Reconcevoir le système d'information : Une architecture de composants dans un environnement distribué construite pour la réutilisation à l'aide d'une approche framework

Evelyne CHARTIER
Expert Audit et Sécurité des Systèmes d'Information
et de Communication à la BNP

RÉSUMÉ

La construction d'application élaborée à partir de scénarios d'entreprise consistant à découper des processus en services et à les implémenter en frameworks est un changement culturel important dans la méthode de développement de logiciels.

Les méthodes de conception classiques avaient l'objectif d'améliorer le dialogue entre l'informaticien et l'utilisateur afin de réduire l'écart entre le besoin réel et l'application implémentée. La manière de travailler change puisque le développeur doit chercher à utiliser ce qui a été codé et testé et concevoir en fonction de la réutilisation a priori.

L'architecture du système d'information est au service des métiers de l'entreprise. Cela signifie que l'approche méthodologique a changé de sens et que la nature du travail a évolué. La nécessité de s'abstraire pour définir des invariants de composants de système d'information provient de nos objectifs de réutilisabilité et d'interopérabilité.

Le système d'information vu du client est le savoir-faire à modéliser dans le modèle externe. Ce n'est pas la vision habituelle des développeurs qui conçoivent le modèle externe uniquement comme une interface. De telles approches sont structurées par ligne de métiers, évolutifs par nature et non par le traitement de l'information. C'est un renversement de perspective. Il existe un point de rupture dans l'objectif à atteindre et dans les moyens à mettre en oeuvre, un déplacement des responsabilités qui font qu'une entreprise doit prendre conscience que l'architecture du système d'information est une cellule à part entière, c'est-à-dire ayant son autonomie de fonctionnement.

Mots-clés : Frameworks, Re-engineering, Système d'information, Objets Distribués, Patterns, Architecture de composants, Réutilisation, Document composite, BPR (Business Process Reengineering), Workflow, Workgroup, Internet.

ABSTRACT

We suggest building the applications starting from enterprise scenarios and breaking down processes into services which are implemented in frameworks. This is an important cultural change in the software development method. The developer has to use what has already been coded and tested as a function of a priori re-use.

The classical approaches had the objective to improve the dialog between the software developer and the enduser to reduce the difference between the real needs and the implemented application.

The way we work is changing. The software developer must re-use what has been already tested and designed with the a priori re-use.

The architecture of the information system is a consequence of the businesses of the firm. Therefore, the methodological approach has changed and the nature of the work has evolved. A higher level of abstraction is required in order to define invariants in the information system and to ensure re-usability and interoperability.

Such approaches are business driven, and not technology driven. There is a break between the objective and means, a change in the responsabilities. The architecture of the information system is a cell by itself, and this should be understood by the firm. The information system is structured with actors, external and internal services and events which represent the reality of the life of the enterprise.

Key-words : Frameworks, Re-engineering, Information System, Distributed objects, Patterns, Componant Architecture, Re-use, BPR, Compound document, Workflow, Workgroup, Internet.

INTRODUCTION

Le premier besoin d'une banque d'aujourd'hui est de mettre au point une méthode d'ingénierie de fabrication et de réutilisation de services qui permette le développement rapide d'applications offertes par Internet et par le commerce électronique.

Avec la mise en œuvre d'atelier de génie logiciel, l'informaticien abandonne l'approche segmentée par applications au profit d'un assemblage de fonctions. Il s'agit de gérer l'équilibre entre les besoins individuels et les exigences de l'entreprise, de travailler par capitalisation d'expériences et de savoir-faire acquis. Cette démarche doit être simple pour construire une architecture de services. Elle doit permettre la création rapide de composants logiciels stockés dans un référentiel d'entreprise.

La méthode qui consiste à étendre les modèles de données, fonctionnel et processus en un modèle objet est beaucoup trop analytique, trop lourde et trop compliquée pour les développeurs. Elle relève d'un travail de spécialistes en modélisation. Elle est incompatible avec un développement rapide d'application. Il faudrait éviter les consolidations de modèles faites *a posteriori* ainsi que les modèles eux-mêmes au profit de structures génériques d'intérêt commun aux applications et plus stables puisque validées une fois pour toutes.

L'architecture client-serveur dite de "troisième génération" évolue vers les objets distribués et communiquant de manière dynamique à travers les réseaux de télécommunications. L'article propose une démarche de création des services dynamiques personnalisés à la demande du client d'une banque et commercialisables sur le marché des technologies de l'information et de la communication.

