinadier, *Ingénierie et intégration des systèmes* et *Le métier d'intégration de systèmes*, chez Hermès-Lavoisier.

mes de réservation des compagnies aériennes (par exemple SABRE pour American Airlines ou AMADEUS pour Air France), l'évolution vers les architectures distribuées complexes, et les problèmes afférents, se dessine déjà clairement (NB : ce type de systèmes gèrent des dizaines de milliers d'équipements sur toute la planète, et absorbent des charges de traitements dont le pic peut dépasser 20 000 transactions à la seconde, avec des contraintes importantes de sûreté de fonctionnement). Le transactionnel[1] naît dans cet environnement

C'est grâce à la présence d'un moniteur transactionnel que le programmeur COBOL a pu passer sans trop de difficulté du monde de la programmation batch à celui de la programmation dite transactionnelle qui est celle de tous les systèmes d'information. Faute de ce moniteur, la gestion des files d'attente de messages, de la mémoire contextuelle spécifique à une transaction, de la gestion des priorités et des ressources entre les transactions, etc., auraient été à la charge du programmeur d'application. Le moniteur transactionnel gère la partie la plus délicate, les interactions et les événements avec l'environnement de la transaction. Ne reste à la charge du programmeur, que la partie transformationnelle concernant les données de l'application.

La même remarque vaut pour la gestion des données faite par les SGBD qui prennent en charge la gestion de l'adressage interne et de l'intégrité des données stockées dans les mémoires de masse, dont le nombre d'occurrences peut aller de quelques milliers à plusieurs dizaines de milliards dans les « magasins de données ».

Dans un système distribué, où les ressources sont réparties entre différents ordinateurs et équipements reliés entre eux par des réseaux, toute cette belle mécanique quasiment invisible au programmeur d'application remonte dans les couches applicatives. Les programmes d'applications doivent alors prendre en charge certains états du réseau qui affectent leurs comportements ; ils doivent se faire notifier des débuts ou des fins de travaux effectués sur d'autres équipements pour pouvoir eux-mêmes se déclarer en état de terminaison. S'il y a des incidents ou des pannes sur le réseau, le programmeur devra s'assurer que l'intégrité des travaux en cours n'est pas affectée, ou que l'on sait arrêter proprement les applications et/ou les systèmes, d'où une interface beaucoup plus complexe avec les équipes d'exploitation. Ce qui était intégré se désintègre, avec comme conséquence immédiate une explosion du coût d'intégration système, quasiment inexistant en centralisé, qui est un nouveau coût à la charge de l'équipe d'ingénierie. L'évolution d'un programme devient l'évolution d'un système. Le concepteur d'une application distribuée doit savoir explicitement qu'elles sont les ressources logiques dont l'application a besoin, de façon à contrôler le mapping de ces ressources par rapport aux ressources physiques effectivement disponibles sur la plate-forme d'exécution.

L'émergence désormais irréversible de l'informatique distribuée et des architectures clients-serveurs, ainsi que la nécessité de la décentralisation géographique des

1. Cf. le système de réservation SABRE définit conjointement par IBM et American Airlines, qui a servi de modèle, beaucoup plus tard, au système SOCRATE de la SNCF dont on se rappelle les déboires. À l'époque la charge était déjà aux alentours de 1 000 TPS.

traitements, transforme le concepteur d'application – c'est déjà une tâche fort difficile – en architecte de système qui doit s'assurer que les aléas qui affectent à l'instant *t* tel ou tel élément du réseau, ne compromettent pas la cohérence globale du système. Le bon fonctionnement de cette nouvelle couche d'infrastructures, appelé « *middleware* » dans le jargon informatique, est le prix à payer des facilités et de la souplesse d'emploi incontestable qu'apporte l'informatique distribuée.

Contrairement à ce qu'un marketing parfois racoleur essaye de faire croire, les progiciels systèmes « *middleware* » sont intrinsèquement complexes car ce sont des programmes réactifs, asynchrones et globalement non déterministes à cause des réseaux ; ils sont loin d'offrir, pour le moment, la même sûreté de fonctionnement que celle que l'on trouvait dans les bons vieux mainframes pour des raisons techniques inhérentes à la complexité des architectures distribuées comme on le verra avec les exemples du *cloud computing* et des microprocesseurs multicœur, et de la virtualisation. Ce sont des logiciels très coûteux à développer, ce qui exclut de les trouver sur des plates-formes à faible diffusion où alors avec une performance et une fiabilité mauvaise, sans parler de leur maintenabilité ou de leur pérennité aléatoire. La conséquence de ce nouvel état de fait est que le concepteur d'application se voit obligé de développer certains mécanismes qui lui étaient jusqu'alors cachés, ou dont il n'avait pas à se soucier, de façon à pouvoir utiliser de façon sûre les parties les plus solides des middlewares ou des environnements d'intégration qui sont à sa disposition. De vraies compétences d'architecte, au sens défini dans cet ouvrage, deviennent indispensables. Sans se transformer complètement en concepteur système, il doit cependant être à même de comprendre ce qui se passe, ne serait-ce que pour gérer correctement tous les évènements nouveaux dont il a désormais la charge : administration automatisée et autorégulée, surveillance des ressources, modes de fonctionnement dégradés et continuité du service, contrat de service (*Service Level Agreement*), etc.

