# THÈSE

présentée par

**Lionel BLANC DIT JOLICOEUR**

pour obtenir le diplôme de
**DOCTEUR DE L'UNIVERSITÉ DE SAVOIE**
(Arrêté ministériel du 30 mars 1992)

**Spécialité : GÉNIE INDUSTRIEL**

---

## *Modélisation des processus « métier » mis en œuvre dans une approche EAI en vue de leur pilotage « Le pilotage des applications intégrées »*

---

Soutenue publiquement le 22 décembre 2004 devant le jury composé de :

| | | |
|---|---|---|
| **Alain BOURJAULT** | Président du jury | Professeur à l'Université de Franche-Comté |
| **Jean-Claude DERNIAME** | Rapporteur | Professeur à l'Université de Nancy |
| **Hervé PINGAUD** | Rapporteur | Professeur à l'Ecole des Mines d'Albi Carmaux |
| **Michel SCHNEIDER** | Examinateur | Professeur à l'Université de Clermont-Ferrand |
| **Alain HAURAT** | Directeur de thèse | Professeur à l'Université de Savoie |
| **Christian BRAESCH** | Co-encadrant | Maître de Conférences à l'Université de Savoie |
| **Pierre LADET** | Membre invité | Professeur à l'Institut National Polytechnique de Grenoble |
| **Flavio OQUENDO** | Membre invité | Professeur à l'Université de Savoie |

Préparée au sein du LISTIC
Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance

En collaboration avec THESAME – Mécatronique & Management

*A mes proches et amis…*
*A mon grand-père…*

# REMERCIEMENTS

Ce travail de doctorat a été réalisé au sein du LISTIC – Laboratoire d'Informatique, Systèmes, Traitement de l'Information et de la Connaissance, un des trois laboratoires de l'ESIA – Ecole Supérieure d'Ingénieurs d'Annecy.

J'adresse mes plus vifs remerciements à M. Hervé PINGAUD, Professeur à l'Ecole des Mines d'Albi Carmaux et à M. Jean-Claude DERNIAME, Professeur à l'Université de NANCY, pour m'avoir fait l'honneur d'étudier mes travaux de thèse et de les avoir cautionnés en qualité de rapporteurs.

Je tiens également à remercier M. Alain BOURJAULT, Professeur à l'Université de Franche-Comté qui m'a fait l'honneur de présider le jury, ainsi que M. Michel SCHNEIDER, Professeur à l'Université de Clermont-Ferrand, M. Pierre LADET, Professeur à l'Institut National Polytechnique de Grenoble et M. Flavio OQUENDO, Professeur à l'Université de Savoie, pour avoir accepté de faire partie de ce jury.

Je tiens à exprimer toute ma gratitude à M. Alain HAURAT, Professeur à l'Université de Savoie et nouvellement en retraite, pour avoir dirigé mes travaux, pour ses conseils et son expérience. Un grand merci pour ses critiques toujours très constructives durant le déroulement de la thèse et au moment de la rédaction, et pour son perfectionnisme qui ont rendu ce manuscrit plus compréhensible.

Je tiens également à remercier M. Christian BRAESCH, Maître de Conférences à l'Université de Savoie, pour m'avoir co-encadré et soutenu durant ces trois années. Une pensée toute particulière quant aux nombreuses heures passées ensemble, à rédiger des livrables relatifs au projet européen ArchWare.

Mes sincères remerciements vont aux membres de THESAME – Mécatronique & Management, notamment à M. André MONTAUD et à M. Régis DINDELEUX, pour avoir eu confiance en moi et pour avoir financé mes trois années de thèse à travers la réalisation des tâches relatives au Workpackage 7 du projet européen ArchWare, dont THESAME était le pilote.

Un merci plus particulier à mes amis et collègues Fabien LEYMONERIE et Frédéric POURRAZ pour leur travail et leur bonne humeur. Merci également aux personnes de la société AGILIUM pour leur disponibilité et leur gentillesse.

Mes remerciements les plus sincères vont aux différents membres du laboratoire LISTIC, et plus particulièrement aux personnes du groupe Génie Logiciel et Méta-modélisation, à l'ensemble des doctorants du laboratoire, et à Valérie pour la relecture de ce mémoire de thèse.

Enfin, je remercie de tout cœur tous mes proches pour la confiance, le soutien et l'aide qu'ils m'ont apportés durant toute la durée de cette thèse. Merci également à tous mes amis chasseurs pour tous les bons moments passés hors du laboratoire.

# TABLE DES MATIERES

# TABLE DES FIGURES

## CHAPITRE 1 :

## PROBLEMATIQUE ET POSITIONNEMENT



## CHAPITRE 2 :

## ETAT DE L'ART AUTOUR DE NOTRE APPROCHE DE MODELISATION

**CHAPITRE 3 :**

**NOTRE CADRE DE MODELISATION**



**CHAPITRE 4 :**

**DEFINITION ET UTILISATION DE STYLES ARCHITECTURAUX**



**CHAPITRE 5 :**

**MISE EN ŒUVRE, IMPLEMENTATION ET PROTOTYPAGE**

**ANNEXES**

# Introduction

# Introduction

Aujourd'hui, il n'est plus à démontrer que les entreprises doivent tenir compte de plus en plus d'informations provenant des multiples applications hétérogènes. A l'heure actuelle, il existe des solutions technologiques permettant d'intégrer les applications logicielles d'une entreprise. Toutefois, ces solutions souffrent d'une absence de cadre de modélisation capable de fournir une vision globale de l'entreprise et, plus particulièrement, de son système d'information.

**Le système d'information d'une entreprise doit donc masquer l'hétérogénéité des applications logicielles en proposant une vision unifiée des données aux différents acteurs de l'entreprise. Cette caractéristique nécessite un cadre de modélisation capable de fournir une vision globale de l'entreprise.**

De plus, les entreprises doivent être agiles (réactives et proactives) pour répondre aux besoins de leurs clients. Cette agilité s'appuie sur une approche processus. Aussi, la maîtrise du système d'information de l'entreprise nécessite une modélisation de ces processus pour permettre leur pilotage.

**En conséquence, le système d'information doit s'appuyer sur un cadre de modélisation formalisant les différents processus d'une entreprise et proposant un modèle de pilotage de ces processus.**

Le cadre de modélisation doit garantir la cohérence entre les perceptions des différents acteurs de l'entreprise. Le paradigme d'ambivalence propose d'étudier la réalité opérationnelle d'une entreprise à travers deux points de vue. Le point de vue « métier » fournit une représentation des activités de l'entreprise réalisées en vue de satisfaire les besoins du marché tandis que le point de vue « donnée » fournit une vision unifiée des données et de leurs traitements.

**Le système d'information doit s'appuyer sur le paradigme d'ambivalence pour appréhender la réalité opérationnelle d'une entreprise à travers les points de vue « métier » et « donnée ».**

Lorsque le nombre de composants d'un système est important et que les étapes de développement font intervenir différents métiers (par exemple, génie industriel et génie logiciel), il est légitime de formaliser le système par son architecture. La représentation de la réalité opérationnelle au sein des deux points de vue est alors obtenue par la formalisation des architectures industrielles et logicielles.

**Le système d'information doit donc s'appuyer sur un cadre de modélisation capable de formaliser les architectures industrielles et logicielles d'une entreprise.**

Le paradigme d'ambivalence précise qu'il est nécessaire de définir des correspondances entre les concepts utilisés dans chacun des points de vue. Ces correspondances sont nécessaires car toute modification ou changement réalisé au sein d'une architecture doit avoir un impact immédiat sur l'autre. Nous considérons que la définition des correspondances entre les concepts des deux points de vue est facilitée par l'utilisation d'un langage formel centré architecture.

