Panorama des Technologies de l'Information et de la Communication

État major des armées – ENST

Synthèse sur les Architectures des Systèmes d'Information

Jean-Marie LAPEYRE
14 octobre 2005

Version 4.0 2005-10-02T1150

- Cette présentation s'inspire d'une expérience de pilotage d'un programme de refonte complète d'un grand système de gestion : le **programme Copernic** de refonte du système d'information fiscal.

- Elle suppose que l'organisation reconnaît sa dépendance au SI et souhaite en rester maître.

- À relativiser dans le cas d'une rénovation partielle (insertion d'un « front-end » sur la base d'un système de gestion inchangé) ou d'une externalisation lourde du SI.

copernic

DGI
DGCP

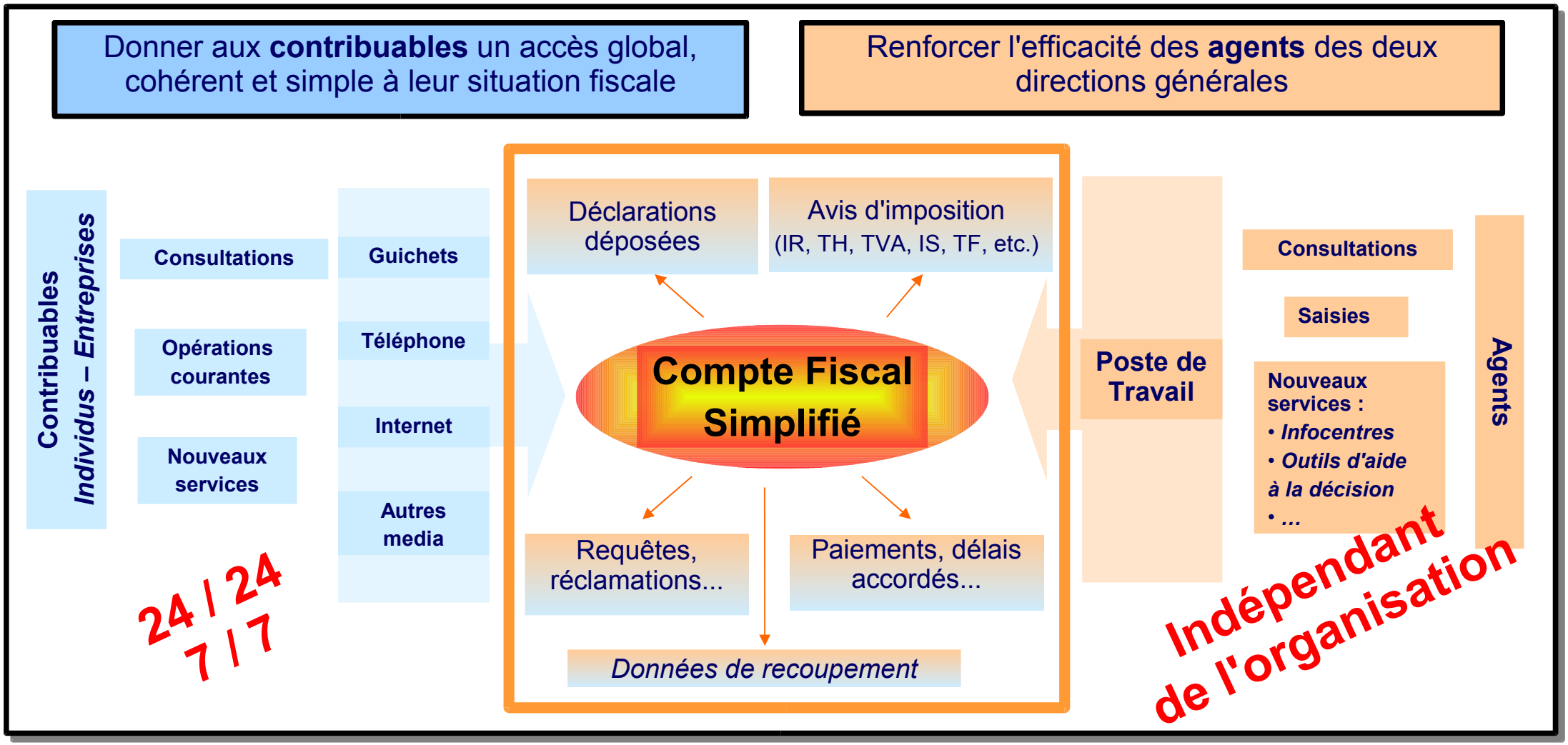
Jean-Marie LAPEYRE

Directeur technique du programme Copernic

- Curriculum Vitae :
  - http://jml.lapeyre-s.net/CV
- Contact par mèl. :
  - Personnel (pour le cours) : enst@lapeyre-s.net
  - Professionnel : jean-marie.lapeyre@dgi.finances.gouv.fr

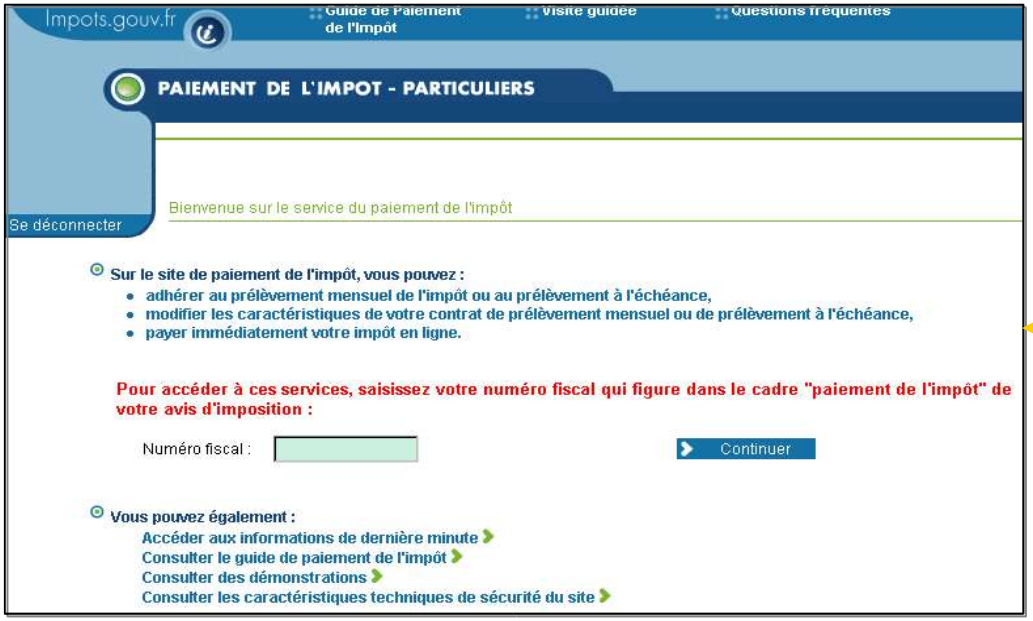
COPERNIC : Enjeux

- **Un système d'information unique pour les deux directions générales en charge de la fiscalité directe (DGI, DGCP) et rendu accessible aux citoyens**
- **1 milliard d'€ de budget complémentaire sur ~ 10 ans (2001-2009) et ~ 60 projets**



COPERNIC : Exemples

- **Télé-déclaration et consultation IR**
  - ~ 3 800 000 télé-déclarations en 2005
  - ~ 120k en 2002,~ 600k en 2003, ~ 1250k en 2004



- **Paiements en ligne**
  Impôt sur le revenu, Taxes d'habitation et Taxes foncières, CSG

- **Adhésions et modification en ligne**
  montants des prélèvement, changement de situation...

Plan

- **Quelle stratégie pour un système d'information ?**
  - Constats, Objectifs et Démarche ;
  - Paradigme d'architecture ;
  - Normes et standards ;
  - Logiciels libres.

- Mise en oeuvre opérationnelle
  - Architecture des applications ;
  - Méthodologie de conception ;
  - Choix techniques ;
  - Modalités de contrôle.

- Constat 1 :

  Le coût de conception et de réalisation d'un projet de SI est faible au regard du cumul des coûts totaux d'exploitation, de maintenance (corrective, adaptative, évolutive), et de retrait de service.

  Le Gartner estime qu'il représente en moyenne 20% du « TCO ». Le ratio est aggravé dans un contexte d'évolution permanente et non aisément contrôlable (par ex. secteur très dépendant de mesures législatives récurrentes – loi de finance).

  Ce facteur est d'autant plus ignoré qu'il ne représente pas une difficulté immédiate de projet... Pourtant, les mesures correctrices efficaces sont inapplicables au delà de la phase de conception.

- Constat 2 :

  Les outils logiciels « propriétaires » ont une durée de vie très courte, la compatibilité croissante est de fait rare (stratégie des éditeurs tentant de ménager des sources de financement récurrentes) et le coût de l'obsolescence est très élevé.

  - Exemple 1 :
    évolution incontrôlée des formats de données en bureautique notamment (et contre-coup sur les applications qui y sont interfacées) ;
  - Exemple 2 :
    un atelier de génie logiciel a une durée de vie moyenne de 18 mois.

- Constat 3 :

  L'éparpillement fonctionnel et technique est une source importante de charge et de non-qualité

  - Exemple fonctionnel :
    multiplication d'implantations d'un même ensemble cohérent de règles de gestion (dans un ESB/EAI et dans le « legacy » p.ex) ;
  - Exemple technique 1 :
    gestion dans des applications micro-informatiques dispersées peu contrôlées
  - Exemple technique 2 :
    exploitations différenciées dans plusieurs centres de production.

- Objectifs possibles d'une stratégie d'un système informatique :
  - **Maîtrise** du SI
    ou être apte à rapidement adapter le système à des besoins fluctuants tout en assurant la qualité du service rendu ;
  - **Pérennité** du SI
    ou faire en sorte que les choix d'aujourd'hui nous permettent de maîtriser le système demain ;
  - **Indépendance** aux technologies et aux fournisseurs.
- Démarche de mise en oeuvre :
  - Paradigme d'architecture fort (« SOA ») ;
  - Adoption systématique de normes et standards ;
  - Utilisation rationnelle de logiciels libres.

- La robustesse d'un système d'information est avant tout liée à la qualité du paradigme de construction sur lequel il repose.