Le but de ce livre est de mieux faire comprendre, par un certain nombre d'exemples et d'analyse, ce qu'est l'architecture logicielle et le métier d'architecte de façon à permettre aux DSI[1], aux chefs de projet MOA/MOE, aux responsables d'équipes de développement et de maintenance, ainsi qu'à la grande masse des programmeurs, de pouvoir continuer à travailler correctement avec une bonne productivité, et de bien comprendre le contexte système dans lequel ils opèrent. C'est une nouvelle responsabilité pour le concepteur d'application et/ou le chef de projet qui doivent en mesurer la difficulté afin d'éviter de se lancer dans des développements dont les fondements ne sont pas solidement établis et qui les conduiraient droit à l'échec[2]. Un nouveau métier apparaît de façon explicite, celui d'architecte de systèmes, au sens large, dont le rôle est d'assurer l'intermédiation entre les métiers déjà connus comme le développement, la maintenance, l'exploitation et les usagers du système généralement représenté par une maîtrise d'ouvrage. C'est un métier très technique qui requiert de grande capacité

1. Cf. le livre de Y. Caseau, *Urbanisation et BPM – Le point de vue d'un DSI*, Dunod, 2011, qui explique ce qu'un DSI doit comprendre de l'architecture pour effectivement manager le SI de l'entreprise.
2. Cf. L'étude du Standish Group : *Chaos chronicles*, version 3.0 ; édition 2003 du rapport.

d'abstraction et de communication. L'architecte doit se préoccuper de l'acceptabilité sociale du système du point de vue de la communauté des exploitants et des usagers métiers, ainsi que de l'intégration globale dans le système d'information de l'entreprise, rôle assimilé métaphoriquement à de l'urbanisme.

Nous aborderons brièvement les aspects économiques (CQFD des projets, TCO et ROI des systèmes) dans la mesure où ces différents coûts sont largement dépendants de la qualité du travail de l'architecte. Les DSI, les responsables de projets de MOA, sans être eux-mêmes nécessairement des architectes, doivent comprendre les enjeux de l'architecture en sachant que le TCO d'un système est engagé dès le début de la vie du système, lorsque l'architecture est fixée dans ses grandes lignes.

L'ouvrage est organisé en quatre parties : la première partie présente la problématique d'architecture dans ses différents aspects ; la deuxième partie est un témoignage vécu par l'auteur sur deux aspects fondamentaux de l'architecture des systèmes, les langages et compilateurs d'une part, les processus et leurs interactions d'autre part ; la troisième partie présente un ensemble de modèles d'architecture de haut niveau basé sur les deux domaines les mieux connus de la technologie informatique, les traducteurs/compilateurs de langages et les moniteurs, qui s'intègrent tous deux dans une problématique plus générale de machines abstraites ; la quatrième partie explique pourquoi il est fondamental pour l'architecte de se préoccuper le plus tôt possible des caractéristiques qualité de simplicité, disponibilité, évolutivité ainsi que de la définition des interfaces.

À l'occasion de la deuxième édition, nous avons profité de l'opportunité qui nous était donnée pour enrichir la quatrième partie avec un chapitre consacré au métier de l'architecte et à la façon dont il devrait aborder les problèmes auxquels il est quotidiennement confronté. Nous avons délibérément choisi de nous intéresser à deux questions de fond et de les approfondir en restant dans les limites de cet ouvrage.

La première question est celle de la complexité qui envahit progressivement nos systèmes informatisés. Nous avons essayé de définir ce qui pourrait constituer les éléments d'une mesure utile de la complexité. Sans une telle mesure, même imparfaite, des questions aussi simples que « Est-ce plus complexe ? Est-ce moins complexe ? » sont dénuées de sens. Or elles sont le quotidien des chefs de projets et des architectes. Quel est l'impact de l'ajout de telle ou telle exigence en termes de coût, qualité, délai ? Quelle est la conséquence sur le contrat de service de la mise en interopérabilité des systèmes S1, S2, S3… ? Quelles sont les limites au-delà desquelles les projets deviennent ingérables ? On est ici très proche de ce que dans les théories du chaos on appelle l'effet papillon, où un battement d'aile à Pékin, déclenche une tempête aux Caraïbes. Et en final, comment se former sur ces sujets particulièrement délicats et souvent contre intuitifs.