**Le système d'information doit intégrer les correspondances entre les concepts issus des deux points de vue pour garantir la cohérence des perceptions des différents acteurs de l'entreprise.**

Enfin, face à l'évolution du marché, les entreprises éprouvent de plus en plus de difficultés à réagir aux sollicitations auxquelles elles sont soumises. Ce constat nécessite pour l'entreprise une adaptation constante à un environnement en perpétuelle évolution.

**Le système d'information doit donc permettre l'évolution des architectures industrielles et logicielles en garantissant la satisfaction d'un ensemble de propriétés définies par le domaine étudié.**

Le modèle OLYMPIOS fournit un cadre de modélisation permettant de formaliser les processus de l'entreprise. Le projet ArchWare (*ARCHitecting evolvable softWARE*) – IST-2001-32360 – propose quant à lui, un ensemble de langages et d'outils centrés architecture pour l'ingénierie des systèmes logiciels évolutifs.

**Le travail réalisé dans le cadre de cette thèse consiste à intégrer les différents concepts du modèle OLYMPIOS dans un ensemble de langages centrés architecture. Il a donc été nécessaire d'étudier les différents concepts du modèle OLYMPIOS pour les appliquer dans le cadre du projet ArchWare.**

En résumé, l'approche présentée dans le cadre de ce mémoire considère que :
**L'agilité d'une entreprise nécessite un système d'information capable d'intégrer la modélisation des processus par l'utilisation d'un langage formel centré architecture pour appréhender les points de vue « métier » et « donnée » à travers un ensemble d'architectures industrielles et logicielles évolutives. Nous nous appuyons sur le cadre de modélisation proposé par OLYMPIOS pour formaliser les processus et sur les langages centrés architecture pour représenter les architectures industrielles et logicielles.**

Nous proposons de traiter cette problématique en montrant dans un premier temps que l'entreprise doit s'appuyer sur une approche processus et que la réalité opérationnelle doit être appréhendée à travers un point de vue « métier » et un point de vue « donnée ». Nous identifions ensuite des solutions technologiques capables de prendre en compte la synchronisation des processus d'une entreprise. Toutefois, nous montrons que ces solutions ne permettent pas de modéliser l'entreprise dans sa globalité. C'est pourquoi, nous proposons un cadre de modélisation capable de formaliser le pilotage des processus d'une entreprise. Ce cadre représente les points de vue « métier » et « donnée » par des architectures industrielles et logicielles. Aussi, nous nous appuyons sur le concept de style architectural pour formaliser ces architectures. Enfin, nous illustrons l'application de ces concepts à travers un cas d'application développé grâce à l'environnement proposé par le projet ArchWare.

En conséquence, ce mémoire de thèse est organisé comme suit :

- Le **chapitre 1** présente la problématique de nos travaux et notre positionnement. Nous exposons, dans un premier temps, l'importance de considérer l'entreprise comme un système pour caractériser son système d'information. Nous présentons ensuite le paradigme d'ambivalence qui considère que la réalité opérationnelle de l'entreprise doit être appréhendée à travers un point de vue « métier » et un point de vue « donnée ». Enfin, ce chapitre présente le projet ArchWare et le cadre de modélisation OLYMPIOS utilisés dans le cadre de nos travaux.

- Le **chapitre 2** décrit l'évolution des approches liées aux points de vue « métier » et « donnée ». Il présente ensuite des solutions existantes permettant l'intégration de ces deux points de vue et identifie les lacunes de ces approches. Enfin, ce chapitre montre que ces différentes évolutions ont modifié la démarche d'ingénierie des systèmes d'information.
- Le **chapitre 3** présente un modèle d'architecture et un modèle de pilotage qui sont à la base des approches utilisées pour formaliser les architectures industrielles et logicielles. Dans un second temps, ce chapitre propose des modèles d'architectures industrielles et logicielles pour formaliser les points de vue « métier » et « donnée ». Enfin, ce chapitre décrit les mécanismes utilisés pour caractériser les interactions entre les architectures industrielles et logicielles.
- Le **chapitre 4** présente les principaux concepts et mécanismes d'un langage formel permettant de définir des styles architecturaux. Il décrit ensuite succinctement la définition de style « patrons » d'architecture qui permettent la formalisation d'architectures adaptées à un domaine spécifique. Enfin, ce chapitre présente des styles architecturaux issus de travaux réalisés dans le cadre du projet ArchWare.
- Le **chapitre 5** présente, dans un premier temps, les différents langages centrés architecture et les différents outils proposés par le projet européen ArchWare. Il décrit ensuite l'application mise en œuvre pour valider les travaux réalisés dans le cadre de cette thèse.

# Chapitre 1 :
# Problématique et positionnement

# Chapitre 1 :
# Problématique et positionnement

## 1. Introduction

Par définition, « *une entreprise correspond à la plus petite combinaison d'unités légales qui constitue une unité organisationnelle de production de biens et de services jouissant d'une certaine autonomie de décision, notamment pour l'affectation de ses ressources courantes. Une entreprise exerce une ou plusieurs activités dans un ou plusieurs lieux. Une entreprise peut correspondre à une seule unité légale*[1] ». Rappelons les trois approches caractéristiques que nous avons de l'entreprise [Pau 92] :

- La première approche vise à étudier les contraintes subies par les différents membres, du fait de leur appartenance à une organisation. En effet, il existe souvent une divergence entre les exigences de l'organisation et les aspirations de chacun de ses membres. L'entreprise apparaît alors comme un lieu où s'exerce un pouvoir.
- La seconde approche consiste à caractériser le comportement de l'entreprise à partir des processus décrivant son fonctionnement interne. A travers cette approche, l'entreprise est caractérisée comme un ensemble de moyens destinés à garantir la gestion d'un ensemble de ressources pour assurer une fonction reconnue.
- Enfin, la dernière approche conduit à étudier l'entreprise comme un agent entretenant des échanges avec son environnement (échanges marchands, rapports non marchands et stratégies de pouvoir). En fait, cette approche vise à spécifier les conditions stratégiques de fonctionnement de l'entreprise, c'est-à-dire les conditions externes qui déterminent sa survie et sa capacité d'adaptation.

Dans le cadre de nos travaux, nous nous focalisons sur la seconde approche et, plus particulièrement, sur la formalisation des processus en vue de leur pilotage. Cette approche est rendue nécessaire du fait que les entreprises éprouvent de plus en plus de difficultés à réagir aux multiples sollicitations auxquelles elles sont soumises. En effet, l'entreprise doit, pour assurer sa survie et sa croissance, s'adapter, voire anticiper les évolutions de l'environnement (évolutions sociales, techniques ou technologiques).

Par conséquent, pour survivre et continuer à se développer, l'entreprise doit d'une part, être réactive face à l'évolution de son environnement et, d'autre part, tenir compte de l'évolution permanente des différentes techniques et technologies, en particulier dans le domaine des technologies de l'information. Nous définissons par **agilité** la capacité d'une entreprise à faire face à ces évolutions. Ce constat nous conduit à percevoir l'entreprise à travers deux points de vue :

- un point de vue « métier » qui identifie les différents processus mis en œuvre au sein de l'entreprise pour coordonner ses activités en fonction d'une finalité,
- un point de vue « donnée » qui fournit une vision unifiée des données nécessaires au déroulement et au pilotage des processus.

[1] Source : Règlement (CEE) n° 696/93 du Conseil, du 15 mars 1993, relatif aux unités statistiques d'observation et d'analyse du système productif dans la Communauté, section III A.
http://forum.europa.eu.int/irc/dsis/bmethods/info/data/new/coded/fr/gl000057.htm

Aussi, nous nous appuyons sur le paradigme d'ambivalence pour appréhender ces deux points de vue. Mais, avant de présenter ce paradigme, nous rappelons dans un premier temps les principaux concepts issus de la théorie des systèmes. En effet, cette approche est à la base de la construction de représentations pertinentes. Dans un troisième temps, nous présentons le cadre de modélisation OLYMPIOS qui permet la formalisation des processus d'une entreprise. Enfin, nous concluons ce chapitre en introduisant le projet européen ArchWare qui a été le cadre de développement de ces travaux.