  Un modèle très puissant est aujourd'hui accessible compte tenu des technologies disponibles :

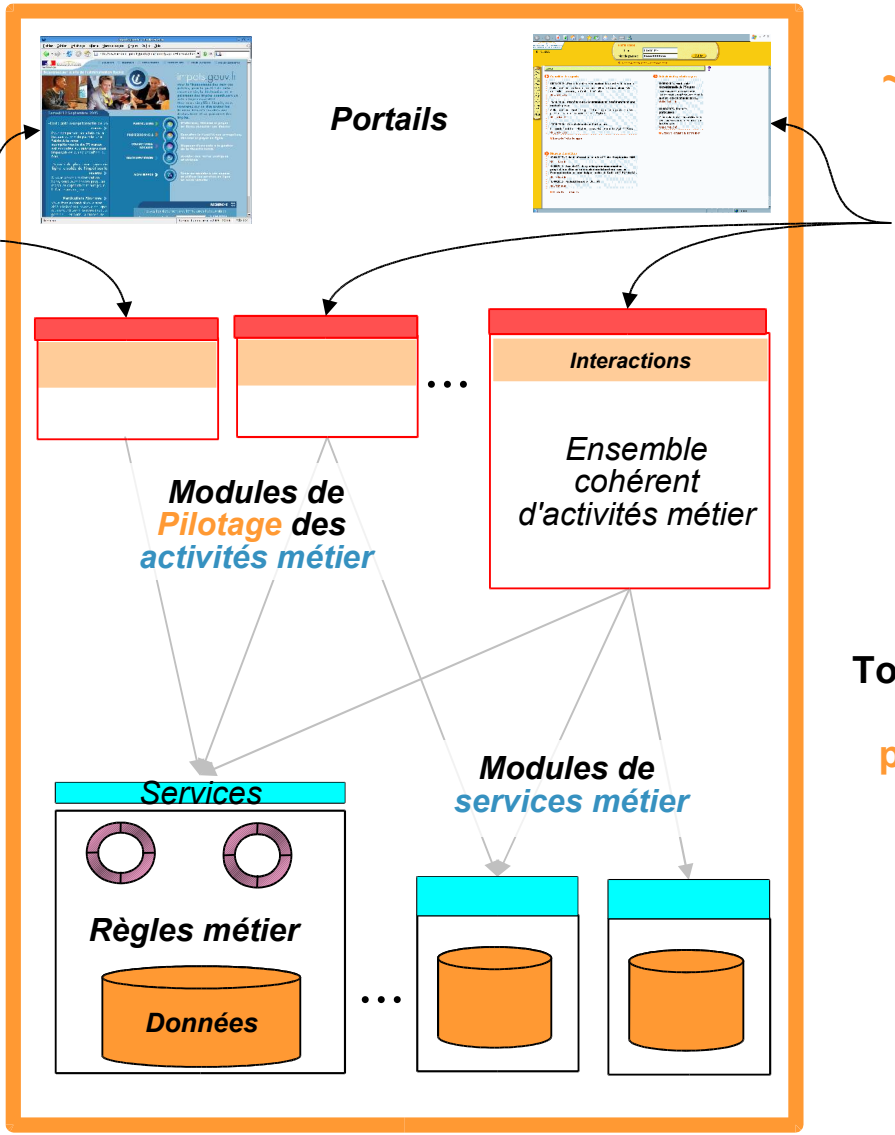
L'Architecture Orientée Service (SOA)

- Le modèle, dans son acception la plus pure (telle que mise en oeuvre dans Copernic), repose sur la distinction, dans un système d'information, des « processus métiers » et des « services et données métiers élémentaires ».

  Notions séparées car subissant des contraintes d'évolution souvent non convergentes (la fragilité des modèles antérieurs pouvant être expliquée par le fait de lier les deux notions dans une même entité : l'application).

Stratégie
Paradigme d'Architecture (II)

~ *55M citoyens*
~ *3M entreprises*



~100K agents

Les processus métier et l'organisation peuvent évoluer indépendamment des services métiers élémentaires et des données

Les données métier peuvent toujours rester cohérentes et peuvent être atteintes par n'importe quel processus à tout instant

Pré-requis impératif :

Toute interaction doit être réalisée selon un protocole normalisé et standard

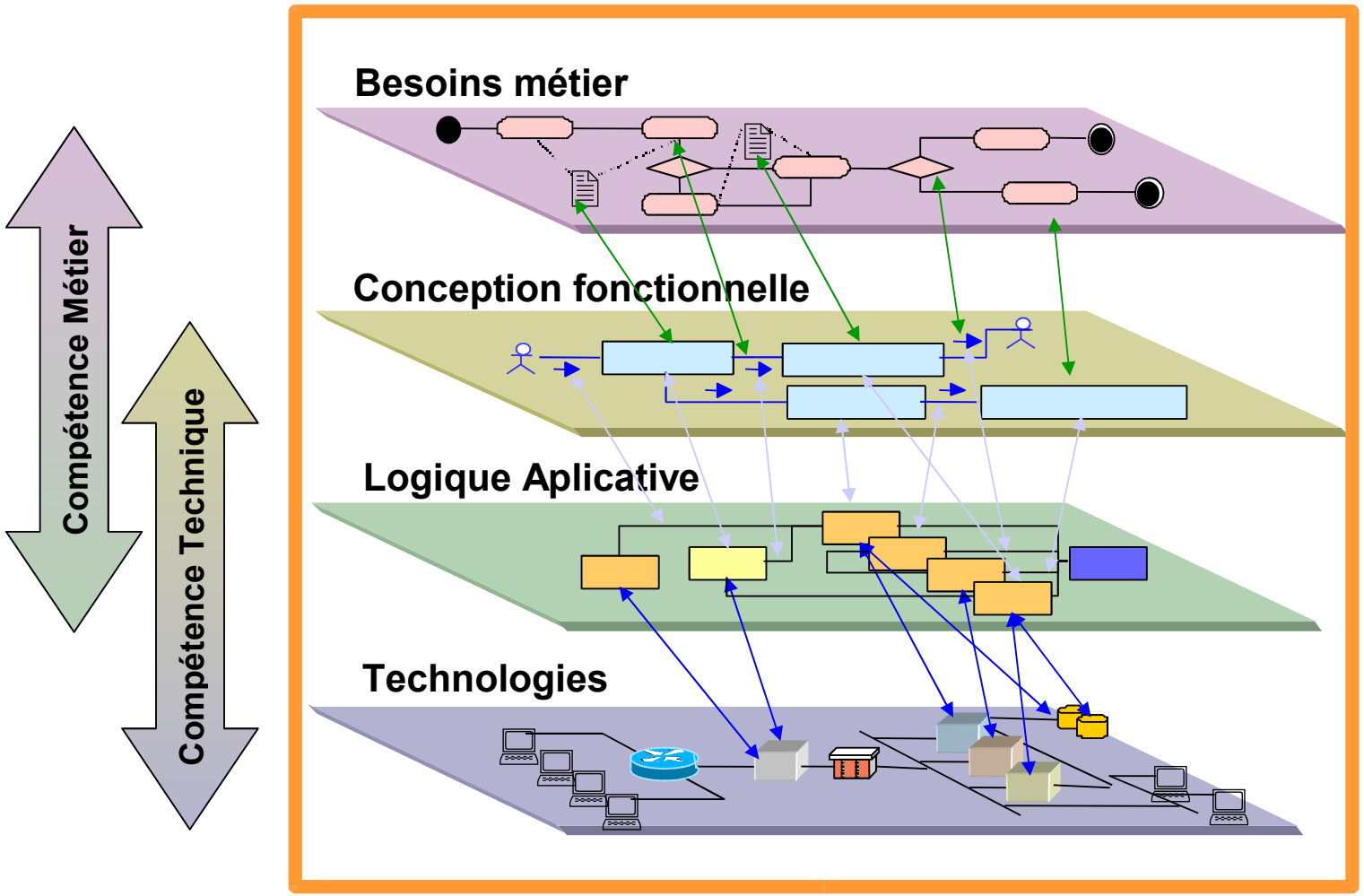
- Le choix et le maintien d'un paradigme d'architecture vise à diminuer la complexité de maintenance et abaisser les coûts opérationnels de gestion (facteurs majoritaires du coût total de possession « TCO »).

- Sa mise en oeuvre ne peut être effective qu'à condition de clarifier (voire formaliser) le rôle des acteurs dans la chaîne de conception / réalisation / maintenance / certification / exploitation.

  Le modèle SOA, outre un paradigme d'architecture, porte un modèle d'organisation du travail au sein des organisations en charge des systèmes d'information.

Paradigme d'Architecture (IV)

Le système d'information peut-être appréhendé selon différentes **vues** selon la compétence et la responsabilité des acteurs.



Vue métier

Description donnée pour / par les utilisateurs

Vue fonctionnelle

Conception fonctionnelle par **cas d'utilisation** et **exigences**

Vue applicative

Conception logicielle avec une méthodologie **orientée objet**

Vue Technique

Services Web (sans ESB) sur clusters Linux...

- Les définitions fonctionnelles, applicatives, et techniques doivent s'articuler autour de concepts communs.
  - Par exemple, le choix d'une conception orientée objet ou le paradigme SOA lui-même.

- Les organismes internationaux de normalisation sont les vecteurs de réflexions structurelles sur la constitution des systèmes d'information.

  Ils incarnent sous forme de spécifications ouvertes les concepts fondamentaux qui sous-tendent les SI.

  Celles-ci sont souvent très stables et exemptes des travers qui peuvent être introduits par certaines ambitions monopolistiques des acteurs du domaine.

- Une norme est une spécification ouverte, conçue et approuvée sous le contrôle d'un organisme de normalisation.
  Un standard est une technique largement employée.

  La notion anglaise de *standard* fait la confusion entre les deux concepts.

- Les organismes de normalisation peuvent être classés en fonction de la liberté d'accès à leur production et de la capacité d'un expert à influer sur le processus de définition (selon la conviction que plus ouvert est le processus, plus la norme est conceptuellement et techniquement solide – notion de *peer's review*).
  - En tête, l'IETF, le W3C, l'OMG ;
  - Plus loin, l'ISO, l'UIT et les groupes « constructeurs ».