La deuxième question est la conséquence de la première. Comment raisonner dans un univers complexe et incertain, là où notre intuition est presque systématiquement prise en défaut. Toute notre formation cartésienne, en particulier en France, privilégie un univers logique fixe, stable, où les vérités sont éternelles, comme en mathématiques, et où rien ne bouge.

La première informatique, jusque dans les années 1980, était majoritairement dans un tel univers. Mais avec l'arrivée des systèmes distribués, le développement des réseaux haut débit, les interfaces hommes-machines graphiques, la compétition économique et la pression des pays émergents, tout s'est mis à bouger et nous sommes aujourd'hui dans un monde hautement dynamique. De plus, les systèmes sont ouverts. Le « *just in time* », l'interaction immédiate, la réponse instantanée à n'importe quelle question inopinée est un avantage compétitif évident, avec en plus l'omniprésence des acteurs humains, isolés ou dans des organisations, au centre de la boucle, et c'est souvent le maillon faible.

Nous avons profité de cette troisième édition pour donner :

a- dans le chapitre 17, un exemple complet, mais succinct, de système complexe de la famille C4ISTAR, parmi les plus avancés du moment ;

b- pour illustrer les changements de comportement des acteurs informatiques nécessités par l'abondance relative des ressources de calcul et de mémoire apportée par le *cloud computing* et les architectures des microprocesseurs multicœurs, dans les paragraphes 10.6 et 11.5.

Comment raisonner sainement dans un tel univers ?

En physique ou en chimie, en biologie théorique[1], on sait ce qu'est un système dynamique. On sait ce qu'est un équilibre. On sait ce qu'est une explosion, ou une onde de choc. Mais les mathématiques qui sont derrières ces phénomènes (où comme par hasard, on retrouve Von Neumann) sont très au-delà des cursus généraux, et très peu enseignées en informatique, où des enseignements de recherche opérationnelle surnagent ça et là. On n'a pas appris à penser la complexité, du moins en des termes utilisables par des ingénieurs[2]. On ne sait pas non plus vraiment mesurer l'information.

En abordant la question de la logique utile à l'architecte de systèmes d'information, nous n'avons pas la prétention de construire ici une théorie de cette logique, c'est totalement prématuré, et au-delà de cet ouvrage, mais de simplement jeter quelques bases qui en feront nécessairement partie. Comme on dit, qui peut le plus, peut le moins. Ce qui est expliqué ici doit être considéré comme indispensable pour faire de la bonne informatique.

En attendant mieux, il faudra que l'architecte « se débrouille » et raisonne comme un ingénieur, comme cela a été souvent le cas dans l'histoire des technologies. La machine a souvent précédé la théorie, comme la machine à vapeur avec la thermodynamique, ou les premiers avions, jusque dans les années 1950.

Nous espérons que les considérations exposées ici, fondées sur notre propre expérience, acquise « sur le tas », comme on dit, l'aideront à conceptualiser les mécanismes qui seuls permettront de canaliser la complexité qui nous envahit, et limiter les

1. Voir C.H. Waddington, qui a beaucoup inspiré R. Thom.

2. Tout le monde pensera à Edgard Morin, intéressant, mais inutilisable par des ingénieurs. Voir les travaux du Santa Fe Institute, au Nouveau-Mexique (www.santafe.edu) et ceux de l'association CESAMES (www.cesames.net).

métastases. Pour le reste, faisons confiance à nos chaires d'ingénierie du Cnam, à celle de systèmes complexes, à l'École polytechnique, qui travaillent activement sur le sujet, avec quelques autres, trop peu nombreuses. Par ailleurs, chacun doit être convaincu que rien ne remplacera le travail personnel, l'intuition qu'il nous faut cultiver en permanence, et que là comme ailleurs, il n'y a pas de solutions toutes faites. Rien ne remplacera jamais l'intelligence.

Remerciements

La matière de ce livre n'aurait pas pu être assemblée sans les deux expériences qui ont le plus marqué mon parcours professionnel.

Bull, dans les années 1970-1980 a su réunir autour du projet DPS7/GCOS7 une somme exceptionnelle de talents, et à vrai dire, unique dans le paysage informatique français et européen de cette époque. Ces talents ont irrigué toute la profession. Le lien direct avec le MIT, avec les concepteurs de MULTICS, avec les architectes de GE et de Honeywell, etc., nous garantissaient l'accès aux créateurs de cette technologie nouvelle révolutionnaire. Ce fut une période magique de grande créativité.