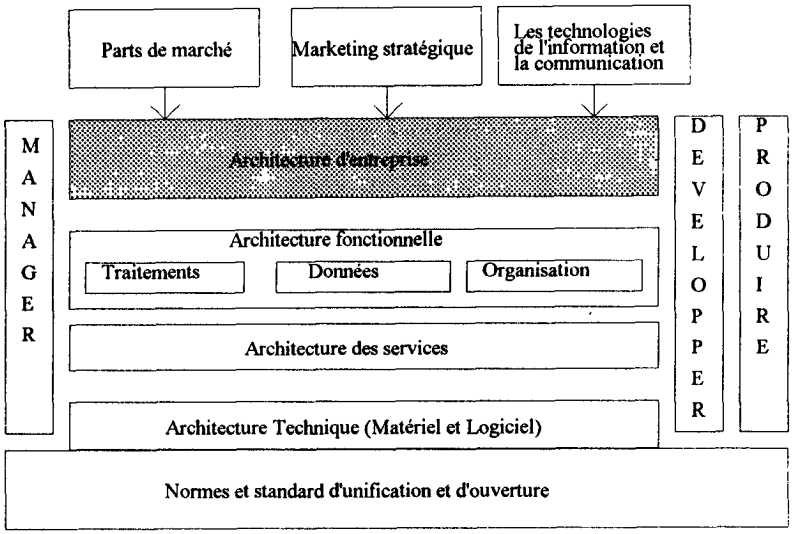
C'est la vision externe du système d'information de l'entreprise qui devient la pièce maîtresse de la solution de l'architecture. Elle s'intéresse aux scénarios de services vus du client et aux rôles des acteurs internes et externes à l'entreprise (partenariats et co-branding). A l'opposé du BPR, la vision transversale du système d'information devient la ligne de force du maillage fondamental de l'architecture. Elle a pour objectif d'analyser les métiers, de valoriser les savoir-faire et de les proposer comme produits dans le circuit commercial. Voici comment peut se positionner l'architecture d'entreprise dans un environnement traditionnel : cf. schéma ci-après.

1. LA RECONCEPTION DU SYSTÈME D'INFORMATION

1.1. Les approches objet classiques

Un des premiers objectifs assignés aux technologies objet est d'améliorer le processus de développement de logiciels grâce à la réutilisation de code et du design des applications. Il faut éviter de redécouvrir l'entreprise à chaque projet. Les technologies objet ont comme ambition de permettre la création d'une infrastructure (charpente logicielle) facilitant la cohérence et l'intégration des différents systèmes d'information de l'entreprise (architecture de système d'information construite *a posteriori*). Cette infrastructure intègre notamment la possibilité d'avoir des applications consti-

Positionnement de l'architecture d'entreprise dans un environnement classique



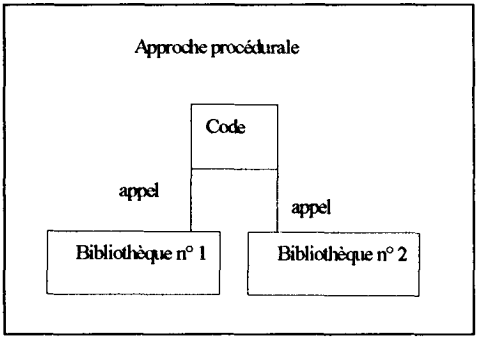
tuées de composants logiciels assemblés ou modifiés dynamiquement.

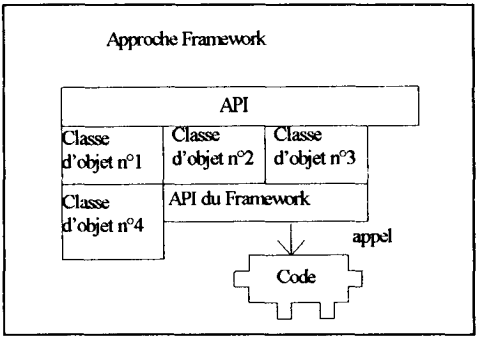
La dynamique de l'entreprise et du système d'information imposent des évolutions. Il faut donc concevoir des applications qui peuvent changer de composants en cours d'exécution. C'est changer de modèle d'analyse et de design : les interactions entre les différents composants ne sont plus figées *a priori*. Les composants peuvent également être sollicités sans connaissance *a priori* des comportements et du contexte des transactions.

C'est le modèle MPVSC (Modèle de données - Présentation - Vue - Sélection - Commande) qui permet d'implémenter les modèles de développement et produit un support pour la réutilisation des composants.

La première génération des systèmes orientés objet s'appuie sur des bibliothèques de classes qui regroupent des éléments réutilisables *mais indépendants les uns des autres*. Les morceaux de code réutilisés ne fournissent que *peu de comportements par défaut, le flot de contrôle reste dans l'application développée*.

La librairie ne fournit pas *une norme d'interface entre applications*. L'approche reste fondamentalement la même qu'en programmation procédurale, avec une meilleure encapsulation. Avec cette approche, les taux de réutilisation restent trop faibles pour justifier des investissements à réaliser. L'utilisation de framework élève le niveau d'abstraction et donc le niveau de réutilisation, facilite l'assemblage des objets et permet de profiter de l'expérience "fonctionnelle/métier" accumulée.