# 2. L'entreprise-système

Les trois approches, identifiées dans l'introduction de ce chapitre, induisent des contraintes non homogènes qui rendent légitime le fait de considérer l'entreprise comme un système. Il est à noter que la conceptualisation d'une entreprise, et plus généralement d'une organisation, comme un système n'est pas originale. En effet, c'est au début des années 50 qu'apparurent et commencèrent à se développer des théories et des modèles basés sur le courant systémique. Dans un article de synthèse, B. Paulré [Pau 87] identifie deux grands courants issus de deux cultures disciplinaires différentes.

Le courant systémique en sociologie des organisations, d'inspiration fonctionnaliste, privilégie les ressemblances entre les structures biologiques et sociales et considère les organisations comme des systèmes naturels qui :

- ont des besoins qu'elles doivent satisfaire pour survivre,
- ont des comportements,
- interagissent avec d'autres systèmes sociaux,
- englobent des processus ou des composantes indépendantes.

Les organisations sont traitées comme des systèmes ouverts ce qui conduit à introduire l'environnement dans l'étude de l'entreprise.

Le second courant, d'inspiration scientifique et technique, a donné lieu à des travaux appliqués ou à des modèles d'entreprise. Beaucoup de ces travaux sont basés sur l'approche cybernétique pour :

- caractériser les conditions de viabilité de l'entreprise,
- étudier la décomposition, le contrôle et la structuration informationnelle des systèmes hiérarchisés,
- analyser les systèmes socio-économiques complexes (dynamique des systèmes),
- réinterpréter les fonctions de planification, d'organisation ou de communication.

Cette présentation succincte des principaux travaux réalisés au sein de ces deux courants peut laisser penser que tout a été dit dans le domaine d'une approche systémique des entreprises. Mais deux évolutions conduisent à reconsidérer le contenu et la pertinence d'une approche systémique des entreprises. La première évolution concerne l'environnement technologique, social et économique tandis que la seconde est liée au développement de l'approche systémique elle-même.

## 2.1. Les entreprises face à l'évolution de leur environnement

Depuis le premier choc pétrolier, l'environnement des entreprises a été modifié sous l'effet de l'évolution économique et industrielle (mondialisation des marchés, restructurations rapides et brutales, accentuation de la concurrence, etc.), de l'évolution financière (développement des

marchés financiers, internationalisation, accentuation des contraintes induites par les manœuvres financières, etc.), de l'évolution technologique (informatisation et automatisation des tâches, progrès technologique, etc.) et de l'évolution sociale (diminution des capacités hebdomadaires, etc.).

Pour faire face à cette évolution, les entreprises s'appuient sur des méthodes de planification et de gestion qui, pour la plupart, sont basées sur le cadre conceptuel proposé par R. Anthony [Ant 66]. Ce cadre propose une typologie de la planification et du contrôle de gestion :

- La planification stratégique élabore les objectifs et la politique de l'entreprise par des décisions majeures (alliance, différenciation, croissance, innovation, etc.) et elle définit les ressources à utiliser pour atteindre ces objectifs (acquisition, utilisation organisation).
- Le contrôle de gestion s'assure que les ressources sont utilisées de manière efficace et rentable pour satisfaire les objectifs de l'entreprise.
- La gestion opérationnelle vérifie que les tâches liées à l'exploitation sont accomplies de manière efficace et rentable.

Ce cadre s'appuie sur trois dimensions pour classifier chacun de ces modes de gestion :

- le temps (long terme, moyen terme, court terme),
- le niveau hiérarchique,
- l'importance des décisions.

Ce cadre conceptuel donne beaucoup d'importance à la planification et au contrôle de gestion, il privilégie donc la vision gestion à la vision stratégique. Aussi, M.E. Porter [Por 80] propose une vision différente en préconisant qu'une stratégie doit permettre de faire face aux cinq forces concurrentielles qui sont :

- le risque de nouveaux entrants dans le marché,
- l'intensité de la concurrence,
- la pression des produits de substitution,
- le pouvoir de négociation des clients,
- le pouvoir de négociation des fournisseurs.

Cette vision stratégique doit fournir à l'entreprise une position lui permettant de se différencier, de réduire ses coûts, d'innover, de croître et de s'allier. En effet, un environnement difficilement prévisible et évolutif exige « souplesse » et « mobilité ». En d'autres termes, les capacités d'adaptation et de changement sont désormais au cœur des stratégies de survie et de croissance des entreprises.

Parallèlement à ces évolutions, l'entreprise doit également faire face au développement rapide et constant des technologies de communication et de traitement de l'information qui a considérablement élargi le spectre des applications informatiques utilisables en son sein. En effet, il n'est plus à démontrer aujourd'hui que l'efficacité des entreprises dépend de sa capacité à s'appuyer sur différentes applications logicielles. Cette évolution des capacités logicielles concerne l'ensemble des secteurs d'activités de l'entreprise. Nous avons toutefois choisi de l'illustrer dans le domaine de la gestion de production car nous considérons que c'est dans ce domaine que les changements ont été les plus marquants [Thé 02] [BrB 04].

Dans les années 60 et au début des années 70, les applications informatiques étaient spécialisées et autonomes (logiciel de gestion des stocks, par exemple). L'apparition des bases de données a permis le développement de logiciels capables d'appréhender de façon plus

globale les fonctions métiers de l'entreprise (gestion de production, par exemple). Ces applications de type MRP – *Material Requirements Planning* – s'appuyaient sur des structures informatiques de type « *mainframe* » et avaient pour objectif de planifier les besoins en composants de l'entreprise [Orl 75]. Au début des années 80, l'augmentation de puissance des ordinateurs, associée au contexte économique concurrentiel, a fait évoluer les applications de type MRP en applications MRP II – *Management Resource Planning* – qui s'appuient sur le cadre conceptuel de planification et de gestion proposé par R. Anthony [VBW 88]. Cette augmentation de puissance des ordinateurs a également donné naissance à de nouveaux outils capables de prendre en charge des aspects tels que les prévisions et la distribution (DRP – *Distribution Resource Planning*). De plus, la plupart des applications installées au sein des entreprises ont fait l'objet de développements spécifiques pour introduire de nouvelles fonctions telles que la gestion de la qualité, de la maintenance, le suivi de production, etc.

Dans les années 80-90, l'utilisation de telles applications informatiques pose divers problèmes. En effet, ces applications sont souvent isolées et s'appuient sur des bases de données spécifiques, ce qui implique une structuration différente des données. Face à cette situation, des solutions s'appuyant sur une base de données unique apparaissent, c'est le cas des applications CFAO qui unifient les bases de données des solutions CAO (Conception Assistée par Ordinateur) et FAO (Fabrication Assistée par Ordinateur). Une autre approche pour maintenir la cohérence entre les données est le développement de programmes spécifiques pour synchroniser les bases de données entre les applications. Ces applications sont souvent dédiées à la gestion d'un site unique. Aussi, leur mise en œuvre dans le cadre d'une décentralisation nécessite la mise en place de mécanismes capables d'harmoniser les bases de données locales (celles des agences) avec une base de données centrale (celle du siège). Enfin, le contexte économique induit souvent des acquisitions ou des fusions entre entreprises, ce qui nécessite une mise en cohérence des différentes applications de gestion. Face à ces problèmes, une nouvelle génération d'applications émerge dans les années 1995. Les applications MRP II deviennent des ERP – *Enterprise Resource Planning* – où les différentes fonctions sont intégrées autour d'une base de données centralisée. Les applications DRP sont alors devenues des applications SCM – *Supply Chain Management* – tandis que les applications dédiées au contrôle des activités de production se sont transformées en MES – *Manufacturing Execution System* [MES 97b]. A l'heure actuelle, ces applications sont souvent considérées, à tort ou à raison, comme les « applications pivot » sur lesquelles de très nombreuses entreprises investissent pour obtenir un système d'information intégré capable d'apporter des gains sur l'ensemble de l'entreprise, grâce à une meilleure circulation de l'information. Il est à noter que des applications telles que les entrepôts de données [EVO 01] ont complété cette offre logicielle en fournissant des outils permettant une exploitation efficace et performante de données hétérogènes pour l'aide à la décision.