Normes et standards (III) – Références

- Internet Engineering Task Force (IETF)
  - http://www.ietf.org
  - Page des RFC : http://www.rfc-editor.org
- World Wide Web Consortium (W3C)
  - http://www.w3c.org
  - Voir en particulier http://www.w3.org/2001/tag
- Object Management Group
  - http://www.omg.org
- Java & Java Community Process
  - http://java.sun.com & http://www.jcp.org

- Les logiciels libres sont un moyen de maintenir les objectifs techniques :
  - Conformité aux normes ⇒ solutions de référence
  - Modularité ⇒ adaptables aux besoins
  - Accès non exclusif ⇒ corrections rapides
  - Flexibles et adaptables ⇒ possibilités d'expérimentation
  - Pas de coût d'entrée ⇒ pas besoin de contrat public

Logiciels Libres (II) – cas de l'administration fiscale

Excellent respect
des normes et standards et
grande flexibilité d'utilisation

Expériences de mise en oeuvre
très positives

Preuve d'un coût de possession
très faible (économie d'au moins 90%)

Support mature sur le marché

2000

2004

Option d'utilisation ouverte

- ~ 4000 serveurs linux déployés
- Administration Système entièrement basée sur du logiciel libre (Nagios, MRTG)
- Infrastructure production basée sur du logiciel libre (Apache, JBoss)
- Plate-forme de développement basée sur du logiciel libre (Eclipse)

Choix stratégique :

Le logiciel libre est désormais le choix par défaut pour toutes les applications fiscales

Logiciels Libres (III) – Références

- Définition des logiciels libres :
  - http://www.fsf.org & http://www.opensource.org
- Fondations et sites de référence :
  - http://www.osdn.com & http://www.tigris.org
  - http://lwn.net & http://slashdot.org
  - http://www.toolinux.com & http://linuxfr.org
- Répertoires de logiciels libres :
  - http://freshmeat.net
  - http://sourceforge.net & http://savannah.gnu.org

Plan

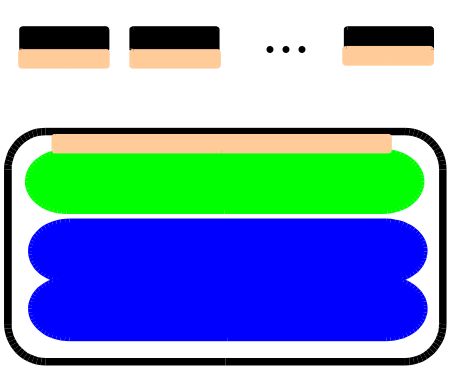
- Quelle stratégie pour un système d'information ?
  - Constats, Objectifs et Démarche ;
  - Paradigme d'architecture ;
  - Normes et standards ;
  - Logiciels libres.

- Mise en oeuvre opérationnelle
  - Architecture des applications ;
  - Méthodologie de conception ;
  - Choix techniques ;
  - Modalités de contrôle.

- (Rappel)
  Depuis les débuts de l'informatique de gestion plusieurs modèles applicatifs se sont succédés :
  - *Mainframes* (batch ou transactionnel, terminaux passifs) ;
  - Client/Serveur (client « lourd » spécifique) ;
  - Micro-Informatique (avec éventuellement SGBD centralisé) ;
  - *Intranet* (client/serveur avec client normalisé « universel »).

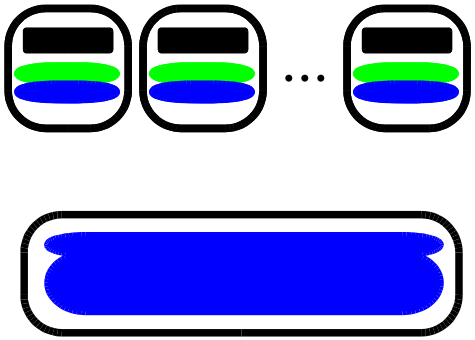
- Le retour marqué aux architectures centralisées relève de l'augmentation importante de la puissance des réseaux (à coût constant) et la maturité de sémantiques applicatives génériques (Web).

Architectures applicatives / Rappel Historique – Illustration



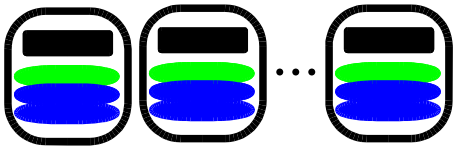
Mainframes

Clients légers
Concentration appli.
Standards de fait



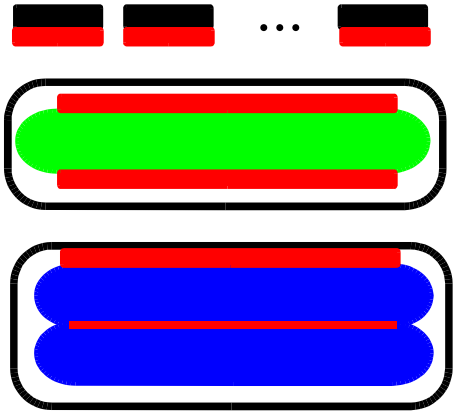
Client/Serveur

Clients spécifiques et complexes
Concentration données



Micro

Clients ultra-spécifiques
Pas de centralisation



Mode Web

Clients universels
Concentration appli.
Modularité
Interfaces normalisées

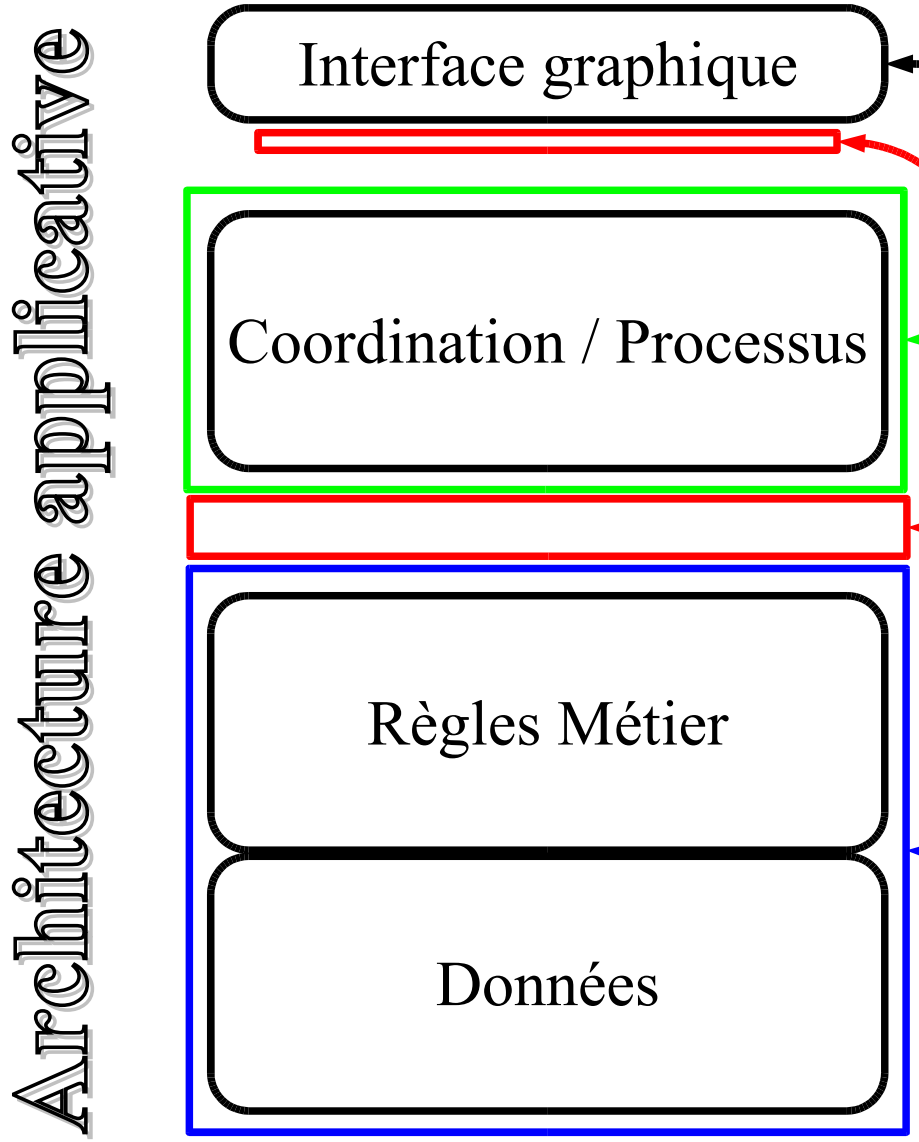
Légende :

- *Couche applicative / process*
- *Données et règles*
- *Interface homme-machine*
- *Protocole normalisé*
- *Protocole standard*

Architecture : Modèle en couches (I)

- Le modèle en couche (*n-tier*) segmente une application selon une architecture logique générique qui agrège les fonctions de gestion des données (couche « persistance »), les fonctions de pilotage des processus métiers (couche « application ») et les fonctions de gestion de l'interface usager (couche « présentation »). Ce modèle est le fondement du paradigme SOA.
- L'objectif est d'identifier les ensembles de fonctions relevant des mêmes contraintes de fonctionnement et d'évolution afin d'éviter les couplages intra-applicatifs inutiles.
- Les notions de « service » et d'interface sont au coeur de ce modèle qui transpose au niveau macroscopique les notions fondamentales de la conception orientée objet.