Les méthodes d'analyse et de design de la technologie objet classique sont basées sur un modèle abstrait (structurant et à caractère sémantique). Il est une représentation la plus fidèle de ce qui se communique entre le développeur informaticien et l'utilisateur, spécialiste d'un domaine de l'entreprise. Le modèle sert à spécifier les besoins en informatisation des utilisateurs. Les modèles utilisent les concepts entité/relation/attribut étendus aux propriétés de l'objet (encapsulation, polymorphisme, héritage simple et multiple).

Pour construire une architecture de système d'information, on a envisagé la consolidation des modèles conceptuels représentant un domaine de l'entreprise à informatiser. Ces méthodes dotées de niveau conceptuel étaient faites pour le traitement de l'information.

Les modèles "d'entreprise" n'avaient ni les outils suffisamment perfectionnés pour valider l'activité de modélisation *ni la sanction de la réalité de l'entreprise.* Ils servaient de base de compréhension pour réaliser une application en fonction des besoins définis par l'utilisateur et non par les résultats attendus par l'informatique. Ce niveau sémantique était intéressant pour dialoguer avec le spécialiste du domaine et permettre à l'informaticien développeur de comprendre l'univers d'intérêt.

Les approches OOA/OOD (Object - Oriented Analysis / Object - Oriented Design) peuvent être dirigées par les données, par les événements ou par les scénarios suivant les différentes méthodes (OMT (Object Modeling Technique), Fusion, Schlaer&Mellor, Coad/ Yourdon OOSE (Object - Oriented Software Engineering), etc.

Ces méthodes sont un point de départ important pour former de bons concepteurs du système d'information et pour construire les bases d'une architecture applicative. Mais, il s'agit d'une vision figée au sens de la dynamique : enchaînement des états d'un système enregistrés et non transformation du système d'information au cours du temps prise comme une photographie du système d'information à l'instant t et une représentation cartographique du domaine de l'utilisateur. La carte n'est pas le territoire, dit-on en sémantique générale.

Les phases d'analyse, de design et de code doivent évoluer sur des échelles de temps courtes pour

répondre à une véritable dynamique du système d'information. En même temps, elles doivent être guidées par une vision à long terme pour construire une architecture du système d'information.

La nouvelle approche concerne l'analyse basée sur les métiers de l'entreprise. La réalité du métier de l'entreprise change au cours du temps en fonction des acteurs externes et internes à l'entreprise, des marchés concurrents, de l'ouverture aux plans européen et international, et des volumes traités. Nous allons voir en quoi consiste la rupture vers les nouvelles approches de re-engineering du système d'information (Chartier, 1996).

1.2. Les nouvelles approches objet dans un monde distribué

L'entreprise connaît de graves problèmes dans des périodes de crise mondiale et d'ouverture sur des marchés très concurrentiels au niveau international. Notre vision est que les entreprises sont en train de recentrer leurs activités sur leur mission et leur savoir-faire.

Elles mettent en œuvre des stratégie de survie dans lesquelles s'élaborent des scénarios d'entreprise. *Le système d'information est comme un modèle des métiers*, ce qui nécessite sa refonte puisqu'il devient un outil d'analyse stratégique. Les méthodes de développement et techniques appropriées, ainsi que l'organisation informatique sont complètement modifiées. Cette transformation se fait en trois phases.

La première phase consiste à restructurer le département des développements applicatifs en fonction des grandes directions opérationnelles et à organiser le secteur *administration* du système d'information.

La deuxième phase voit une décentralisation vers des entités opérationnelles qui elles-mêmes *sont restructurées en lignes de métiers de l'entreprise.*

La dernière phase correspond à une filialisation de certaines entités métier.

L'architecture de système d'information concerne uniquement *le métier* de l'entreprise. Elle vise à repérer ses acteurs, à identifier des services et à enregistrer des événements qui font son activité (au sens de faire du "business"). L'avenir pourrait être de créer une entreprise "virtuelle" recomposée dynamiquement en fonction des besoins des acteurs socio-économiques et pour chaque spécialiste du métier comme le secteur bancaire, industriel, etc.

Elle permettrait à terme de commercialiser une partie ou la totalité des composants du système d'information et de s'ériger en prestataire de services de son propre métier en créant des structures SSII, des filiales travaillant à son profit. Cet archipel de l'information serait constitué de centres de profit, des acteurs de l'économie. Les centres de coût deviennent ainsi des centres de profit. Un point de rupture s'établit avec le passé. C'est la technologie cette fois qui par ses possibilités informatiques nous met en position de définir une couche "composants métiers du système d'information". Mais il faut attendre que les outils soient opérationnels et que la standardisation soit effective pour penser l'informatique au bout des doigts.