Cependant, ces différents progiciels de gestion intégrés ont montré leurs limites dans le fait d'intégrer l'ensemble des fonctionnalités de l'entreprise [MES 97a] [MES 97b]. Des solutions de type *Workflow* capables d'automatiser l'enchaînement d'un ensemble d'activités [Bou 02] et des outils tels que les APS – *Advanced Planning and Scheduling* – ont donc vu le jour pour suppléer aux déficiences des ERP. A l'heure actuelle, les solutions EAI – *Enterprise Application Integration* – et BPM – *Business Process Management* – viennent compléter cette offre pour d'une part, intégrer les processus transversaux de l'entreprise au sein de son système d'information et, d'autre part, ouvrir l'entreprise vers le marché du « *e-business* ». Les entreprises sont de plus en plus conscientes que cette intégration des processus

transversaux est un élément clé en terme de performance industrielle et un élément différenciateur en terme de compétitivité.

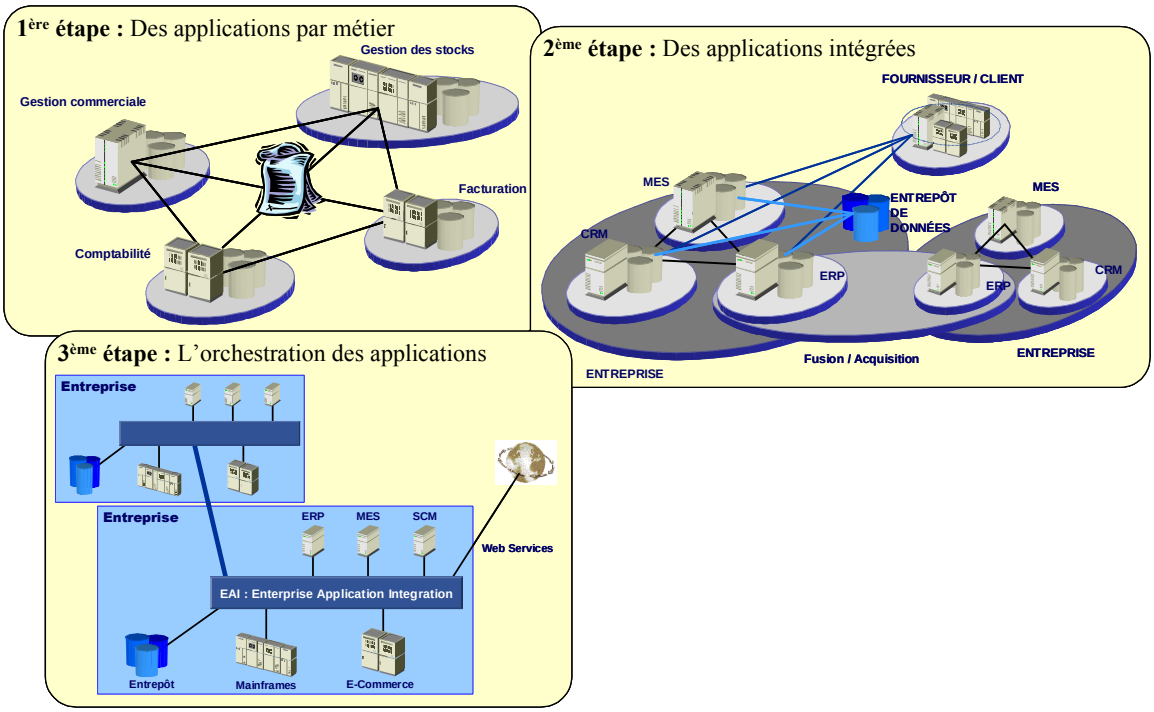



*Figure I.1. : Les trois grandes étapes de l'évolution [BrB 04]*

Parallèlement à ces évolutions, les échanges de données se sont automatisés grâce au standard EDI – *Electronic Data Interchange*. En effet, les normes X.12 aux États-Unis et Edifact en Europe ont formalisé les formats d'échange tels que les commandes, les confirmations, les factures, etc. Cependant, le prix du ticket d'entrée de ces systèmes est élevé car, pour en bénéficier, il faut s'équiper en matériel de terminaison, se connecter au réseau privé et interfacer ses propres applications avec le nouveau système de communication [AJP 00]. L'évolution d'Internet permet de généraliser l'interconnexion numérique entre les entreprises. Les standards tels que HTTP – *Hyper Text Tranfer Protocol* – et HTML – *Hyper Text Markup Language* – démocratisent l'accès aux solutions compatibles avec Internet car le prix d'un accès est dérisoire comparé aux coûts des solutions électroniques traditionnelles. La banalisation des échanges électroniques sur Internet marque une réelle évolution et entraîne de nouvelles possibilités. En effet, les flux EDI classiques relevaient généralement d'échanges entre deux partenaires (point à point) et traduisaient des processus simples. Les outils déployés sur Internet permettent (en s'appuyant sur les standards EDI) de gérer des processus transactionnels impliquant plusieurs parties (gestion des échanges avec les fournisseurs par une plate-forme de « *e-procurement* » par exemple).

En résumé, les entreprises éprouvent de plus en plus de difficultés à réagir aux multiples sollicitations auxquelles elles sont soumises. En d'autres termes, pour survivre et continuer à se développer, l'entreprise doit :

- être réactive pour réagir et s'adapter rapidement aux changements brusques de la demande et pour intégrer les transformations de l'environnement social, technique et technologique,
- tenir compte de l'évolution permanente des différentes techniques et technologies, en particulier dans le domaine du traitement de l'information.

Mais, si la fonction de progrès de l'entreprise devient nécessaire face à l'évolution de l'environnement, il doit subsister au sein de l'entreprise un « point fixe » (un invariant) sur lequel elle se fonde pour déterminer son identité. Cet invariant ne peut plus uniquement se baser en terme de « produit-marché », il doit se baser sur l'intelligence de l'entreprise c'est-à-dire l'ensemble de ses « savoir-faire », ses « savoir-décider » et de ses « savoir-s'informer » [Pau 87]. Aussi, nous nous appuyons sur la théorie des systèmes pour caractériser cet invariant.

## 2.2. Approche systémique et modélisation de l'entreprise

De nombreux travaux basés sur la théorie des systèmes se réfèrent au modèle classique de l'organisation structurée en deux sous-systèmes hérités de la cybernétique :

- le système de pilotage,
- le système piloté (souvent désigné *système opérant*).

Cette structuration permet la mise en œuvre d'une boucle de rétroaction entre ces deux parties. Le système de pilotage gère le fonctionnement du système piloté en fonction du (ou des) objectif(s) assigné(s) tout en tenant compte de l'évolution de l'environnement. Un des principaux aspects de l'approche cybernétique est la gestion des échanges d'information entre ces deux parties [Val 95]. Le volume d'information devenant de plus en plus grand à cause de la complexité croissante des structures traitées, il est nécessaire d'adopter une vision plus globale : l'approche systémique [Bev 95].