Le ministère de la Défense, par l'ambition des systèmes et systèmes de systèmes nécessaires à la sécurité du pays, par la complexité de leur définition et de leur mise en œuvre, m'a fourni la matière de la partie III. Le contact direct avec les programmes et les architectes de la DGA, ainsi qu'avec les industriels, et plus particulièrement EADS THALES, sont, pour ce qui me concerne, un témoignage de la prudence (au sens de la vertu cardinale, avec ses deux visages, la sagesse de l'expérience et la force de la jeunesse) avec laquelle il faut attaquer ces problèmes extrêmes.

Que tous ceux avec qui j'ai pu interagir et travailler durant toutes ces années soient ici remerciés, ce livre est un peu le leur. Ma secrétaire du CMSL, Jacqueline Pujol, en me soulageant du travail logistique, m'a permis de disposer de plus de temps pour le travail de fond.

Enfin, depuis la première édition de cet ouvrage, l'association CESAMES (cf. www.cesames.net), créée par mon collègue le professeur Daniel Krob, titulaire de la chaire Systèmes complexes de l'École polytechnique, est devenue un lieu d'échanges bien vivant où industriels et enseignants-chercheurs de ce domaine hautement stratégique peuvent partager leurs expériences, réfléchir ensemble au sein de groupes de travail, réfléchir aux nombreux problèmes qui accompagnent le développement de notre société désormais massivement « numérique ». Les problèmes liés à l'accroissement inéluctable de la complexité des systèmes rendent plus nécessaire que jamais l'organisation de cette complexité, c'est-à-dire son architecture. Pouvoir en parler simplement est en soi un progrès qui mérite d'être salué.

Je remercie également les éditions Dunod qui m'ont fait confiance avec patience, car ce livre aurait dû paraître il y a déjà quelques années… mais avec cette troisième édition le retard est désormais comblé.

PARTIE 1

Qu'est-ce que l'architecture du logiciel ?

1

L'architecture dans les sciences de l'ingénieur

1.1 L'ARCHITECTURE AU SENS LITTÉRAL

Le terme architecture, dans le sens originel qu'il a en construction civile désigne tout à la fois :

- Une compréhension profonde des besoins présents et futurs (c'est un facteur d'incertitude) de ceux pour lequel le bâtiment est destiné et des contraintes de l'environnement, y compris les contraintes sociales, dans lequel le bâtiment doit être inséré.
- Une connaissance approfondie des matériaux à assembler ainsi que de leurs limites (usinage, vieillissement, rupture, etc.).
- Une connaissance approfondie des procédés de construction (i.e. les méthodes du génie civil, la gestion de projet, l'acquisition) permettant d'organiser l'activité des différents corps de métiers qui concourent à la réalisation et au maintien en conditions du bâtiment (surveillance, maintenance, adaptation, etc.).

L'architecte est celui qui rassemble et sait mettre en œuvre ces différentes connaissances au service exclusif du but poursuivit par le commanditaire ou son mandant : le maître d'ouvrage. L'architecte est un chef d'orchestre qui doit savoir jouer de plusieurs instruments (pas de tous, car cela est humainement impossible !) et qui doit surtout connaître la musique.

Les architectes du Moyen Âge[1] comme Villard de Honnecourt, Robert de Luzarches ou Pierre de Montreuil, ont excellé à traduire le sentiment religieux de leurs

1. Cf. E.Viollet-Leduc, *Encyclopédie médiévale*, Interlivres – J. Gimpel, *Les bâtisseurs de cathédrales*, Le Seuil, et R. Bechmann, *Villard de Honnecourt – La pensée technique au XIII^e siècle et sa communication*, Picard éditeur, 1993.

contemporains dans ces cathédrales de pierres que l'on ne se lasse jamais de contempler. Comme l'a dit un historien du Moyen-Âge : « La cathédrale est un problème technique victorieusement résolu, mais, dans sa conception, elle reste une inspiration grandiose de la Foi ». Sur les épitaphes de leurs pierres tombales, ou dans la crypte des cathédrales qu'ils ont construites, on trouve des qualificatifs comme « docteur es-pierres » ou « très élevé dans l'art des pierres » qui traduisent l'admiration et la reconnaissance de leurs contemporains dans la maîtrise du matériau de base des cathédrales qui ne sont que des pierres savamment taillées et assemblées.

Vauban quelques siècles plus tard érigera la construction des fortifications en une véritable « science »[2] qui traduisait parfaitement le besoin de défense des cités frontalières du nord et de l'est de la France en intégrant complètement dans le système défensif l'art de la guerre de cette époque. On peut encore admirer tout un ensemble de maquettes dont il fit démarrer la réalisation, au musée des Plans et Reliefs, aux Invalides, à Paris. De toute évidence, Vauban maîtrisait complètement l'art du maquettage, et il savait s'en servir pour expliquer à ses commanditaires la nature des travaux qu'il proposait d'entreprendre. La fortification enterrée était la seule façon de répondre au feu des canons désormais capables de briser n'importe quelle muraille verticale conçue initialement pour résister aux flèches et aux assauts de l'infanterie.