2. DES SERVICES IMPLÉMENTÉS COMME DES FRAMEWORKS

2.1. L'architecture des services

Le système d'information définit *un modèle des métiers*. Le métier est analysé à partir des processus opérationnels adaptés à chaque canal : voix, minitel, GAB (Guichet Automatique de Billets de Banque), Internet, etc. (Chartier, 1996).

Ces processus sont élaborés jusqu'à présent sans lien aucun les uns avec les autres. Ils correspondent à l'approche événementielle qui, par l'intermédiaire d'un acteur, agit sur le système en rendant des services. Suivant la typologie événementielle (qui peut être de niveau entreprise, interne à une application ou entre des applications), la réponse c'est-à-dire le service rendu est plus ou moins réutilisable. C'est le domaine du BPR qui introduit un niveau de *généricité transversal* de définition du système d'information.

Les scénarios construits à partir du poste de l'utilisateur sont une bonne base de départ pour trouver des services métier puisqu'ils correspondent à des situations réelles.

L'idée est d'avoir une vision transversale du système d'information de manière à *typer la nature des événements* et *à relier les processus opérationnels les uns aux autres*. Le système d'information se construit à partir de services dont la généricité dépend de la granularité en termes de réutilisabilité (entreprise, interne à une application, entre applications). Ces services génériques sont encapsulés dans des structures d'information correspondant à des dynamiques de système d'information. Ils traduisent des comportements liés aux scénarios d'entreprise. C'est *la conception dirigée par le métier. Ces composants sont appelés frameworks.* Nous construisons ainsi l'architecture *des services de l'entreprise* qui est le système d'information (cf. schéma ci-après).

2.2. L'implémentation de l'architecture des services en frameworks

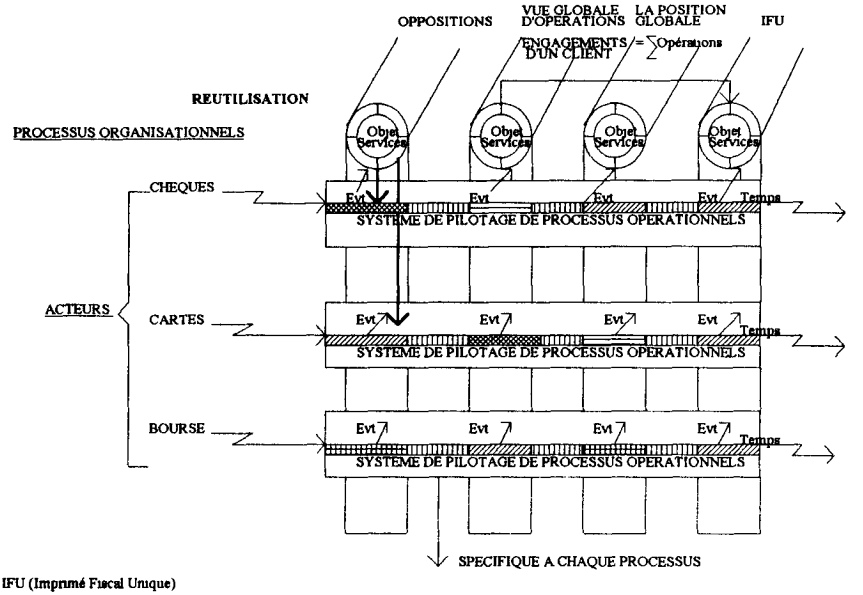
Cette approche framework substitue à la notion de librairie (outil de rangement), celle de collection de classes cohérentes destinées à *coopérer* ensemble pour répondre *à une catégorie de problèmes dans un domaine précis.*

Un framework impose *une méthode de décomposition d'un problème* et une interface externe (par la fourniture d'un ensemble de classes abstraites). Il est chargé d'assurer *la gestion du flot de contrôle* et fournit des comportements par défaut pour les classes d'interfaçage qui n'ont pas été spécialisées par le développeur.

Un framework sert normalement de base pour un développeur spécifique par spécialisation des classes qu'il définit : il sert de cadre au développement et à la spécification. Il est cependant possible de l'étendre par adjonction de nouvelles classes, le résultat demeurant un framework destiné à un domaine différent ou plus spécifique.