Architecture : Modèle en couches – Illustration (I)



Poste de travail
Navigateur (client universel)

Serveur d'application « à état »
Servlets + Session Beans

Couches d'accès normalisés
XHTML over HTTP pour le poste
SOAP ou RMI/IIOP en inter-appli

Silo de données
Règles de gestion « sans état »
Entity Beans
et
Stockage
SGBDR ou LDAP

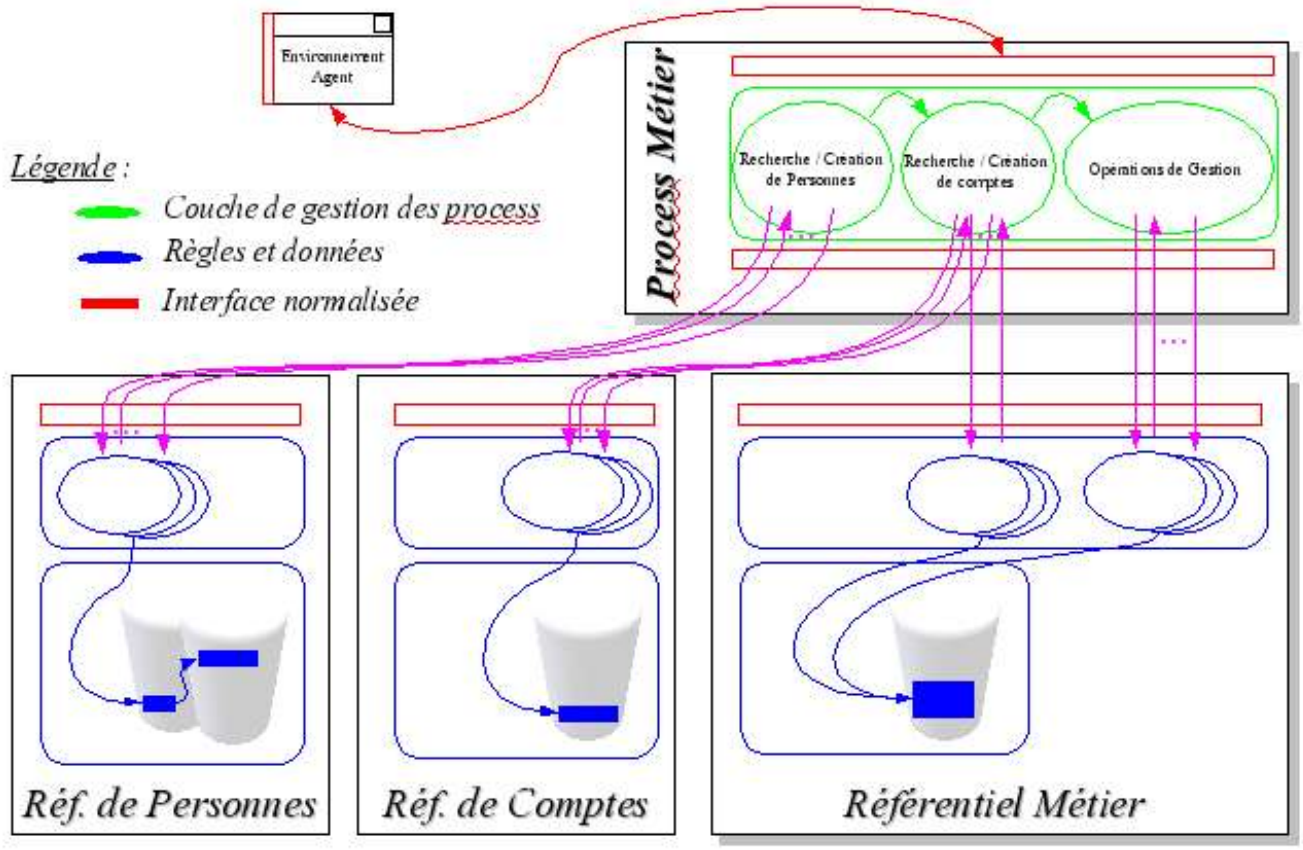
Architecture : Modèle en couches – Illustration (II)

Le pilotage centralisé des processus

Un module applicatif est défini, soit comme un ensemble cohérent de processus, soit comme un un ensemble cohérent de services de gestion élémentaires (souvent lié à des données).

Tout ensemble de règles de gestion inter-dépendantes doivent vivre dans la même application, condition fondamentale pour assurer la maintenabilité du système.

Dit autrement, minimiser les éléments intersticiels (danger lié aux outils dits de *Workflow* ou *EAI* ou *ESB*)

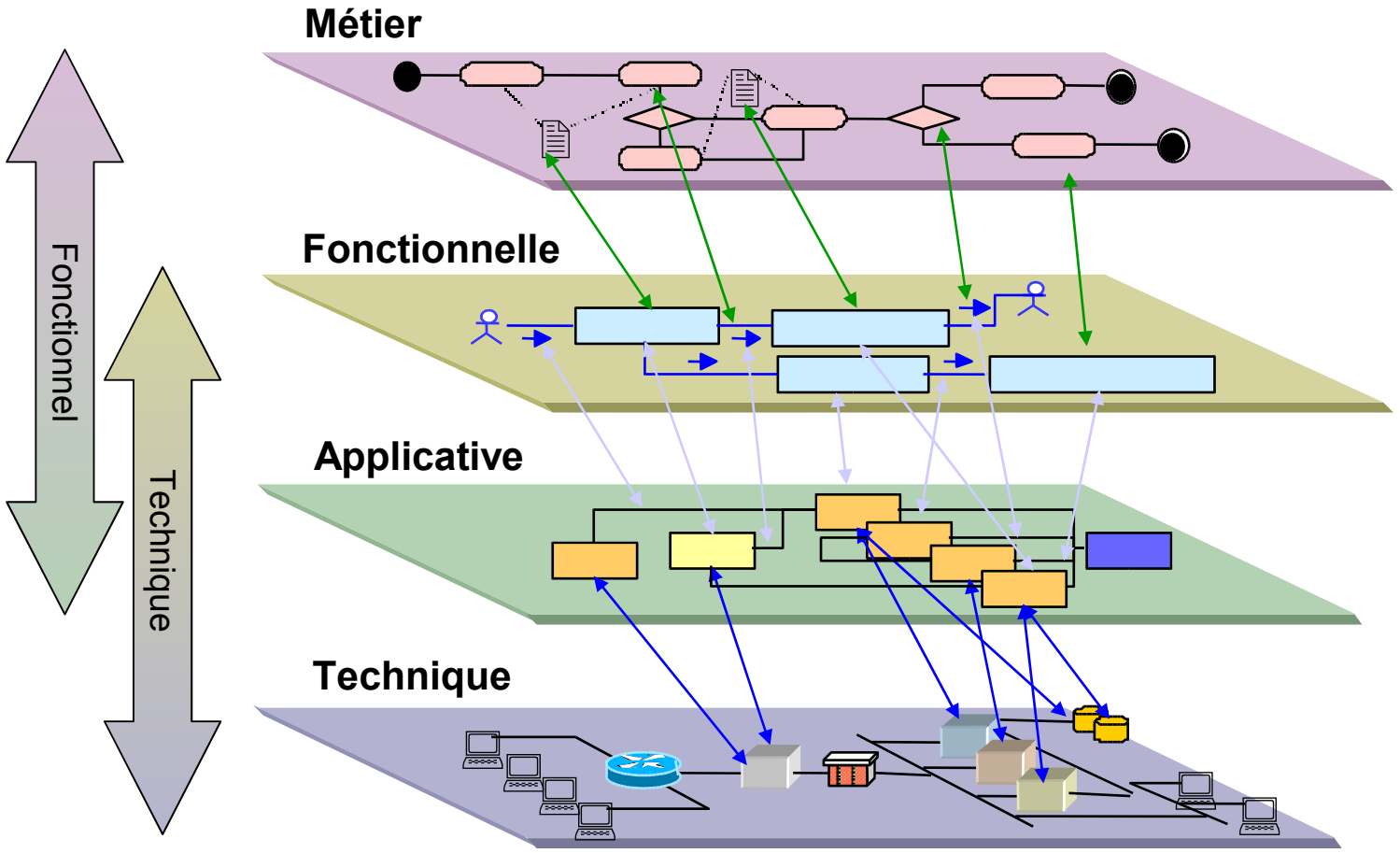


Architecture : Modèle en couches (II)

- Principal inconvénient du modèle : un « coût d'entrée » supérieur (difficulté de conception accrue, niveau technique à la mise en oeuvre supérieur, intégration plus sensible, performances moindres).
- Ses avantages : impacts moindres des évolutions fonctionnelles, souplesse du déploiement, robustesse (adaptable à la charge – *scalability*)
  - Ex : la mise en place de *clusters* de haute disponibilité et de partage de charge n'oblige pas à reconcevoir une application.

- La phase de conception d'une application (hors réalisation) repose sur une méthodologie et un langage de formalisation des spécifications. Ils assurent le lien entre MOA et MOE.
  - Historiquement en France, Merise, méthode complète proposant une formalisation, est très utilisé.
  - Toutefois, Merise ne conduit pas naturellement à expliciter les inter-dépendances intra-applicatives et donc complexifie le processus de maintenance évolutive (et confond le possible et l'opportun).
  - L'introduction des concepts objets a permis de palier le problème. Aujourd'hui, l'attention se focalise autour de RUP et d'UML

Mise en oeuvre
Conception : les vues sur le SI



Description du métier compréhensible par les acteurs du métier

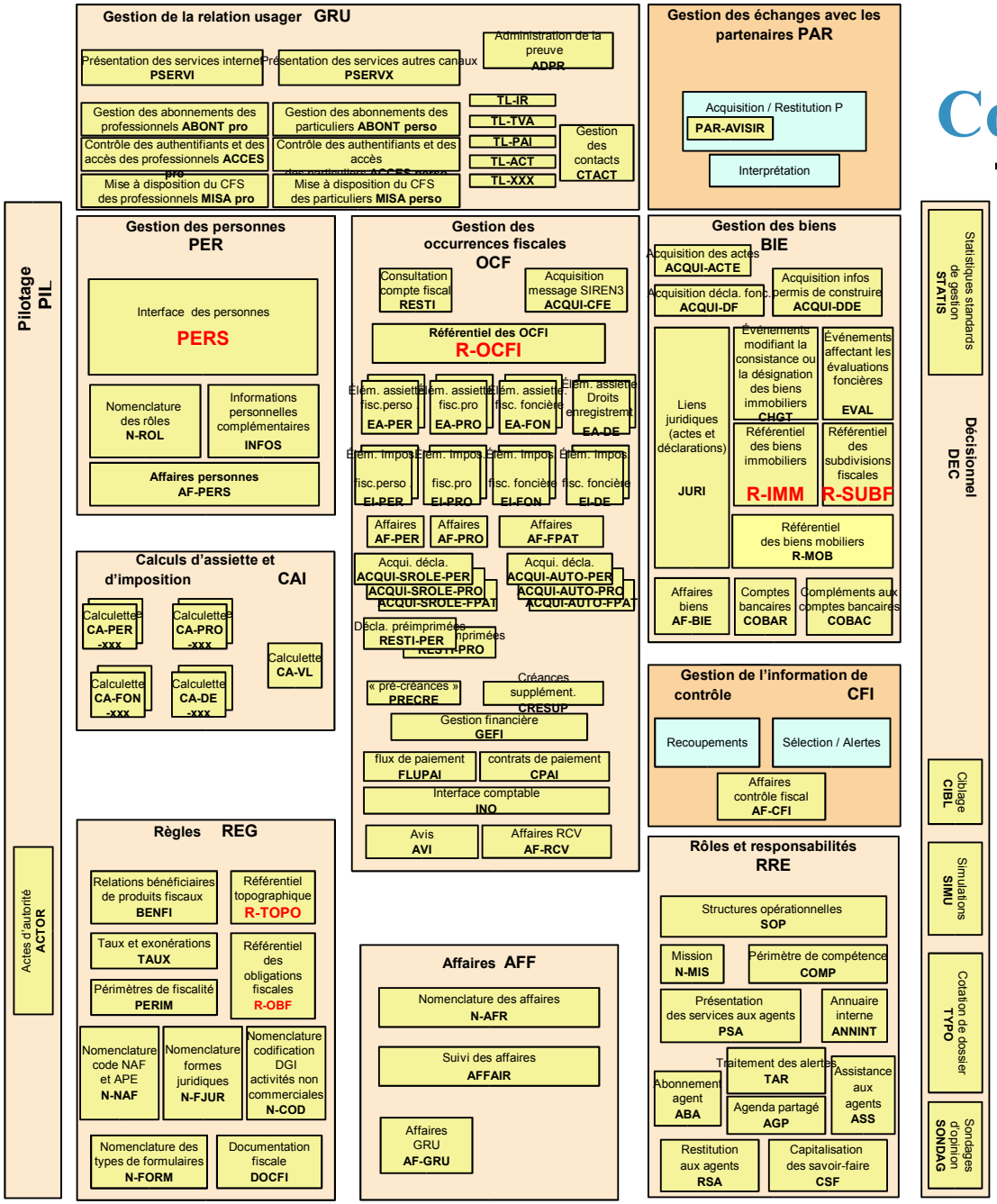
Description et structuration fonctionnelle du système d'information (services)