Compréhension du besoin, connaissances des matériaux, capacité d'organisation, motivation des corps de métiers, conviction et persuasion vis-à-vis des autorités religieuses et/ou militaires commanditaires des travaux, forment la trame culturelle et intellectuelle de ces architectes du passé qui étaient également, et avant tout, des ingénieurs[3].

Toutes ces constructions reposent sur une innovation technique au service d'une idée. Les cathédrales n'existent que par la croisée d'ogives, les arcs-boutants et l'art de la taille des pierres comme les clés de voûtes dont la précision requiert une grande connaissance de la géométrie[4], ainsi qu'une très grande capacité d'organisation de chantiers qui pouvaient rassembler plusieurs milliers de personnes dans une grande variété de métiers. La taille des pierres se faisant dans des carrières parfois fort éloignées de la cathédrale[5], donc sur plans ou sur modèles en bois, ce qui nécessitait déjà un excellent niveau de standardisation. La standardisation libère l'imagination et rend la construction possible sans en être pour autant une condition suffisante. La

2. Cf. le remarquable ouvrage de J. Langins, aux MIT Press, *Conserving the enlightenment – French military engineering from Vauban to the Revolution*, 2004.

3. Cf. P. Rossi, *Aux origines de la science moderne*, Ed. du Seuil, collection Points Sciences (chapitre III, Les ingénieurs) ; Le Corbusier, *Vers une architecture*, Flammarion ; *L'art de l'ingénieur - constructeur, entrepreneur, inventeur*, Ed. Centre Georges Pompidou – Le Moniteur.

4. C'était un des arts libéraux enseigné dans nos universités du Moyen-Âge ; il est certain que les bâtisseurs de cathédrales connaissaient Euclide, mais la taille des pierres requiert des savoir-faire précis qui se transmettaient par compagnonnage.

5. Les pierres de la cathédrale de Cantorbéry, en Angleterre, viennent de la région de Caen réputée pour la qualité de son calcaire.

construction doit également être « belle », selon les règles esthétiques de l'époque. Il faut donc rajouter du subjectif et du qualitatif pour rendre la construction socialement acceptable. Les soldats de Louis XIV se sentaient bien protégés quand ils étaient en garnison dans un fort conçu par Vauban ; ils avaient confiance dans leur commandement.

Une construction est « belle » si elle est adaptée à sa finalité, ce qui n'est jamais acquis d'avance. La recopie sans l'intelligence de l'environnement qui lui a donné naissance mène rapidement à la dégénérescence. Ce qui était beau devient conventionnel. Les lignes de forces, si visibles et naturelles dans les premières cathédrales disparaissent sous les fioritures du gothique flamboyant. L'intention primitive cède le pas à la prouesse technique, comme les voûtes du cœur de la cathédrale de Beauvais qui s'effondrent en 1284. La guerre de mouvement napoléonienne rend obsolète l'idée même de place forte, symbole de la guerre de position, qu'il est très facile de contourner avec une cavalerie et surtout une artillerie mobile, afin d'en désorganiser les arrières ; une évidence oubliée par les constructeurs de la ligne Maginot. On sait que Napoléon disposait d'un système de communications hérité de la révolution (le télégraphe de l'ingénieur Chappe) et d'une artillerie mobile et puissante (le canon Gribeauval) qu'il a magistralement exploitée dans des mouvements fulgurants, relativement à l'époque, jusqu'à ce que ses adversaires fassent comme lui.

Rien n'est donc jamais définitivement acquis et l'architecte doit évidemment faire preuve d'une très grande ouverture d'esprit et d'une vraie capacité d'anticipation : « il pense à cent ans » disait Le Corbusier.

Plus près de nous, Le Corbusier a su tout à la fois imaginer des solutions architecturales et urbanistiques qu'autorisaient les possibilités de nouveaux matériaux comme le béton armé ou l'usage systématique du fer, déjà utilisé dans les cathédrales. Le Corbusier définissait la ville comme un outil de travail qui doit être adapté à sa fonction[6] administrative, militaire, protectrice, marchande, conviviale, etc. L'ordre géométrique était pour lui d'une importance primordiale tout autant qu'un facteur d'efficacités des actions humaines ; il disait que « l'architecte, par l'ordonnance des formes, réalise un ordre qui est une création de son esprit »[7]. Il distinguait soigneusement l'architecture « un fait d'art » de la construction « c'est pour faire tenir ». Pour faciliter la reconstruction des villes détruites durant la seconde guerre mondiale, il avait même créé une unité de mesure : le *modulor*[8] qui permettait, disait-il, de construire en série des maisons variées à partir de blocs modulables mais respectant des proportions agréables aux humains. En quelques sortes, des règles de modularité et de réutilisation avant la lettre ! Une ville comme Le Havre, complètement détruite en 1943-1944, fut rebâtie par A. Perret, disciple de Le Corbusier, sur ces principes ; elle est aujourd'hui classée au patrimoine mondial de l'UNESCO.