L'utilisation de frameworks élève le niveau d'abstraction donc le niveau de réutilisation, facilite l'assemblage des objets et *permet de profiter de l'expérience "fonction-*

Système de pilotage de processus gérant le parallélisme défini par fonction de la banque donnant une vision transversale



IFU (Imprimé Fiscal Unique)

SERVICE DE BASE : LIEU DE RENCONTRE INSTANTANE ENTRE UN EVENEMENT, DES DONNEES ET DES REGLES DE CIRCONSTANCE (ACTEURS EXTERNES ET INTERNES)

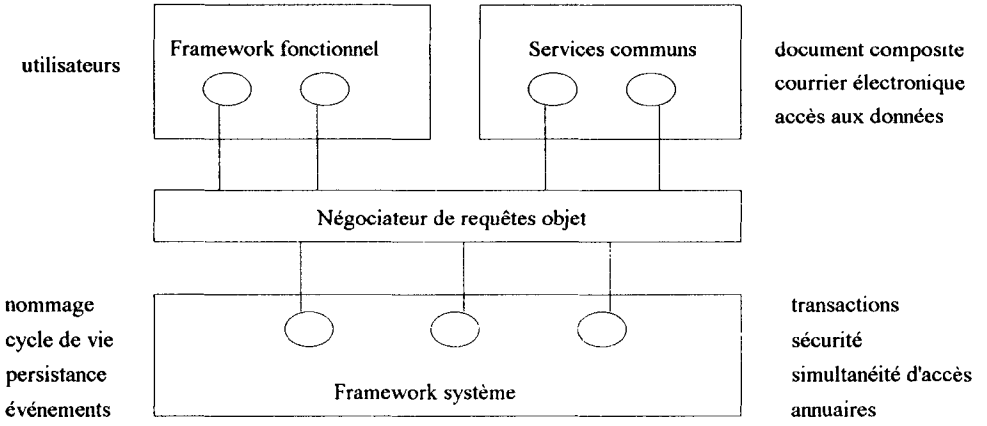
nelle/métier" accumulée qui est le savoir-faire du concepteur.

La notion de composant objet est une extension fonctionnelle et applicative de la notion de framework. Un composant fournit un service complet tout en étant capable de *s'intégrer dynamiquement* à d'autres composants *pour générer un service plus complexe* (par exemple, les oppositions sur chèques, sur les cartes bancaires, au guichet de l'agence, etc.).

Un composant dispose d'interfaces dynamiques très générales et standardisées dans un domaine qui lui permettent de dialoguer avec d'autres composants du même domaine. L'utilisation de composants assure la maintenance dynamique d'applications : changer, mettre à jour ou ajouter un composant ne nécessite plus nécessairement une mise à jour du reste de l'application. Les capacités d'intégration dynamique des composants ne peuvent cependant être obtenues que dans le cadre d'une norme d'implémentation et de design (pattern) qui définit les interfaces d'interaction.

Ce processus généralise des structures et des comportements de système d'information capables de se reproduire au niveau métier, au niveau applicatif et au niveau système.

Le modèle des services



Ainsi, le modèle des services est décomposé en :

- services *techniques* définissant l'architecture technique (les standards) grâce à l'interopérabilité des matériels (par exemple : les interfaces de normalisation et de standardisation Corba (Common Object Request Broker Architecture) de l'OMG (Object Management Group) ;

- services *fonctionnels* définissant une architecture applicative (par exemple, les formats de message, ...) fabriquant des briques logicielles réutilisables ;

- services *métiers* définissant l'architecture des métiers de l'entreprise (par exemple : la comptabilité, le portefeuille financier ou commercial, ...).

Ces services forment un puzzle cohérent élaboré à partir d'un modèle à Lego. L'architecture de système d'information se définit sous forme de couches hiérarchisées de services de niveau système, applicatif et métier. Actuellement, les services techniques et fonctionnels sont réalisés grâce aux architectures client-serveur existantes. Aujourd'hui, les services fonctionnels sont appelés abusivement services métiers. L'architecture des services que nous venons de présenter explique clairement la voie à suivre pour construire le système d'information vu du client.

Cette architecture des services correspond à l'architecture de l'outil CommonPoint fabriqué par la société Taligent (IBM, Apple et Hewlett Packard) et racheté en 1995 par la Compagnie IBM.

3. L'OUTIL COMMONPOINT COMME POINT DE DÉPART

3.1. L'architecture de l'outil CommonPoint

L'outil CommonPoint contient cp-Constructor et cp-Professional.

Cp-Constructor est un générateur d'interfaces (IHM) pour des applications distribuées de CommonPoint. *L'interface est un document de type "Container" (ou* plus précisément ce qu'on appelle *un*

"dossier composite" dans le cas d'un ensemble d'opérations bancaires liées à un client).

Cp-Professional est un environnement de développement (OS/2, Unix, Microsoft). Il contient des méthodes de distribution des objets propriétaires avec l'intention de fournir des passerelles CORBA et OLE2.0/COM. L'offre est un exemple d'environnement distribué basé sur les notions de framework et de composant.

Les modèles de distribution des données ou des traitements sont :

- le modèle SQL qui, à travers le framework "SQL Data Access", permet d'accéder à des modèles relationnels ;

- le modèle d'appels distants qui, à travers le framework "Remote Object Call" permet d'implémenter le modèle client-serveur classique ;

- le modèle de notification qui, à travers le framework "Notification" permet d'implémenter un modèle événementiel 1-n ;

- le modèle "réunion" qui, à travers le framework "Caucus" devrait permettre d'implémenter des modèles de travail en groupe/ messagerie 1-n.