### 2.2.1. La notion de système

Un système est défini comme « *la représentation d'un phénomène actif perçu identifiable par ses projets dans un environnement actif, dans lequel il fonctionne et se transforme téléologiquement*[2] » [LeM 77]. De façon plus mnémotechnique, un système peut être perçu comme :

- quelque chose (n'importe quoi présumé identifiable),
- qui dans quelque chose (environnement),
- pour quelque chose (finalité ou projet),
- fait quelque chose (activité = fonctionnement),
- par quelque chose (structure = forme stable),
- qui se transforme dans le temps (évolution).

Suite à cette définition, le système peut être représenté par la figure suivante :

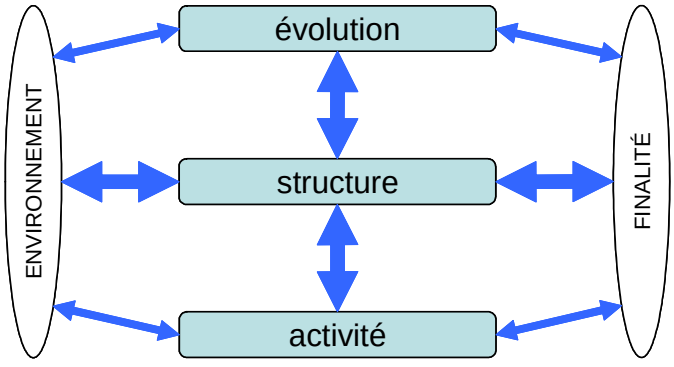



*Figure I.2. : Le paradigme systémique*

[2] Téléologique : en fonction d'un but.

D'après J. De Rosnay [Ros 75], l'entreprise est une cellule de production ayant sa vie propre (elle naît, croît, se développe, arrive à maturation et meurt). Elle produit des biens matériels (produits) ou immatériels (services) destinés au marché des biens et des services. La plupart du temps, cette fonction de production nécessite l'acquisition de produits et de services sur ce même marché, ce qui induit un flux bidirectionnel (de produits ou de services) circulant entre l'entreprise et le marché des biens et des services. L'entreprise doit également disposer de « facteurs de production » qui fournissent les ressources nécessaires à la mise en œuvre de la fonction production :

- travail : énergie fournie par les opérateurs, techniciens et cadres pour fabriquer les produits ou réaliser les services,
- énergie : combustibles et électricité nécessaires au fonctionnement des différents équipements de production,
- capital : ressources financières nécessaires à l'acquisition des équipements de production,
- savoir : brevets, licences et tous biens immatériels traduisant un savoir préalablement accumulé.

Ces différents facteurs induisent des flux spécifiques circulant de l'environnement (qui est structuré en marchés) vers l'entreprise :

- flux de personnes : marché du travail,
- flux de capital : marché des capitaux,
- flux d'énergie : marché de l'énergie,
- flux d'idées (brevets, licences) : marché du savoir.

Dans certains cas, ces flux sont également bidirectionnels. Par exemple :

- suite à une restructuration, l'entreprise peut être amenée à licencier une partie de son personnel,
- lorsque l'entreprise dégage des bénéfices,
- lors du dépôt d'un brevet suite à la conception d'un produit ou d'un procédé,
- une entreprise disposant d'une centrale hydraulique propre, peut vendre de l'énergie au marché de l'énergie.

En nous appuyant sur la définition suivante de l'entreprise[3] : « *Tout système socio-économique donné visant la production de biens ou des services pour satisfaire un marché (sa mission) en utilisant au mieux ses moyens (financiers, techniques et humains)* », nous pouvons caractériser l'entreprise de la façon suivante :

- quelque chose : ENTREPRISE,
- qui dans quelque chose : ENSEMBLE DE MARCHES,
- pour quelque chose : CROITRE ET SURVIVRE,
- fait quelque chose : PRODUIRE,
- par quelque chose : ENSEMBLE DE MOYENS,
- qui se transforme dans le temps.

[3] Définition proposée par le groupe de travail « Modélisation d'Entreprise » du Groupement de la Recherche en Productique. Il est à noter que ce groupe a intégré le groupe de travail Entreprise Communicante et Interopérabilité suite à la création du Groupement De Recherche « Modélisation, Analyse et Conduite des Systèmes dynamiques ».

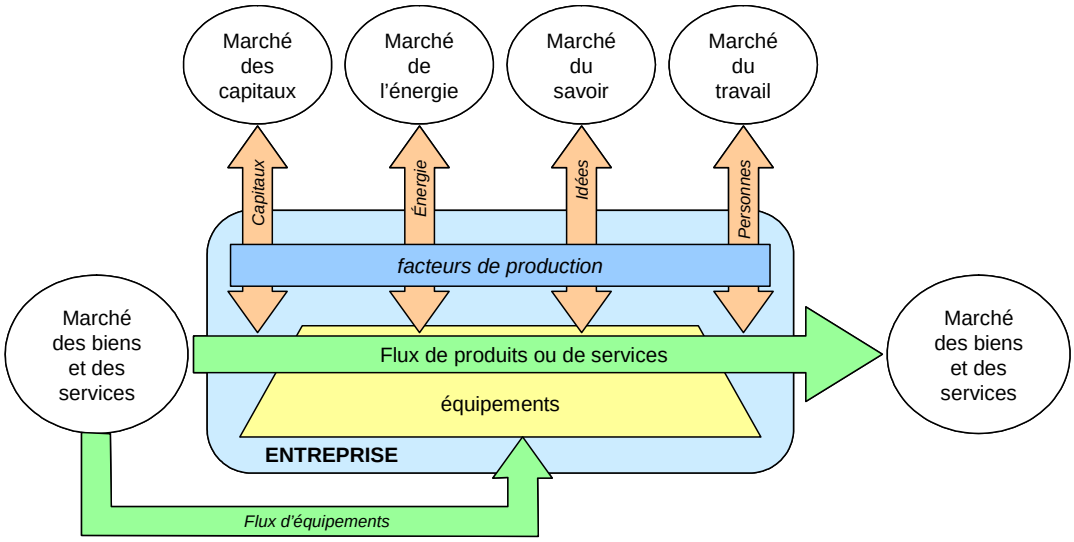



*Figure I.3. : L'entreprise-système*

D'après H.A. Simon [Sim 83], il existe deux modalités d'appréhension du monde réel :

- la perception, c'est-à-dire l'identification des objets du monde exprimée par des états,
- la description des processus et leur mise en œuvre (en termes de production d'objets).

La perception identifie les objets générés ou transformés qui circulent dans les différents flux de l'entreprise. Un processus peut être décrit comme un enchaînement partiellement ordonné d'exécution d'activités pour réaliser un objectif [Thé 02]. Les trois éléments de base d'un processus sont :

- l'**activité** qui est l'élément d'action atomique d'un processus (transformation non décomposable à un niveau d'abstraction donné),
- un **graphe d'activités** qui représente l'enchaînement des activités nécessaires à l'atteinte de l'objectif du processus,
- une **fonction de transition** qui contrôle le déroulement du processus.

Les processus assurent l'écoulement de ces flux, et ils permettent donc de concrétiser un système [BHB 95]. Une autre définition du concept de processus proposée par J. Miller[4] [LeM 77] fait émerger deux concepts importants dans une approche processus :

- les objets générés ou transformés par le processus ; Ils sont appréhendés à travers leurs différents états et ils constituent l'univers **processé**,
- les **processeurs** générant ou réalisant les transformations sur les objets.

Ayant rappelé les principaux résultats issus de la théorie des systèmes, nous allons maintenant détailler la fonction « pilotage » car elle vise à atteindre l'objectif assigné au système. Nous rappelons que nos travaux concernent la formalisation et le pilotage des processus d'une entreprise.

### 2.2.2. La fonction « pilotage » dans un système

La fonction pilotage peut être définie par l'analogie à la conduite des engins complexes proposé par J. Melèse [TaT 87] : « *piloter un engin, c'est choisir un objectif, définir la meilleure trajectoire, lancer l'engin puis corriger en permanence ses écarts par rapport à sa*

[4] Définition du concept de processus proposée par J. Miller : « *Tout changement dans le temps de matière, d'énergie ou d'information* ».