6. C'est le concept d'urbanisme, au sens littéral ; cf. Le Corbusier, *Urbanisme*, Flammarion.
7. Cf. Le Corbusier : *Vers une architecture*, Flammarion.
8. Cf. *The modulor*, Faber paperback, 1951.

1.2 LIMITES DES MÉTAPHORES ARCHITECTURALES

Tout ce qui vient d'être dit de l'architecture et du génie civil, se transpose quasiment mot pour mot dans celui des grandes « constructions immatérielles » que sont les logiciels systèmes ou les systèmes informatisés.

On peut cependant s'étonner de l'emploi de mot comme architecture ou urbanisme dans un domaine aussi abstrait que celui du logiciel. Il y a évidemment de grandes différences, et la métaphore architecturale et urbanistique cesse rapidement d'être pertinente, pour ne pas dire trompeuse. On n'agit pas de la même façon sur un logiciel que sur des pierres ou du béton, fut-il qualifié de « matière grise » comme disait Francis Bouygues ! Le « matériau » informatique qui intègre du sens et de la sémantique, une forme de « vie » avec des comportements bien particuliers selon les contextes, n'a rien à voir avec les matériaux inertes du monde physique. « Analogie ne veut pas dire similitude, il faut se méfier des fausses simplifications ».

Toute architecture est basée sur l'intuition que nous avons de la forme et de l'espace[9]. Les architectes plus que les autres ont évidemment besoin de cette intuition. Les formes géométriques pré-existent, au sens platonicien ; la construction révèle et matérialise la forme. On pourrait dire que l'architecture c'est l'art des formes que les procédés de construction et nos règles sociales autorisent.

Dans une construction artificielle comme l'est n'importe quel programme informatique, la forme géométrique n'existe pas. La forme logique d'un programme doit être créée par l'architecte du logiciel, ou par le programmeur. La seule analogie pertinente à laquelle on puisse se référer nous est donnée par le développement des mathématiques modernes (i.e. à la fin du XIX^{e} siècle) où émerge la notion de structure mathématique. La prolifération des faits mathématiques, des théorèmes et des théories particulières rendent progressivement indispensable la nécessité d'une remise en ordre, ne serait-ce que pour des raisons pédagogiques et de mémorisation des faits mathématiques, associées à un puissant besoin de généralisation comme on peut le voir dans la théorie des équations algébriques et les variétés de géométrie. Les mathématiciens considèrent qu'Henri Poincaré et David Hilbert, son cadet de dix ans, ont été les derniers géants des mathématiques à pouvoir embrasser toutes les mathématiques de leur époque. La matière mathématique est simplement devenue trop riche par rapport aux capacités des cerveaux les plus doués.

D. Hilbert, et son école de Göttingen, ont été les architectes de cette remise en ordre dont on peut dire que Bourbaki a été l'héritier putatif[10]. La construction d'abstraction efficace est un puissant moyen d'emmagasiner plus de faits mathématiques dans une vie de mathématiciens et donc d'en étendre encore le champ afin de faire reculer les limites

9. Sur la notion de forme, l'ouvrage de D'Arcy Thomson, *On growth and forms*, Cambridge University Press, est incontournable. Voir également, R.Thom, *Stabilité structurelle et morphogenèse*, Interéditions, 1977.

10. Sur cette re-fondation, voir l'ouvrage de C. Reid, *Hilbert*, Springer, et le dossier *Bourbaki* dans les cahier de *Pour la science*. Egalement, les deux ouvrages de J. Dieudonné, *Pour l'honneur de l'esprit humain* et *Le choix bourbachique*.

de ce qu'il pourra maîtriser. Sans cette vue abstraite, c'est l'enlisement immédiat dans le sable des particularismes. La même remarque vaut en physique[11].