La manière de programmer sous CommonPoint dépend fortement de l'utilisation systématique des frameworks. Ceux-ci ont en effet, le contrôle effectif de l'essentiel du flot d'exécution. ("Don't call us, we'll call you). Le code ajouté est normalement client du framework par l'intermédiaire d'un mécanisme approprié (action, notification, héritage, polymorphisme).

Ce qui fait l'originalité du concept de framework, c'est l'étendue *des comportements par défaut* fournis par le système ainsi que la capacité du modèle à s'étendre hors des systèmes IHM grâce au polymorphisme et à l'héritage (dans des interfaces de programmation). Il produit également un support pour la réutilisation et l'implémentation de composants fonctionnels.

Il existe plusieurs catégories de frameworks : les frameworks Desktop *(document composite)* et les frameworks Systèmes. Les frameworks applicatifs (fonctionnels) prennent en charge la logique applicative et fournissent les outils pour construire des composants visuels. Ils sont orientés IHM, orientés tâches et ont une infrastructure d'utilisation établie pour la réutilisation de code et de conception (patterns design, patterns domaines, patterns métiers).

Un framework applicatif spécialisé offrira à terme le support nécessaire pour que des composants *puissent réagir au contexte* dans lequel ils sont réutilisés.

Les frameworks systèmes contiennent le middleware (NOS : Network Operating System) et les services OS. Il existe également quelques frameworks de communication. Les objets de base de l'architecture de l'outil sont des frameworks applicatifs, des frameworks du domaine de l'entreprise et des frameworks systèmes.

CommonPoint était une offre technologique distribuée. Dans le monde distribué, c'est la philosophie de l'outil qu'il faut retenir plus que l'échec commercial de Taligent et son absorption par IBM.

Son modèle *MPVSC* (Modèle de données Présentation Vue Sélection Commande) est lié au langage C++ :

- un modèle contient les données destinées à être traitées ;

- une vue est une manière de représenter les données du modèle. Il peut y avoir plusieurs vues pour un même modèle ;

- une présentation est chargée de la coordination des vues et du modèle. Elle a dans ce contexte une fonction de gestion des mises à jour ;

- une sélection est un objet qui sert à repérer l'instance du modèle qui est active. En faire un objet (plutôt qu'un pointeur comme dans OLE2.) a l'avantage de faciliter la distribution ;

- une commande est un objet destiné à encapsuler une demande de modification des données. En faire un objet permet d'accéder à *la notion de paramétrage* et de distribuer les commandes sur un réseau ou entre processus.

La présentation a un rôle de contrôle et de traduction des actions de l'utilisateur : elle gère les interfaces de saisie des actions et les transforme en paires Commande/Sélection appliquées à un modèle précis. Ce mécanisme permet de supporter les notions de collaboration, de script, d'annulation et de reprise sur erreur puisque toute action sur les données passe par des objets pouvant être archivés et diffusés (les commandes et les sélections).

Ce modèle fournit donc les niveaux d'indirection *(les pointeurs intelligents)* suffisants pour pouvoir spécifier des *commandes génériques* dans un domaine IHM. La notion de sélection facilite la gestion des objets distribués. Les standards d'interopérabilité qui seront supportés sont "Systems Requirements Process" de l'X/Open, Domf (implémentation HP de Corba) et Som/Dsom (System Object Management/Distributed System Object Management : implémentation IBM de Corba). *Ce modèle impose un certain style de conception pour lequel il n'existe pas actuellement de support méthodologique.*

3.2. Le besoin d'une nouvelle approche méthodologique

La construction d'application élaborée à partir de scénarios d'entreprise consistant à découper des processus en services et à les implémenter en frameworks est un changement culturel important dans la méthode de développement de logiciels. La manière de travailler change puisque le développeur doit rechercher à utiliser ce qui a été codé et testé et concevoir en fonction de la réutilisation *a priori*.

Les frameworks (métier, applicatif, système) sont réalisés à l'aide d'un processus de développement itératif et incrémental. C'est la modélisation guidée par la démarche "design pattern" ou moules de conception qui est mise en œuvre. *Le modèle est contenu dans le moule (forme qui se répète).*

Dans un développement d'applications distribuées, il ne faut pas hésiter à réaliser plusieurs maquettes jetables (modèles animés) afin de bien identifier les besoins de l'utilisateur. L' IHM est récupérée pour développer un prototype opérationnel qui valide le fonctionnement du modèle. Plusieurs versions sont développées. Le développement d'une autre application offrant une nouvelle vue de la version développée engendrera une ébauche de définition de framework. Le framework est identifié au départ et il faut voir à l'expérience si nous pouvons en généraliser l'usage au niveau entreprise,

interne à une application ou entre les applications.