*trajectoire et, éventuellement, modifier la trajectoire, voire l'objectif, lorsque l'état de l'univers extérieur ou celui de l'engin montrent que le plan initial ne peut être maintenu* ».
En s'appuyant sur cette analogie, J. Mélèse [Mel 72] caractérise le système opérant par des *variables d'entrée* ($V_e$) et *de sortie* ($V_s$), capables d'observer des transformations selon des lois qui ne sont pas toujours linéaires ($V_s=f(V_e)$). De plus, des *variables essentielles* sont définies comme les indicateurs permettant l'évaluation des objectifs visés. Le système de pilotage dispose de *variables d'action* définissant les règles de fonctionnement du système.

Le pilotage s'appuie sur les valeurs des variables essentielles pour mettre en œuvre les deux fonctions suivantes :

- la fonction de contrôle qui détermine les valeurs des variables d'action en fonction des valeurs des variables essentielles,
- la fonction de régulation du système opérant qui, à l'aide de boucles de rétroaction, agit sur les variables d'entrée pour maintenir les variables de sortie à leurs valeurs désirées.

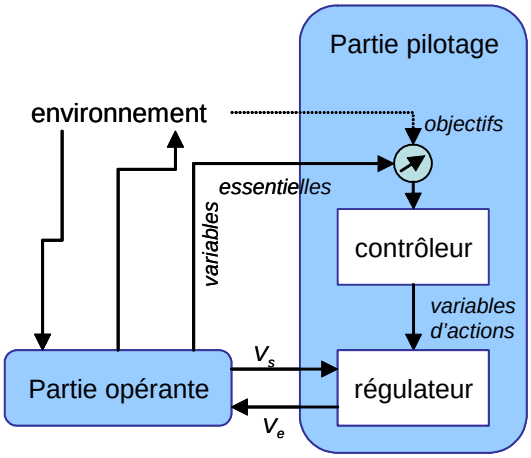



*Figure I.4. : Fonction de pilotage vue par J. Melèse*

Des travaux menés autour du modèle systémique de l'organisation ont montré l'importance de la cognition dans la boucle de pilotage et ont abouti à une nouvelle vision qui est illustrée par la « boucle canonique » proposée par H. Tardieu et M. Theys [TaT 87] :

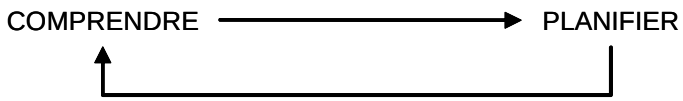



*Figure I.5. : Boucle canonique du pilotage*

La phase de « compréhension » concerne la manière d'appréhender une situation donnée. Elle consiste à trouver les explications et les effets des actions qui ont été mises en œuvre ou planifiées. La phase de « planification » définit un plan d'actions en fonction des objectifs fixés et de la situation appréhendée. Cette approche confère un aspect dynamique au pilotage. La phase de « compréhension » est totalement liée à l'évolution du système, et ne se limite pas aux seules situations bloquantes. En effet, suite à l'identification d'une dérive, la phase de planification doit proposer un plan d'actions capable de la corriger, ce qui induit la possibilité de prévenir l'apparition de dysfonctionnements.

R. Wilenski [Wil 83] montre que la planification et la compréhension ont en commun le même type de connaissance au sujet des plans d'action et propose de formaliser la boucle de pilotage par le schéma suivant :

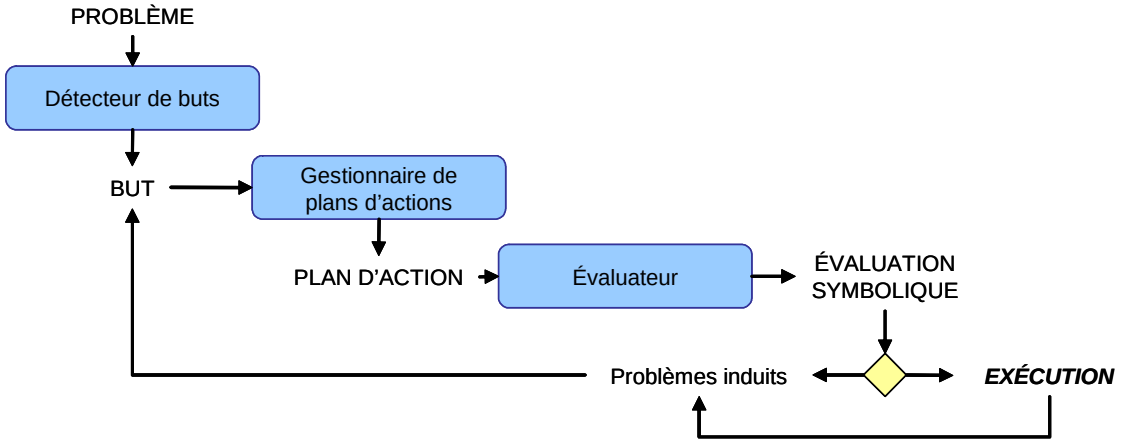



*Figure I.6. : Pilotage suivant la théorie des plans d'actions*

Le détecteur de buts identifie les situations (changement dans l'environnement, existence d'un autre but ou problème dans l'exécution d'un plan d'actions) auxquelles il associe des buts. Le gestionnaire de plans d'actions recherche les plans d'actions associés au but trouvé (plans stéréotypés, adaptation des plans existants à la situation courante ou à la réalisation d'autres buts). Enfin, l'évaluateur simule le comportement de l'organisation si le plan d'actions est exécuté. Il est à noter que le détecteur de buts reste actif durant l'évaluation pour identifier de nouveaux problèmes.

Une difficulté majeure de cette approche réside dans le fait qu'un but n'existe pas de façon isolée. R. Wilenski identifie des interactions négatives ou positives entre les buts d'un même acteur ou de plusieurs acteurs :

| | Interactions négatives | Interactions positives |
|---|---|---|
| Relations entre buts d'un même acteur | Conflit de buts | Superpositions de buts |
| Relations entre buts de différents acteurs | Compétition de buts | Coopération de buts |

La grande originalité de cette approche est de proposer d'utiliser la théorie des plans d'actions comme base d'une théorie de la compréhension pour des acteurs placés en « situation informationnelle » afin de comprendre par inférence le comportement des autres acteurs [TaT 87] :

| Ce qu'il faut expliquer | Explication |
|---|---|
| ACTION | Plan que cette action instancie |
| PLAN | But que ce plan satisfait |
| BUT | Problème qui a donné naissance à ce but |

En considérant le plan d'actions comme la description de l'enchaînement des activités d'un processus, le pilotage d'un processus consiste donc à faire évoluer ce processus vers son objectif, ceci via différentes situations intermédiaires. Pour signifier cette évolution, il est nécessaire de définir un processus de référence, désigné **référent**, définissant la situation finale attendue (réalisation de l'objectif) et les différentes situations intermédiaires. Le

pilotage est appréhendé à travers quatre phases (chaque phase mettant en œuvre des fonctions spécifiques) [Thé 02] :

- la phase initialisation construit le référent capable d'atteindre l'objectif fixé en précisant les activités à exécuter et leur enchaînement,
- la phase d'évolution exécute les activités du référent et évalue les situations obtenues par rapport aux situations attendues,
- la phase de correction stoppe l'évolution, corrige une non-conformité (situation obtenue non conforme à la situation attendue) en construisant un nouveau référent,
- la phase de terminaison évalue globalement le processus suite à son déroulement (état final atteint).