Description et structuration du système informatique en éléments logiciels (implémentation des services)

Infrastructure de fonctionnement du système informatique et déploiement des éléments logiciels

Les relations entre les différentes vues répondent à des règles identifiées de cohérence 2 à 2 et ne correspondent pas à des règles de hiérarchie.

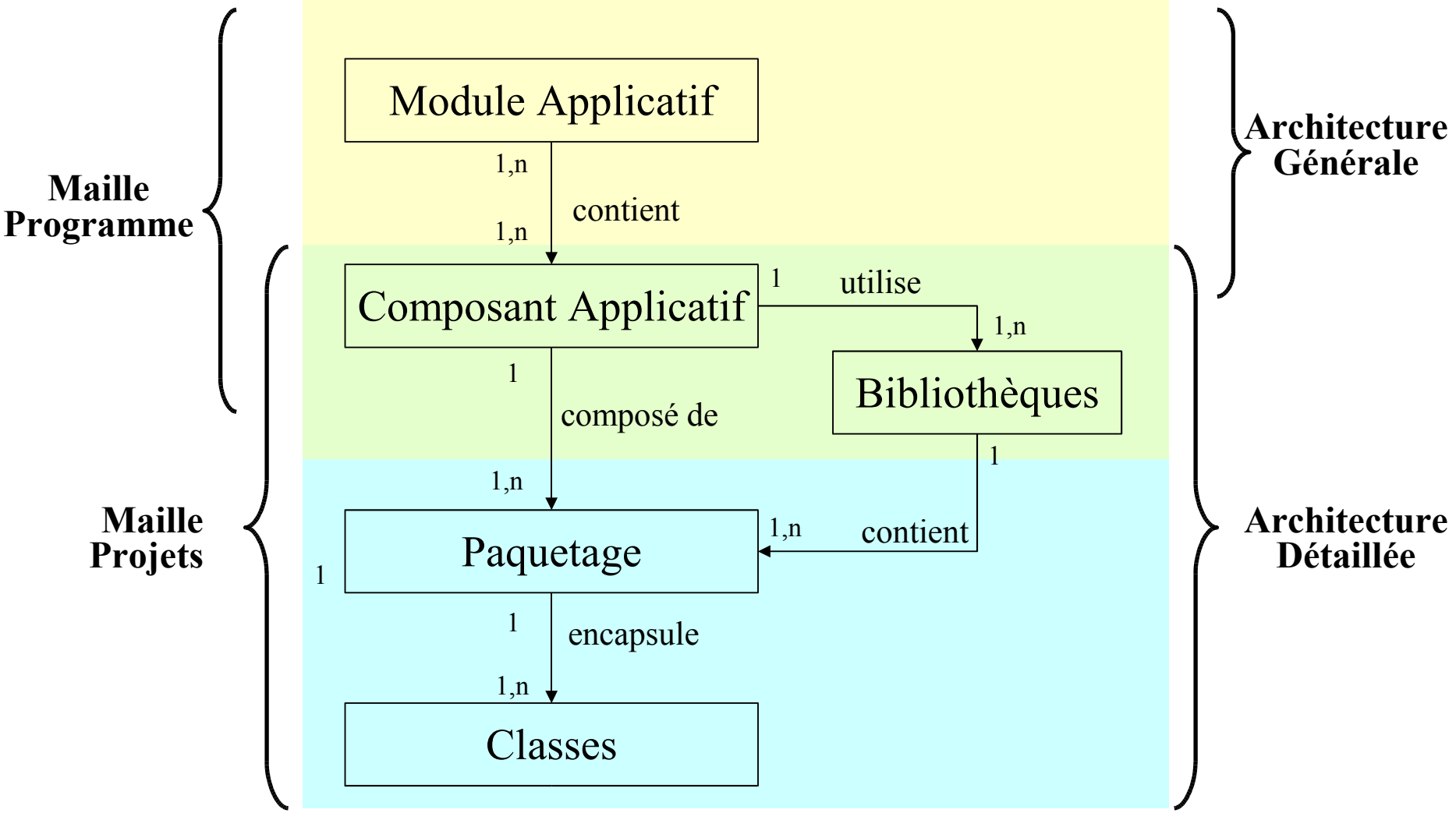
Mise en oeuvre
Conception : vue fonctionnelle



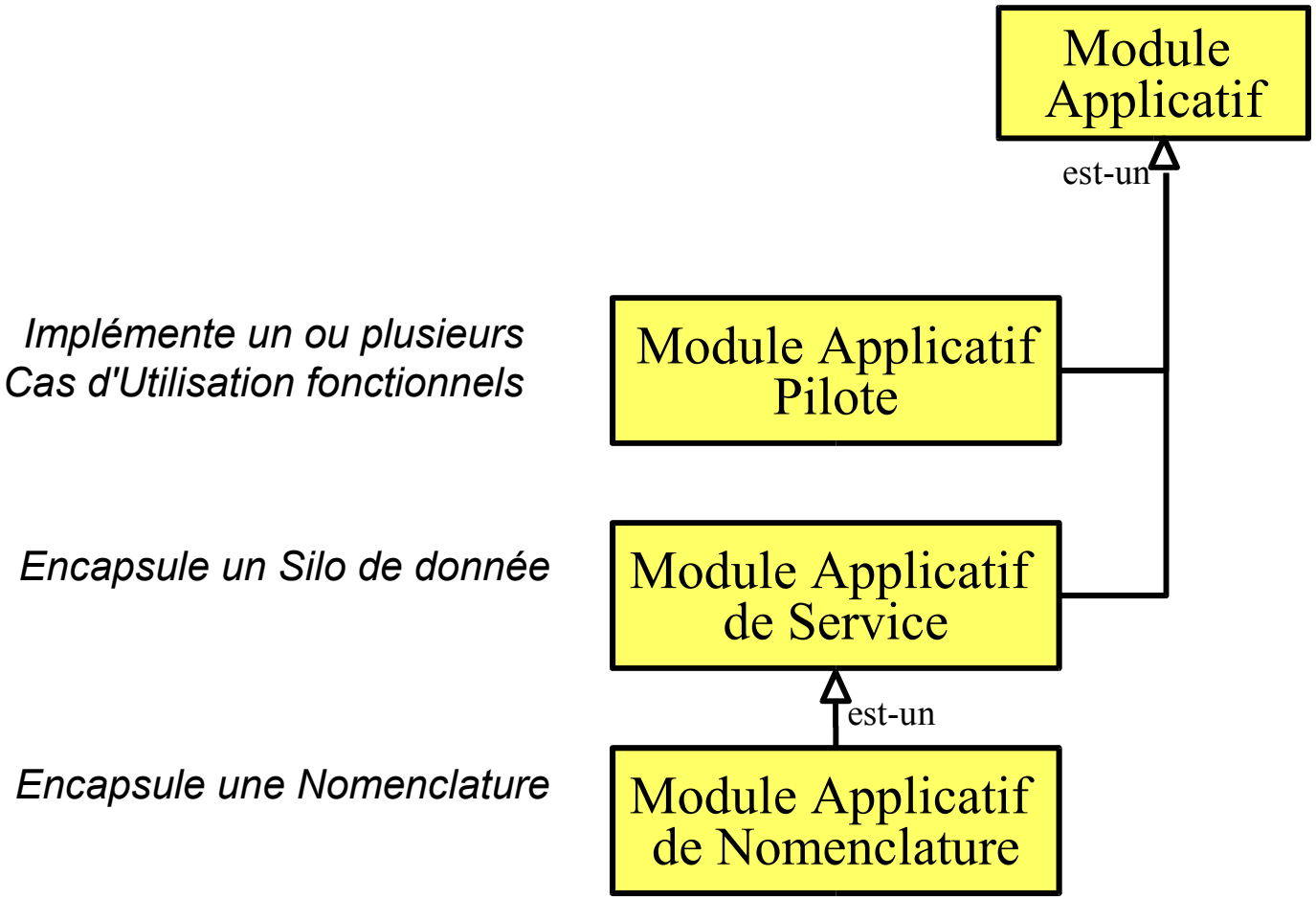
Les Services Fonctionnels :

une urbanisation en ensembles de fonctions

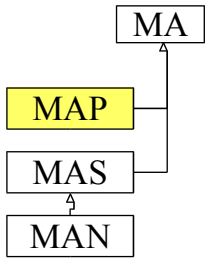
Conception : constitution de la vue applicative



Conception : les modules applicatifs

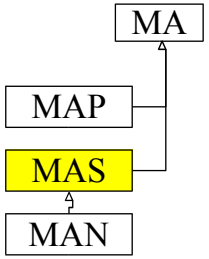


Conception : Module Applicatif de Pilotage



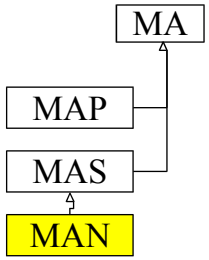
- Il implémente, de bout en bout, un ou plusieurs cas d'utilisation de la vue fonctionnelle
- Il est le point d'entrée pour l'utilisateur et
  Il est le pilote des autres modules applicatifs.
- Il est garant du respect des portées transactionnelles définies dans la vue fonctionnelle
- Il peut (et doit) stocker des données à fin de traçabilité et de reprise après pannes ; il ne stocke pas de données métier.
- Un MAP peut être de type interactif ou de type batch.