C'est la démarche de développement que nous proposons au vu des premières applications développées à l'aide de l'outil CommonPoint. Les frameworks n'ont pas été modifiés, en revanche les frameworks desktop se sont enrichis.

Il faut au maximum standardiser les interfaces, ce qui implique de constituer des communautés qui travaillent dans le même sens. Toutes les communications entre applications sont prises en charge par les frameworks eux-mêmes.

Les outils de programmation visuelle assurent la création et la gestion des applications client-serveur en assemblant des briques logicielles préfabriquées.

CONCLUSION

L'architecture du système d'information est au service des métiers de l'entreprise. Cela signifie que l'approche méthodologique a changé de sens et que la nature du travail a évolué. La nécessité de s'abstraire pour définir des invariants de composants de système d'information provient de nos objectifs de réutilisabilité et d'interopérabilité.

De telles approches sont dirigées par le métier et non par le traitement de l'information. Il existe un point de rupture dans l'objectif à atteindre et dans les moyens à mettre en œuvre, un déplacement des responsabilités qui font qu'une entreprise doit prendre conscience que l'architecture du système d'information est une cellule à part entière.

Le système d'information s'architecture autour des acteurs externes et internes, des services et des événements survenus qui incarnent la réalité de la vie de l'entreprise. C'est un véritable renversement de perspective.

Avec l'explosion des activités du commerce électronique et avec Internet, *le système d'information* va se *modéliser vu du client comme contenu du modèle externe.* Ce modèle externe complété par la vision externe à l'entreprise, celle des activités des clients, va assurer la communication avec le système existant pour le système général d'information de l'entreprise (les clients, les contrats, les comptes, les engagements).

Cette approche permet de gérer les risques informatiques, les risques de marché, les risques de taux, les risques de crédit, les risques de contreparties, etc. en mettant en place une architecture de système information complète adaptée à la réalité et à la vie de l'entreprise.

Les nouvelles unités opérationnelles mises en place par l'entreprise ont alors vocation à profit interne ou externe.

L'entreprise gagne à filialiser ces unités de façon à faciliter les coopérations extérieures, en particulier aux niveaux national, européen et international.

BIBLIOGRAPHIE

Chartier, E. (1996), *Le re-engineering du système d'information de l'entreprise*, Economica.

Cotter, C. et Pottel, M. (1995), *Inside Taligent Technology*, Addison-Wesley. CommonPoint devait être livré avec deux outils de développement : cp-Constructor et cp-Professional.

LEXIQUE

ACTEUR

Un acteur est une personne qui prend une part déterminante dans une action. Un acteur est identifiable par la mission qu'il remplit au sein de l'organisation (acteur interne) ou à l'extérieur de celle-ci (acteur externe). Cette mission se concrétise par un ensemble de fonctions à exécuter. Une personne, un poste de travail, un service peuvent être considérés comme des acteurs, de même qu'un organisme extérieur (confrères, partenaires, etc.).

ARCHITECTURE GOBAN

Cette architecture s'appuie sur l'isomorphisme qui existe entre les perspectives ouvertes par l'explosion de nouveaux modèles d'architecture de réseaux de type Internet, qui créent un maillage planétaire entre les clients et les serveurs, et les stratégies utilisées par les joueurs du jeu de GO.

COMMONPOINT

On consultera l'ouvrage de référence de C. Cotter et M. Pottel : *"Inside Taligent Technology"*, Addison-Wesley - 1995. CommonPoint devait être livré avec deux outils de développement : cp-Constructor et cp-Professional.

cp-Constructor

cp-Constructor est un générateur d'interface homme machine (IHM). Contrairement à la plupart des produits proposés dans ce domaine, il ne génère pas de code à compiler : les structures de données correspondant au travail effectué sont directement sauvegardées sous la forme d'un document binaire. Ce document sauvegardé est utilisé par l'application au démarrage, lorsqu'elle doit régénérer son interface utilisateur. Ce mécanisme implique que l'IHM ne peut communiquer avec l'application qui l'utilise que par messages (actions) et notifications.

cp-Professional

Le livre *"Inside Taligent Technology"* le décrit comme un système englobant éditeur syntaxique, visualiseur de classes, gestionnaire de version, compilateur incrémental, et debugger. Taligent y annonce son intention d'exposer les interfaces des frameworks composant cp-Professional aux fins d'extension et de modification (support d'autres langages que le C++ notamment).

CYCLE DE VIE

Il se caractérise par plusieurs étapes qui se situent sur une échelle de temps. Le cycle commence lorsque l'organisation envisage ou décide de construire le SI. Il passe par différentes étapes de développement préalable à sa mise en place (Etude d'opportunité, Etude préalable, Conception et réalisation). Il connaît une période plus ou moins longue d'utilisation et de maintenance, et se termine lorsque l'organisation juge qu'il est devenu obsolète ou inadéquat.