Ayant décrit les différentes formes de pilotage d'un système, il convient de définir maintenant le système d'information car ce dernier peut être défini comme un système utilisant des technologies de l'information pour saisir, transmettre, stocker, retrouver, manipuler ou afficher les informations nécessaires au pilotage des processus d'une entreprise. Le système d'information devra donc être capable de fournir les informations pour construire un référent, contrôler le déroulement des processus (exécution et évaluation), corriger le référent lorsqu'une non-conformité est détectée et évaluer le résultat final obtenu.

### 2.2.3. Le système d'information

Les concepts liés aux activités de pilotage identifiées dans le paragraphe précédent induisent le support d'un système d'information. En effet, « *un système d'information concerne un individu, ..., confronté à un problème décisionnel précis, dans un contexte organisationnel déterminé* » [Rei 00]. Nous complétons cette définition en considérant que le système d'information peut concerner un individu ou un groupe d'individus car « *une organisation est une unité de coordination, dotée de frontières repérables, fonctionnant de manière relativement continue en vue d'atteindre un objectif partagé par ses membres participants* » [Rob 90]. Il est donc possible de distinguer au sein d'une organisation :

- des activités individuelles définies en fonction du rôle de chaque participant,
- des actions collectives réalisées par un groupe d'individus.

La définition du système d'information est nécessaire car « *la sous-estimation, par la plupart de nos managers, pour ne pas dire leur méconnaissance, de l'importance de l'information comme matière première stratégique, utilisable au même titre que de l'énergie ou des matériaux, constitue, pour notre prospérité économique, une menace, d'autant plus grave que les causes sont plus difficiles à éliminer, cette attitude prenant racines dans les profondeurs de notre éducation* » [Mor 87]. Cette remarque reste malheureusement encore d'actualité car, pour diverses raisons, de nombreux responsables industriels ne considèrent pas la maîtrise de l'information comme un avantage concurrentiel mais comme un « passage obligatoire » à emprunter pour satisfaire les exigences de l'environnement (fournisseurs, clients, etc.).

Dans ce paragraphe, nous revenons dans un premier temps sur les notions de donnée, d'information et de connaissance. Dans un second temps, un rapide historique du concept de système d'information nous permet d'identifier les principaux concepts issus des travaux réalisés dans ce domaine. Enfin, nous identifions les principaux concepts liés à la notion de système d'information.

### *2.2.3.1 Donnée, information et connaissance*

Si la perception d'un signal, son interprétation et son utilisation sont réalisées quasi simultanément par un même individu, la distinction entre ces notions est importante au sein d'une organisation où règne la division du travail car les activités créant les données peuvent être différentes de celles qui les utilisent.

#### *2.2.3.1.a. Donnée*

Une **donnée** est une image de la réalité dans un code convenu qui est définie selon un besoin et qui permet ainsi de conserver une trace de la réalité pour l'exploiter dans les travaux d'aujourd'hui ou de demain. En d'autres termes, ce sont des représentations d'un phénomène réel qui sont distinctes de la réalité et qui n'existent que par rapport à une intention (conservation, communication ou interprétation).

#### *2.2.3.1.b. Information*

Une **information** pour un individu (ou un groupe d'individus) est un signal qui produit un effet sur son comportement ou son état cognitif [Mel 72]. En ce sens, l'information est l'événement déclencheur du processus de décision qui, selon H. A. Simon [Sim 83], se caractérise par trois phases liées à la résolution d'un problème :

- intelligence : identifier et modéliser le problème,
- conception : formalisation et évaluation des plans d'actions possibles,
- sélection : décision d'action, de réflexion, d'information ou de finalisation.

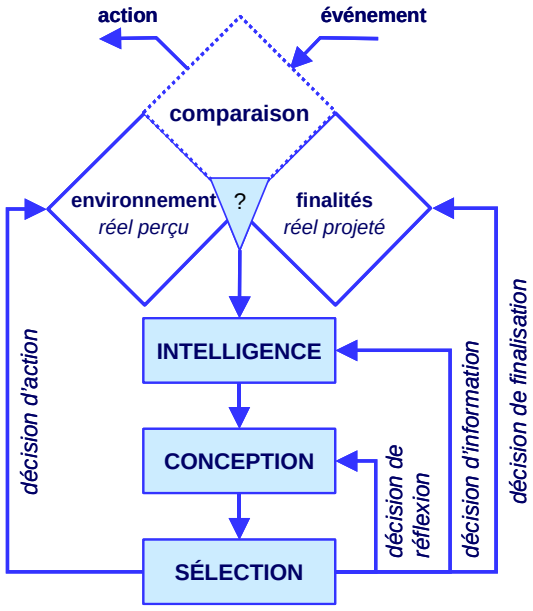



*Figure I.7. : Processus de décision*

Ce processus se déroule lorsqu'une différence entre la description du réel perçu (représentation de l'état courant du système et de son environnement) et du projet conçu (représentation de l'état attendu) est détectée.

Il existe également des activités collectives qui rendent nécessaires le partage des décisions. Ce partage est réalisé suivant deux logiques dominantes [Rei 00] :

- une logique de niveau d'importance définie selon une dimension verticale basée sur le principe de hiérarchie qui établit une logique de pouvoir en subordonnant les décisions d'un niveau à celles du niveau supérieur,

- une logique de différenciation qui justifie un découpage selon une dimension horizontale basée selon le principe de séparabilité qui détermine les besoins de coordination au sein d'un même niveau.

Le principe de séparabilité distingue la séparabilité parfaite à travers laquelle la somme des optimums locaux est égale à l'optimum global, de la séparabilité imparfaite induisant des interdépendances entre les activités :

- interdépendance indirecte : les activités utilisent des ressources partagées,
- interdépendance séquentielle : le résultat d'une activité est une autre activité,
- interdépendance réciproque : échange entre les activités.

Dans ce cas, l'optimum global n'est pas la somme des optimums locaux et la coordination s'impose pour éviter des dysfonctionnements graves (décisions locales inadaptées, conflits, absence de décision).

L'évolution des techniques, et plus particulièrement celle liée aux technologies de communication, a induit une évolution des différentes architectures de pilotage construites à partir de ces deux principes [Tre 96]. Nous nous appuyons sur la notion de centre de décision et de centre de production pour caractériser ces différentes structures. Une étude, présentée au sein de l'annexe 1, décrit plus en détail les caractéristiques de ces différentes structures.