Conception : Module Applicatif de Service



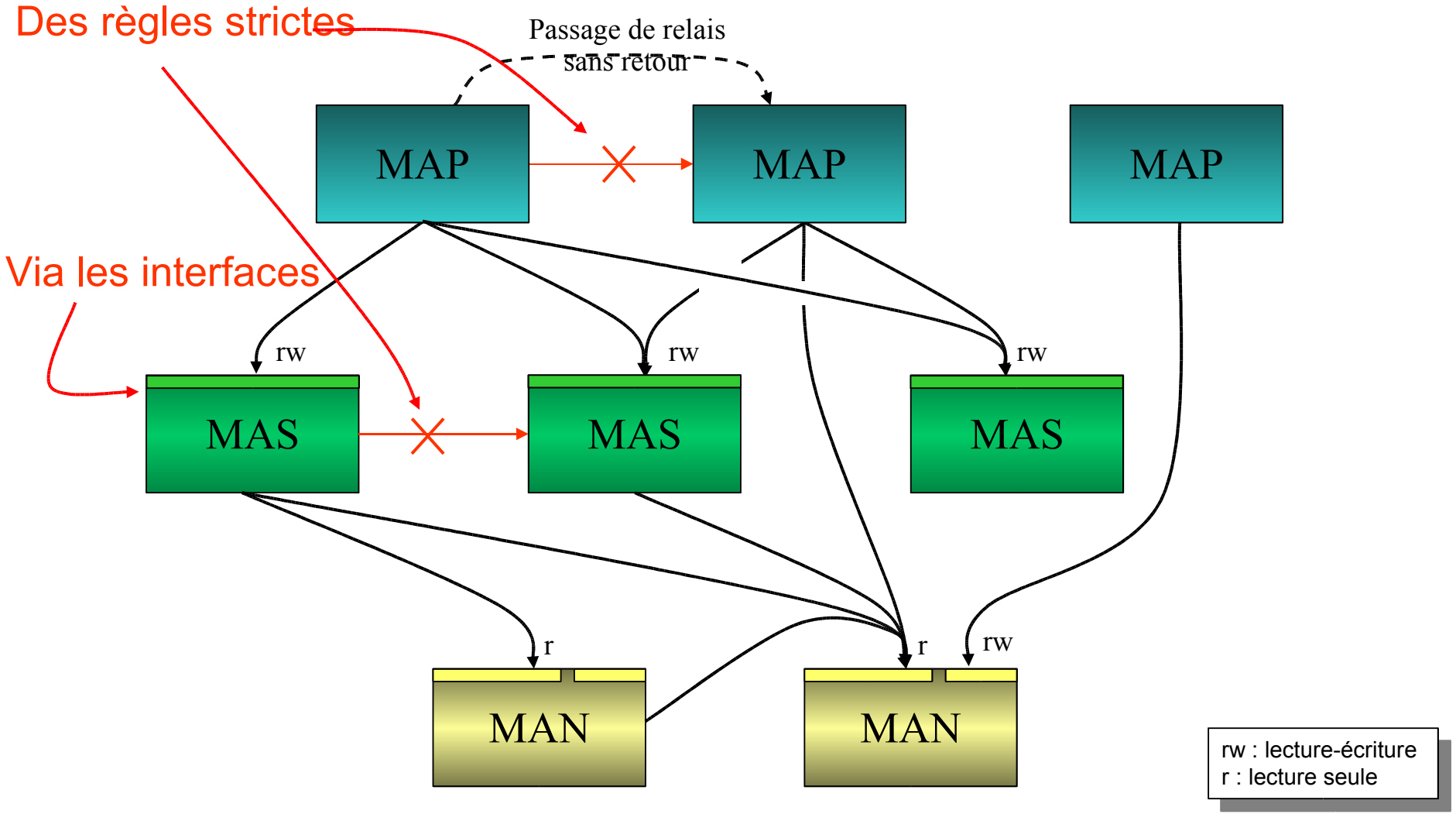
- Il encapsule un ensemble cohérent de données fonctionnelles (le silo) et fournit les services nécessaires à la gestion de ces données ; ensemble dont il est garant aux niveaux fonctionnel et technique
- Les modèles de donnée de deux MAS sont nécessairement disjoints (principe d'unicité de la localisation d'une donnée). Mais un MAS peut stocker des identifiants informatiques exportés par un autre MAS.
- Il offre des services unitaires sans état et atomiques :
  - sans état : le module traite chaque appel de service de manière indépendante sans relations avec les précédents appels, uniquement en fonction de ses paramètres,
  - atomique : un appel de service est soit complètement exécuté soit pas du tout exécuté ; il conduit toujours à un état fonctionnellement et techniquement cohérent des données du silo.
- Aspects non fonctionnels à prendre en compte durant la phase de conception – par exemple:
  - Performance : canal interactif et canal batch
  - Résilience : services répétables

Conception : Module Applicatif de Nomenclature

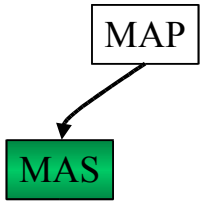


- MAS dont le schéma de données associé est une nomenclature : données stables très majoritairement accédées en lecture.
- Deux catégories de services
  - Accès en lecture seule - accessible à tous
  - Gestion des données de nomenclature, utilisée par le (ou les) MAP implantant les cas d'utilisation de gestion fonctionnelle de ces données.

Conception : Interaction entre modules applicatifs



Conception : interface d'un MAS (I)



L'interface d'un MAS (ou un MAN) est définie par :

- Liste des services exportés (= offerts)
- Sémantique de l'interface (= de l'ensemble de ces services)
- Univers du Discours

Conception : Univers du discours ?

Pour pouvoir interagir, deux modules ont besoin d'une interprétation commune des informations qu'ils utilisent.

Exemple 1

Camion :

Mission = faire partir livrer

Annulation = rappeler - *rentrer au garage*

Missile :

Mission = faire partir livrer

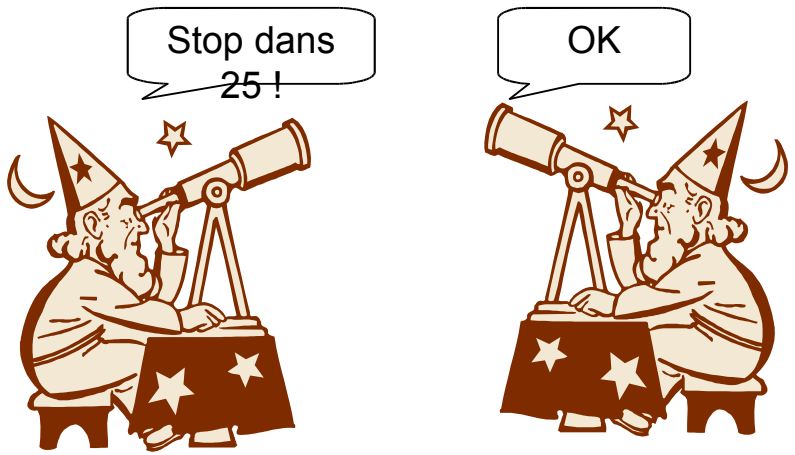
lancer le missile

Annulation = rappeler

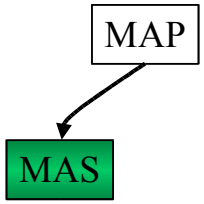
Ramener le missile dans le silo !?

Exemple 2

Sonde



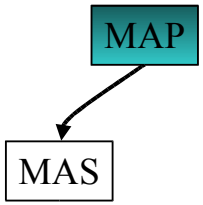
Conception : interface d'un MAS (II)



L'interface d'un MAS/MAN est définie par :

- Univers du discours
  - Modèle de classes (avec attributs)
  - Diagrammes État/Transition
- Liste des services
  - Signature (nom, arguments, résultat)
  - Texte structuré
- Sémantique de l'interface
  - Diagrammes de séquences
    NB : exhaustivité souvent impossible => Exemples de séquences typiques et/ou interdites
  - Diagrammes d'Activité (si besoin)

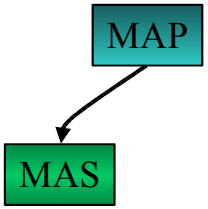
Conception : interface d'un MAP

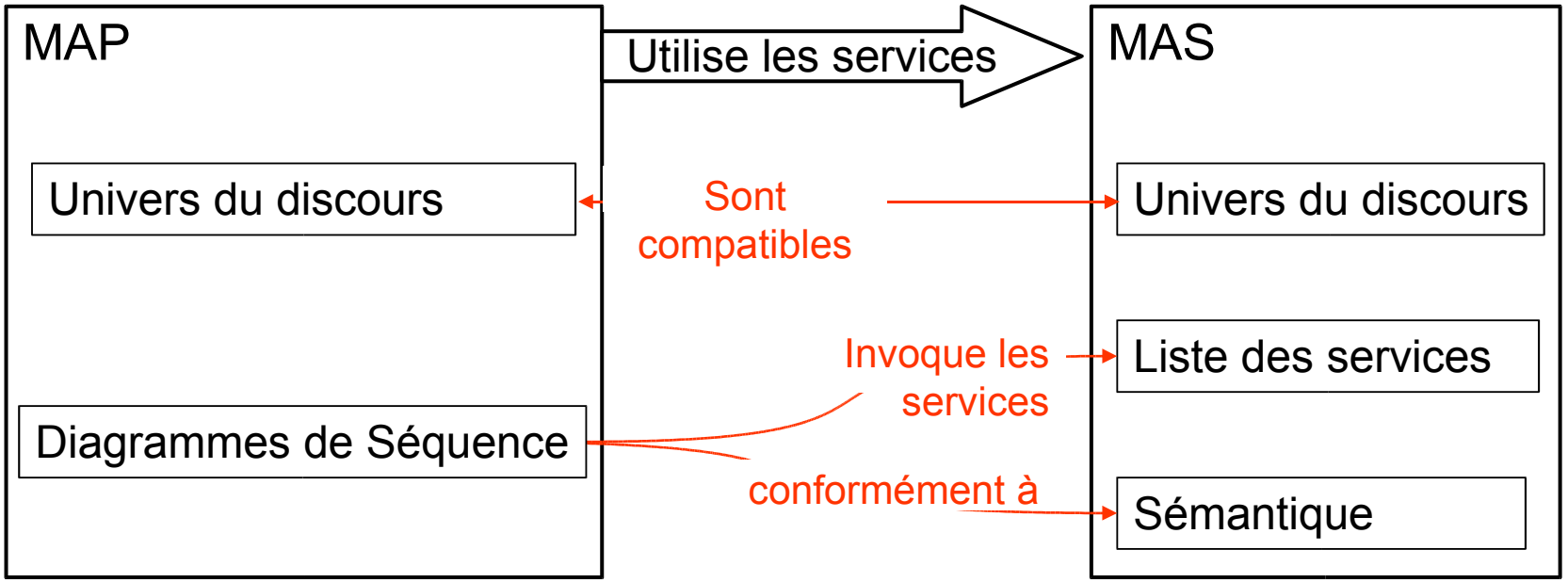


L'interface d'un MAP est définie par :