Au cours du temps, l'organisation est amenée à faire évoluer le système d'information. Il en résulte de difficiles périodes de coexistence et de migration. La littérature anglo-saxonne parle de "Legacy System".

CYCLE D'ABSTRACTION

Il est caractérisé par trois niveaux :

- le niveau conceptuel, qui décrit l'activité et les objectifs de l'organisation sans se soucier des types de ressources utilisées ;

- le niveau organisationnel, qui permet de spécifier l'affectation des ressources, le degré d'automatisation...

- le niveau opérationnel, qui permet l'adaptation du niveau organisationnel à une situation physique déterminée (volume, choix techniques, ...).

ÉVÉNEMENT DECLENCHEUR

C'est un stimulus ou un événement externe qui initie le déclenchement d'une activité ou change l'état d'un objet ou d'une donnée. Les déclencheurs sont définis dans le modèle sous forme d'une hiérarchie permettant de classifier les différents types de déclencheurs. Par exemple : 'Fin de journée' est un déclencheur du groupe 'Fin de période' qui fait partie du type 'Déclencheur de type Temps'.

Un événement est un fait actif vis-à-vis du système d'information, c'est-à-dire susceptible d'en modifier les

composants internes. Il indique que quelque chose s'est passé et que, en conséquence, le système d'information doit réagir. On peut distinguer deux types d'événements :

- des événements externes, qui proviennent de l'environnement ou qui se produisent dans l'univers d'intérêt. Il y a donc franchissement de la frontière du système d'information. Par exemple : 'Demande de chéquier par le client' ;

- des événements internes, qui proviennent de la réalisation d'une condition mettant en œuvre des informations contenues dans le système d'information (par exemple l'événement résultat d'un contrôle de données de saisie). Ces événements ne sont intéressants que s'ils sont suivis d'une nouvelle réaction du système d'information, ou s'ils sont la cause de l'émission d'un message vers l'environnement.

FLUX D'INFORMATION

Un flux indique un ensemble de messages représentant une communication entre deux partenaires au moins : il a une origine et au moins une destination.

FRAMEWORK

Un framework est un ensemble de briques logicielles préfabriquées que l'utilisateur peut utiliser, étendre ou personnaliser en fonction de son cas spécifique, mais qui reste néanmoins "encapsulable" dans le niveau de généricité du framework.

Avec le framework, le développeur ne part plus de rien chaque fois qu'il construit son application. Le framework est en général documenté, et la démarche à suivre pour l'utiliser est décrite par un "pattern" (voir ce mot).

MODÈLE CONCEPTUEL

C'est une représentation du réel perçu en termes de liens sémantiques (domaine du "Quoi"). Ces liens sont les idées et les ponts de dénomination, avec les contraintes entre toutes ces phrases (idées et ponts), c'est-à-dire les conditions qu'une phrase doit remplir pour pouvoir assurer la cohérence de la base d'information.

MODÈLE EXTERNE

C'est le domaine de visibilité du système d'information par le client (par exemple une fenêtre). Cette vue externe comprend la description des informations requises par l'utilisateur et des chemins d'accès à ces informations (c'est-à-dire aussi la manière de les voir). Les utilisateurs ne s'intéressent pas aux mêmes types d'objet, ne désignent pas nécessairement le même objet de la même façon, ne perçoivent pas le même objet de la réalité de façon identique. C'est pourquoi il y a plusieurs modèles externes ou vues. Le modèle conceptuel est un modèle central qui est le sur-ensemble de tous les modèles externes.

MODÈLE INTERNE

C'est une représentation physique du système d'information (domaine du "Comment ranger"). Le modèle interne pré-définit la représentation physique de chaque état autorisé (par le modèle conceptuel) de la base d'information. A ce niveau, on parle de la façon dont les informations sont rangées sous forme d'enregistrements, et on s'intéresse aux chemins d'accès et aux structures de pointeurs.

PATTERN

Expertise et savoir-faire nécessaires pour la résolution d'un problème ou d'une technologie dans une approche problème-solution, qui permettent de gagner du temps face à certains cas de conception ou de réalisation. C'est le "handbook" de l'architecte ou du chef de projet. On sait que les méthodologies classiques ne donnent jamais la solution d'un problème. En particulier, le pattern permet de documenter l'utilisation d'un framework.

PROCESSUS

Ce concept représente la partie dynamique du système d'information. Un processus est l'ensemble des chemins possibles permettant de passer d'une famille d'événements aux résultats correspondants. Il correspond à une activité dont la progression peut être représentée, à des points discrets dans le temps, par son état. Les états possibles d'un processus sont l'attente, l'état actif et l'état terminé.