Quiconque a fait un peu de mathématiques (disons au niveau des classes préparatoires ou de la licence scientifique) peut se faire une idée très précise de la puissance de l'abstraction dans le nombre de pas de raisonnement nécessaires à la démonstration de tel ou tel théorème pénible (Bolzano-Weierstrass, Sturm, la théorie classique des coniques, etc.). Un calcul absolument fastidieux en coordonnées cartésiennes devient trivial en coordonnées affines ou en coordonnées polaires. Le choix du système de représentation d'un problème géométrique, adapté à ce problème, est un élément simplificateur dans l'expression du problème qui va rendre les démonstrations sinon limpides du moins plus faciles à mémoriser, ce qui permettra d'autres rapprochements, et de nouvelles abstractions. Sans le symbolisme du calcul tensoriel, la théorie de la relativité générale serait quasiment inexprimable, et c'est un fait connu qu'Einstein mit plusieurs années à en maîtriser le maniement[12]. La même remarque vaut pour l'emploi des espaces de Hilbert par J. von Neumann en mécanique quantique[13]. Von Neumann, très impliqué dans l'invention des premiers ordinateurs, créateur de l'architecture dite de von Neumann, lui même mathématicien de première force, utilisait dans ses écrits[14] sur les « *computing instruments* » des expressions comme « *logical design* » pour bien distinguer la cohérence logique de la machine de sa réalisation matérielle qui en était la traduction. La forme architecturale d'un logiciel est sa forme logique qui ne se perçoit pas sans une certaine éducation à la rigueur mathématique et à la méthode scientifique[15]. Les notions d'automates, de système à états-transitions, de traduction, de grammaire, de machines abstraites sont des formes logiques fondamentales pour organiser et structurer les logiciels.

A *contrario*, un informaticien issu des sciences « molles », par définition peu familier des abstractions et de la rigueur logique, aura des difficultés à saisir l'essence abstraite de l'architecture qu'il ne distinguera pas de la réalisation matérielle représentée par le programme. N'ayant pas à sa disposition le référentiel culturel et les instruments de pensée indispensables à la perception des abstractions, il ne pourra pas s'en faire une idée précise, d'où des réactions de rejets face à ce qu'il considérera comme de la théorie sans intérêt.

Cependant, comme pour le gothique flamboyant, on peut faire de l'abstraction, en quelque sorte pour le plaisir, avec comme seul objectif de faire plus compact mais sans rien créer de nouveau. Créer un nouveau langage n'est pas la même chose que créer de nouveaux concepts ; une sténographie n'est pas un langage, c'est juste un moyen d'écrire plus vite. Ce fut beaucoup reproché à Bourbaki[16], ou à des théories comme la théorie des catégories[17].

11. Voir par exemple l'ouvrage de O. Darrigol, *Les équations de Maxwell*, Belin, 2005.
12. Cette histoire est fort bien racontée dans : J-P. Auffray, *Einstein et Poincaré*, Ed. Le pommier.
13. Cf. son livre fameux : *Fondement mathématique de la mécanique quantique* (1ère édition en allemand, en 1927, en français en 1946).
14. Cf. œuvres complètes, volume VI.
15. On peut recommander la lecture de l'ouvrage de K. Popper, *La logique de la découverte scientifique*, Payot, 1970 et 1984, 1ère édition en 1934 (en allemand).

De tout cet effort, il ressort qu'il y a deux catégories d'abstractions : a) les abstractions utiles, car non seulement elles rendent la description plus simple (i.e. plus courte ! en prenant le point de vue de la théorie algorithmique de l'information) mais en plus elles ouvrent des portes ; et b) d'autres, qui ne sont que des abstractions stylistiques, que R.Thom qualifiaient d'« algébrose ». On verra, avec l'exemple des techniques de compilation ébauchées en partie 2, la puissance de l'abstraction au service d'une des technologies sur laquelle repose toute la programmation. Une classe logique n'est véritablement intéressante que si elle contient plus d'un élément, sinon on a deux noms pour la même chose : un nom propre et un nom de classe. Les logiciens du moyen age, contemporains des bâtisseurs de cathédrales, dans leur effort de simplification de l'exposé de la théologie, disaient qu'il ne fallait pas créer de catégories inutiles. Guillaume d'Ockham[18], moine franciscain du XIVe siècle, nous a donné une des premières expressions du principe de simplicité ou de parcimonie, la bonne vieille règle dite du rasoir d'Ockham, sous différentes formulations comme : *Entia non sunt multiplicanda praeter necessitatem* (Il ne faut pas multiplier les choses au-delà du nécessaire) ou *Frustra fit per plura quod fieri potest per paucioria* (On fait vainement avec beaucoup de choses ce qu'on pourrait faire avec peu de chose). Thomas d'Aquin qui fut un grand professeur de l'université de Paris un siècle plus tôt, vers 1250[19], disait dans sa *Somme contre les gentils*, au livre I,I et II, XXIV,3 : « […] L'office du sage est de mettre de l'ordre. […] Ceux qui ont en charge d'ordonner à une fin doivent emprunter à cette fin la règle de leur gouvernement et de l'ordre qu'il crée : chaque être est en effet parfaitement à sa place quand il est convenablement ordonné à sa fin, la fin étant le bien de toute chose. Aussi, dans le domaine des arts, constatons-nous qu'un art détenteur d'une fin, joue à l'égard d'un autre art le rôle de régulateur et pour ainsi dire de principe. […] Il en va de même du pilotage par rapport à la construction des navires, de l'art de la guerre par rapport à la cavalerie et aux fournitures militaires. Ces arts qui commandent à d'autres sont appelés architectoniques, ou arts principaux. Ceux qui s'y adonnent, et que l'on appelle architectes, revendiquent pour eux le nom de sages. […] Ceux qui dans le domaine technique coordonnent les différentes parties d'un édifice sont appelés des sages par rapport à l'œuvre réalisée ». Les mots de Saint Thomas ont une saveur étrangement moderne : « piloter », « gouverner », « réguler », « finalité » viennent de la même racine grecque (*cyber*) que N.Wiener utilisa pour créer le mot cybernétique. L'architecte est fondamentalement quelqu'un qui met de l'ordre, qui crée de la