- Univers du discours
  - Modèle de classes (avec attributs)
  - Diagrammes État/Transition
- Réalisation des cas d'utilisation
  - Diagrammes de séquences : invocation des Services des MAS
    NB : Aussi exhaustif que possible
  - Diagrammes d'Activité (si besoin)

Mise en oeuvre
Conception : interaction MAP / MAS





Conception : Liens vues fonctionnelle et applicative (I)

La vue Applicative est structurée pour garantir
- les portées transactionnelles
- les exigences de performance
- la résilience du Système Informatique
- la maintenance du Système Informatique

D'où :

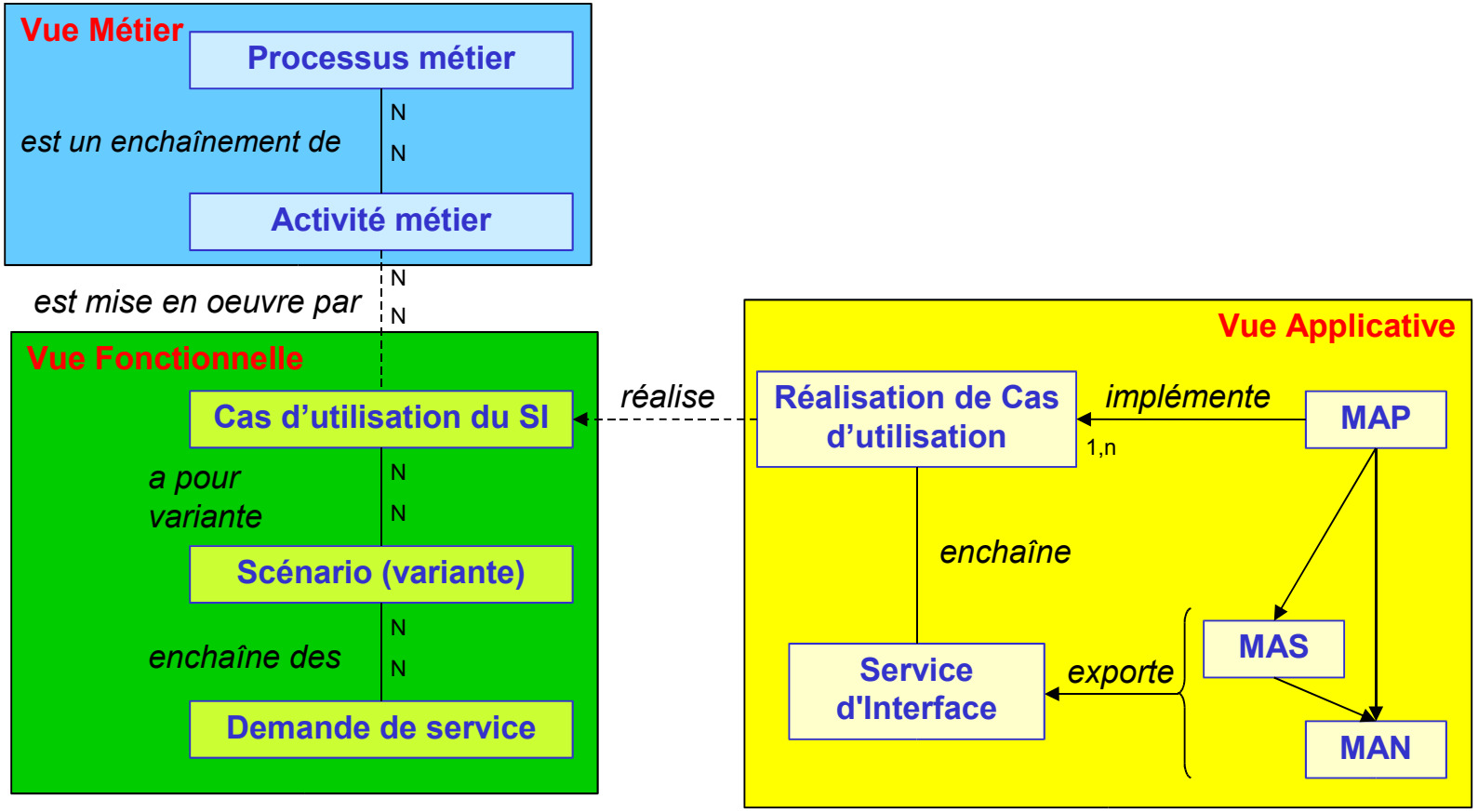
Différences de structure entre la vue fonctionnelle et la vue applicative et pas de relations automatiques directes entre

Services Applicatifs	et	Modules Applicatifs
Demandes de Services	et	Services d'Interface Importée
Offres de Services	et	Services d'Interface Exportée

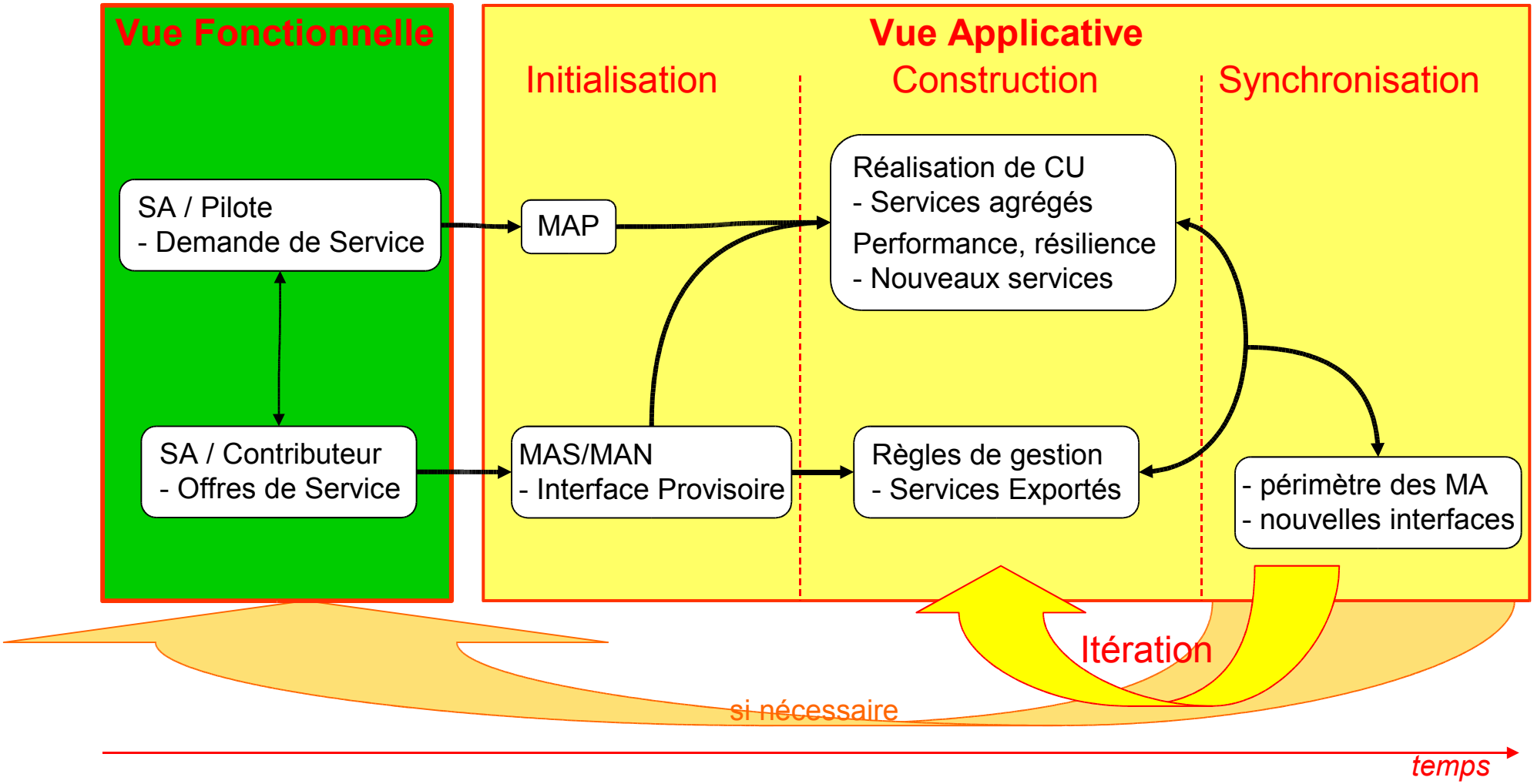
Mais relation forte :

Cas d'Utilisation	⇔	Réalisation de Cas d'Utilisation
vue fonctionnelle		vue applicative

Conception : Liens vues fonctionnelle et applicative (II)



Conception : Liens vues fonctionnelle et applicative (III)



- Un système d'information est composé d'au moins trois types d'applications :
  - Celles pilotant les processus de gestion structurels, soumises à des opérations de maintenance continues et exigeant un niveau de qualité d'exploitation important. Nous leur réserverons la dénomination *modules applicatifs* ;
  - Celles répondant à des besoins ponctuels soit limités dans le temps soit de faible complexité pour lesquelles le temps de réalisation doit être court. Nous les appellerons *appliquettes* ;
  - Celles assurant les opérations d'exploitation techniques sans contenu fonctionnel (p.ex. lancement d'opérations de ré-indexation de bases de données, installation de programmes, archivage des journaux d'activité, etc.). Nous les qualifierons de *scripts*.