16. Cf. R. Thom, *Les mathématiques modernes, une erreur pédagogique et philosophique*, dans *Apologie du logos*, Hachette, 1990 ; également B. Beauzamy, *Méthodes probabilistes pour l'étude des phénomènes réels*, SCM 2004, ISBN 2-9521458-06.

17. N'étant pas mathématicien, ceci n'est pas un jugement mais juste une remarque, d'autant plus qu'avec A. Grothendieck et sa théorie des schémas (ou topos, ou motifs) on pourrait certainement discuter. Voir son ouvrage autobiographique, *Récolte et semailles* (jamais publié, disponible sur Internet).

18. Le personnage de Frère Guillaume, dans le roman de U. Ecco, *Au nom de la rose*, lui ressemble comme un frère.

19. À cette époque, les chantiers de nos grandes cathédrales étaient ouverts : 1163 Paris, 1190 Bourges, 1194 Chartres, 1211 Reims, 1220 Amiens, et la Sainte Chapelle à Paris, joyau de l'art gothique, en 1243, que Thomas d'Aquin, familier du roi St-Louis, a vu construire.

simplicité, pour la satisfaction du plus grand nombre, qui régule pour organiser le chantier de mise en œuvre.

Plus près de nous, l'épistémologie génétique de J. Piaget[20] a mis en évidence l'importance de la construction de ces abstractions dans les phénomènes d'apprentissage et du développement de la maturité, plus particulièrement chez l'enfant. L'assimilation cognitive va toujours du concret vers l'abstrait ; cet ordonnancement a une grande importance pour l'assimilation correcte et la maîtrise progressive des mécanismes cognitifs. L'inverse est anti-pédagogique. On retrouvera cet ordonnancement dans la démarche de conception des systèmes informatisés où l'on analyse en détail le besoin à l'aide de cas d'emplois et de modèles du système à informatiser, fut-il humain, pour, après coup, en abstraire la partie qui sera programmée et automatisée. De même, en théorie des fonctions on passe d'une description en extension à l'aide de tables à une description en intention avec des formules de calcul.

1.3 ARCHITECTURE DE L'INFORMATION

Dans les sciences de l'information, l'identification des processus à informatiser et l'organisation de ces processus, les acteurs humains et les équipements qui les animent, sont la matière première de l'information. À partir de cette réalité informationnelle, le travail d'agencement des instructions et des données pour construire les programmes, la prise en compte des événements qui déclencheront tels ou tels automatismes programmés, le choix de telles ou telles technologies, l'ordonnancement correct des opérations qui transforment progressivement les données initiales en résultats exploitables, etc., que l'architecte/programmeur effectue dans son activité quotidienne, les « preuves » et tests qu'il doit fournir de la justesse de ses décisions et de la validité de ses constructions, etc., a de nombreuses similitudes avec le travail du mathématicien et/ou du physicien face à la réalité du monde physique. La grande différence est que les abstractions fabriquées par le programmeur vont être exécutables et testables sur un ordinateur : l'erreur est immédiatement visible, et ses conséquences peuvent être graves selon les missions dévolues au système informatisé, mais cela permet la validation dans un monde virtuel simulé, avant la mise en service opérationnelle dans la réalité !

Comme le mathématicien ou le physicien, l'architecte / programmeur cherche à construire des entités logiquement cohérentes qui traduisent fidèlement l'intuition du sens profond qu'il a des choses qu'il veut modéliser et abstraire[21]. Comme lui, il cherche les abstractions utiles qui vont diminuer la taille du programme qu'il fabrique, et cela non pas grâce à des astuces artificielles douteuses comme une sténographie, mais par une compréhension profonde de la structure logique et du sens des entités qu'il manipule. Dans sa forme achevée, cette compréhension se manifeste

20. Cf. *Introduction à l'épistémologie génétique*, 3 Vol. et *L'équilibration des structures cognitives*, aux PUF.

21. Sur le rôle de l'intuition dans les sciences, voir le joyau méthodologique de G. Polya, *Les mathématiques et le raisonnement plausible*.