Réalisation : Langages de programmation

- Le choix d'un outil de conception doit reposer avant tout sur un langage de programmation standard et des bibliothèque documentées non spécifiques à un acteur.
- Il est donc préférable, pour le développement des modules applicatifs, de choisir un langage structuré apte à exprimer nativement des sémantiques complexes et pouvoir subir des rétro-analyses efficaces.
  - Java est aujourd'hui le choix incontesté, Python est en forte progression

- Pour les appliquettes et les scripts, respecter de telles contraintes n'est pas nécessaire à condition de respecter les principes d'architectures.
  - PHP et Perl sont des bons choix

Architecture : Formats de données

- Nous disposons aujourd'hui d'une norme de description de données structurées universellement adoptée et très simple d'utilisation : XML.
- Son universalité tient à la distinction faite entre les notions de structure, de sémantique (grammaire : X-Schema), et de présentation (XSL).
- Il est possible, dans le cadre de ces normes, d'assurer une gestion spécifique des sémantiques et de constituer des dictionnaires de données transverses. Des grammaires spécialisées par domaine commencent à apparaître en grand nombre (bureautique – OO.org –, rendu des données – VoiceML, SVG, WML –, annuaires de services – WSDL –, ou échanges de données métier – ebXML).

Architecture : Formats de données – Références

- XML fait référence à un corpus de normes définies par le W3C :
  http://www.w3.org/XML
- Documentation utile :
  - http://www.xml.com
  - http://www-106.ibm.com/developerworks/xml
- La normalisation des schémas est conduite par le consortium OASIS (Organization for the Advancement of Structured Information Standards) :
  - http://www.oasis-open.org

Architecture : Poste client

- Le poste de travail est l'élément le plus fragile d'un système d'information (grand nombre, au contact direct des utilisateurs qui souhaitent les personnaliser, forte variabilité technique, etc.).
- Deux modèles complémentaires permettent de le « solidifier » :
  - Adopter une architecture n'ayant recours qu'à des éléments standards et robustes (navigateurs web comme « client universel ») ne nécessitant pas l'installation de composants spécifiques ;
  - Empêcher les utilisateurs d'effectuer des modifications (p.ex, amorçage en réseau de machines sans stockage, modèle client X11, VNC ou RDP).

- Un poste de travail « universel » est celui respectant le corpus de normes suivant :
  - HTTP et extensions comme protocole d'accès :
    http://www.w3.org/Protocols/Specs.html
  - HTML et normes associées comme langage de description :
    http://www.w3.org/MarkUp
  - ECMAScript (JavaScript) pour les manipulations clientes
    http://www.ecma-international.org/publications/standards/ECMA-262.HTM
  - Capacités de lecture d'objets externes (SVG, PNG, JPEG, etc.)
- Voir aussi le site OpenWeb : http://openweb.eu.org

Architecture : Communication (I)

- (Rappel)
  Historiquement, les réseaux d'informatique de gestion reposaient, en France, sur un protocole perfectionné mais lourd à l'utilisation : X25. La généralisation des réseaux IP est une évolution récente.
- A partir des concepts fondamentaux du modèle ISO, de nombreux protocoles applicatifs ont été spécifiés depuis 20 ans. Toutefois, la tendance actuelle conduit à en privilégier un petit nombre, HTTP en tête.
- Lors de l'adoption d'un protocole au sein d'un système d'information, il est impératif de respecter la logique sur laquelle il repose.
  - Ex : éviter de faire du *reverse-DNS* pour des usages applicatifs

Architecture : Communication (II)

- Les architectures distribuées utilisent des composants logiciels nommés génériquement « logiciels d'intermédiation » (*middlewares*).
- Ce terme recouvre une grande diversité de techniques et de produits souvent inspirés du *Remote Procedure Call* (RPC). Toutefois, nous pouvons les classifier en fonction du niveau d'impact en conception.
  - Les techniques « intrusives » imposant une politique d'utilisation.
  - A contrario, certaines techniques fournissent un mécanisme imposant très peu de contrainte d'utilisation. C'est notamment le cas de SOAP qui fonde la mode des *Web*-Services (http://www.w3.org/2002/ws)

    Cette dernière catégorie est la seule à même d'assurer une certaine indépendance et souplesse sur le long terme.

- Il est possible d'assurer l'interfaçage entre les outils de conception et de développement (XMI).
- Attention toutefois à ne pas tomber dans la tentation de la génération de code intégrale (dépendance à un AGL, surtout en cas de dysfonctionnement).
- Il est toutefois peu risqué de le faire sur certains segments applicatifs (par exemple en couche présentation, en adoptant un schéma MVC implémenté dans un framework standard – cas de la génération, via XSLT, de struts à partir de XMI).

Réalisation : Gestion du cycle de vie

- (Rappel) Un cycle de vie applicatif complet :
  Conception / Spécifications générales et détaillées / réalisation / tests unitaires / recette fonctionnelle et technique / intégration et qualification / expérimentation et mise en exploitation / gestion des anomalies et maintenance / retrait de service
- Un enjeu est de fixer et d'outiller les procédures de suivi du logiciel au long du cycle de vie. Cela concerne en particulier la gestion des versions, la gestion des anomalies, et la gestion de la documentation
- Il est enfin indispensable de définir les jalons qualité (p.ex l'intégration comme un sas) permettant d'améliorer l'exploitation et fixer les responsabilités (éviter le symptôme de la MOE « toujours coupable »).

- Le mode de conception privilégiant la mutualisation et la ré-utilisation fait peser des contraintes de plus en plus forte sur les briques transversales du SI. Pourtant les composants intersticiels, ne portant pas d'enjeu fonctionnel évident et souvent très techniques, n'obtiennent pas toujours une attention à la mesure de leur sensibilité.

  Il est recommandé d'identifier ces éléments, de les organiser en mode projet, et d'identifier la dépendance des projets fonctionnels par rapport à eux. Ceci afin de ne pas sous-estimer les ressources qui leur sont affectées.

- La généricité est la plus grande qualité des composants techniques. Doivent être distingués le mécanisme qu'il procure et son mode d'utilisation (*mecanism vs. Policy*).

Infrastructure : Authentification des utilisateurs

- Isoler un annuaire transverse des données d'identification (LDAP) et d'authentification (mots de passe, certificats X.509).
- Les « serveurs d'authentification » s'appuient sur l'annuaire et assument la charge d'identification des utilisateurs et d'attribution des profils pour le compte des applications (architectures possibles : en coupure – p.ex un reverse proxy HTTP(S) – ou par jetons).
- Les applications et autres éléments d'infrastructure (postes de travail, clients de messagerie) pourront aussi utiliser l'annuaire qui maintient l'homogénéité du système.
- L'annuaire est alimenté par une application extrayant les données du système de GRH, calculant les profils en masse et offrant aux chefs de service une gestion par exceptions.

Infrastructure : exemples

- Transferts de fichiers : séparation du protocole (FTP, SSH) et des fonctions de planification, sécurisation (intégrité) et contrôle.
- Administration et supervision : séparer les sondes de collecte des données, leurs modalités d'accès (SNMP) et les outils d'analyse (levée d'incident, supervision statistique, analyse décisionnelle).
- Messagerie et travail de groupe : attention aux systèmes intégrés, souvent rigides, peu performants et peu respectueux des normes (SMTP, POP, IMAP, vCalendar, vCard).
- Infrastructure d'impression : identifier les gestionnaires de files d'attentes (*spoolers*) alimentés par les applications et postes de travail selon un seul protocole (IPP) et dans un seul format (PostScript) et gérer hors des postes de travail les filtres et pilotes matériels (*drivers*)

- Dans un projet pris dans ses contraintes budgétaires et calendaires, les objectifs non directement opérationnels sont souvent mis au second rang.

  Il est donc nécessaire d'instrumenter le suivi des principes transverses et de les faire porter par la hiérarchie.

- Ce besoin est renforcé si de multiples acteurs interviennent ou en contexte de programme comportant de multiples projets.

- Il est préférable de laisser le contrôle à la direction de projet, garante de la cohérence et de ses objectifs particuliers, et de définir quelques jalons obligatoires et des structures transverses d'expertise et d'assistance chargées de veiller aux objectifs généraux (leur éventuelle application forcée relevant d'un arbitrage hiérarchique).

- Il est impératif d'établir et de livrer aux projets avant leur lancement des spécifications précisant les contraintes techniques générales ;

- Livrables d'un projet devant suivre les recommandations générales :
  - Définition de l'architecture applicative et technique du projet comme déclinaison de la conception d'ensemble (antérieure aux spécifications techniques) ;
  - Cahier de tests unitaires et d'intégration (avant l'entrée en tests) ;
  - Documentation d'exploitation et d'assistance (avant l'entrée en mise au point / certification) ;
  - Documentation de développement (avant le passage en maintenance).

- Par exemple, au sein de l'administration fiscale, ont été créées trois structures transverses (et des processus de décision associés) :
  - Une *direction de la production* chargée d'établir les contraintes d'exploitation et d'assistance générales s'appliquant aux projets, de diriger les centres de production et de gérer l'infrastructure ;
  - Une *direction technique* chargée de piloter les processus de choix technologiques, de contrôler in fine leur bonne application, d'assister les projets en phase d'étude, et de mener les projets d'évolution de l'infrastructure. Les domaines couverts sont :
    - L'architecture des applications, la sécurité et l'infrastructure ;
    - L'ingénierie du logiciel et les tests.
  - Une *structure d'intégration* chargée d'assurer la certification qualité des livrables à mettre en production.

- Outre le contrôle organisationnel des projets et des choix technologiques, la maîtrise d'un système d'information sur le long terme passe aussi par une relation équilibrée avec ses fournisseurs.

  Il est important de noter que le coût de sortie d'une solution est sensiblement plus élevé en informatique que dans beaucoup d'autres secteurs industriels du fait des inter-dépendances étroites entre les éléments du système. L'équilibre client-fournisseur du fait de la concurrence y est donc beaucoup plus incertain.

- La pratique de la neutralité vis à vis des composants matériels est donc critique. Pour les logiciels, outre les préconisations de cette présentation, il est possible de faire appel aux *logiciels libres* du fait de leur supériorité technique sur certains segments (infrastructure) et des offres d'assistance et de support aujourd'hui disponibles.