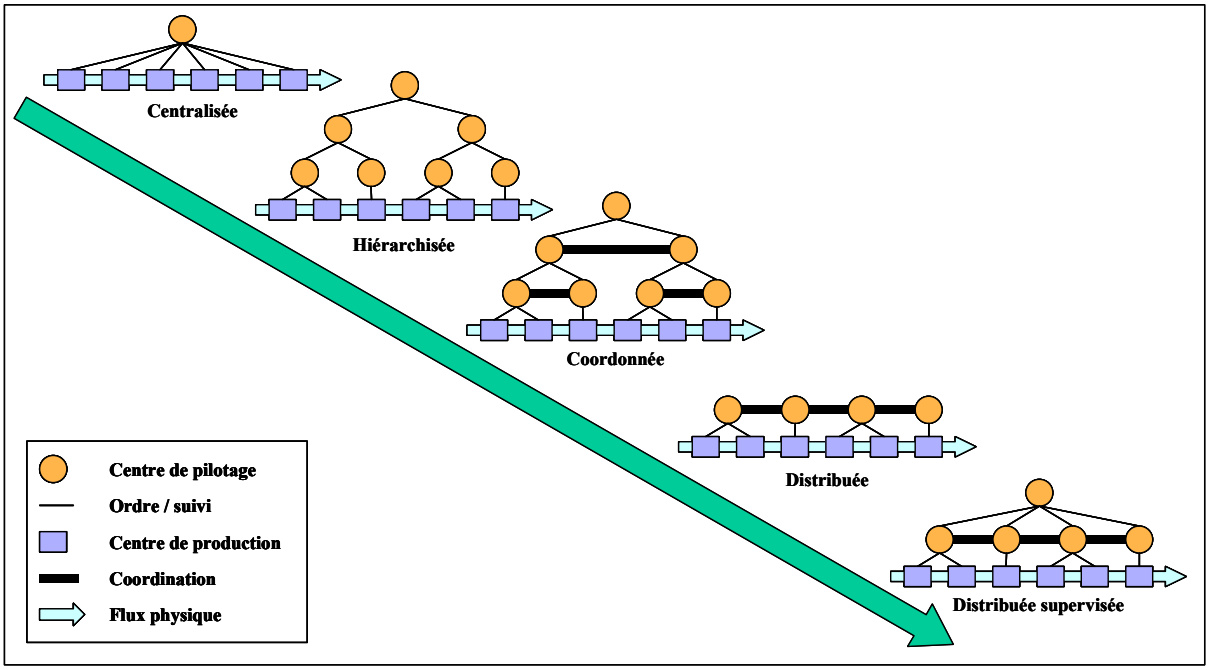



*Figure I.8. : Organisation des centres de décision*

*2.2.3.1.c. Connaissance*

La **connaissance** peut être définie comme un ensemble de schémas (concepts, objets, programmes, séquences d'action, etc.) permettant à un individu de réaliser [BrH 95] :

- des activités de compréhension : attribution d'une signification à un ensemble de symboles,
- des activités de raisonnement : élaboration de conclusions à partir de prémisses,
- des activités d'évaluation : appréciation en fonction d'une référence.

Il est possible de distinguer deux types de connaissance :

- la connaissance formalisée ou explicite qui peut se transmettre par le biais de différents discours,
- la connaissance tacite qui s'acquiert par la pratique.

Le système de gestion des connaissances possède trois types de mémoire [Teb 93] :

- la mémoire de travail (de grande capacité) qui contient les différentes représentations nécessaires à la résolution d'un problème,
- la mémoire à court terme qui filtre et identifie les représentations et traite les actions réflexes,
- la mémoire à long terme qui stocke les connaissances organisées.

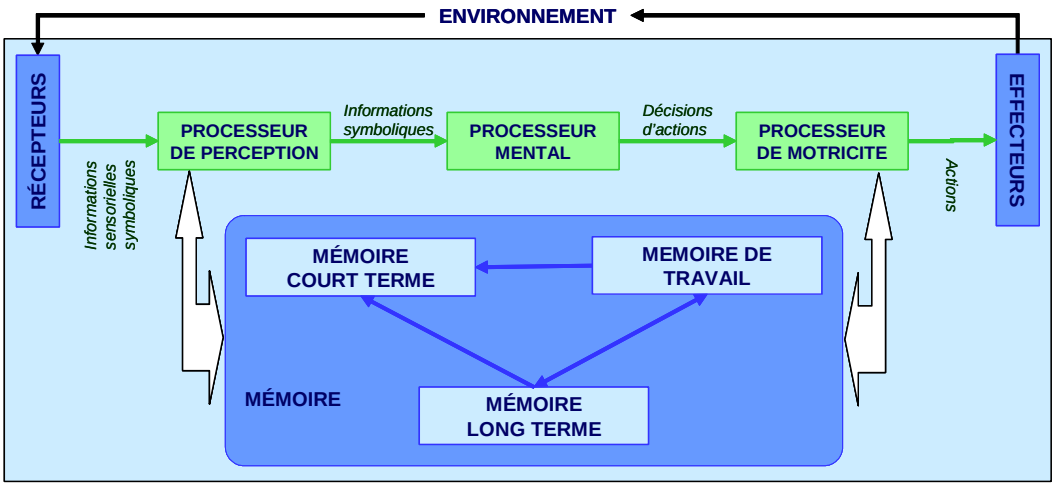



*Figure I.9. : Système cognitif*

### *2.2.3.1.d. Relation entre donnée, information et connaissance*

La figure suivante montre les relations qui existent entre donnée, information et connaissance.

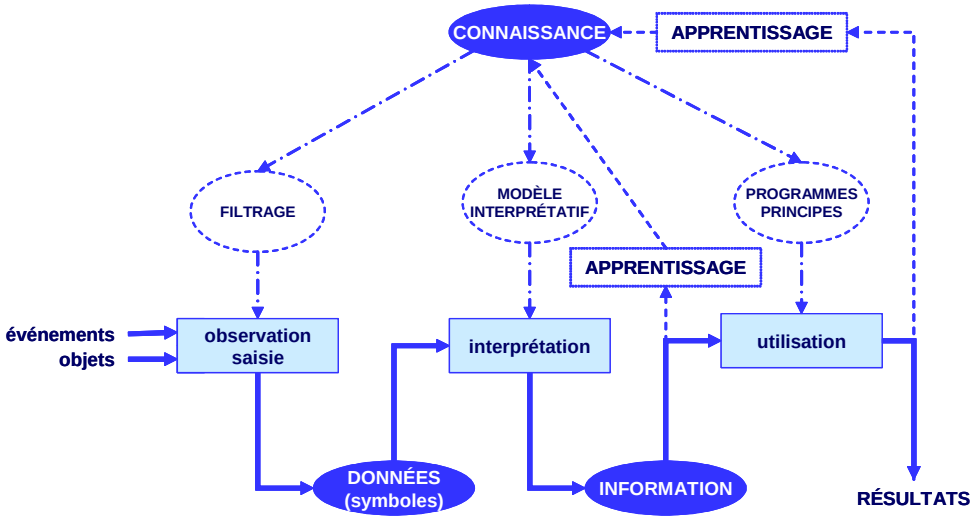



*Figure I.10. : Donnée – Information – Connaissance [Rei 00]*

Ayant présenté ces différents concepts, nous allons maintenant décrire l'évolution du concept de système d'information afin d'identifier les principales caractéristiques issues de ces travaux.

#### *2.2.3.2 Evolution du concept de système d'information*

Le concept de système d'information apparaît au début des années 70 avec la proposition de G.-B. Davis [Dav 74] qui introduit le concept de MIS – *Management Information System* – comme étant le système qui fournit les informations supportant les opérations, la gestion et les prises de décision dans une organisation sociale. Une des principales critiques de cette approche est qu'elle ne traite pas de la production et de la mémorisation des informations. Aussi, J.-L. Le Moigne [LeM 77] s'appuie sur une structuration du système en neuf niveaux de complexité croissante pour introduire un modèle canonique, articulé sur trois niveaux et désigné modèle OID (*Opérant/Information/Décision*) pour représenter un système :

- le système opérant transforme des intrants en extrants (sorties) en fonction d'une finalité donnée,
- le système de décision élabore des commandes (décisions d'action) en fonction d'informations de suivi,
- le système d'information acquiert, mémorise et transmet (sous forme symbolique) :
  - le comportement du système opérant au système de décision,
  - les actions à réaliser par le système opérant définies par le système de décision.

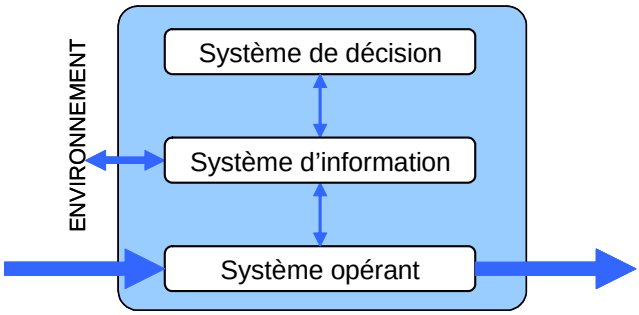



*Figure I.11. : Modèle canonique OID [LeM 77]*

De plus, cette approche distingue la vision synchronique[5] de l'organisation qui rend compte de la production des informations de la vision diachronique[6] qui rend compte de la mémorisation des informations. L'apport principal de J.-L. Le Moigne est d'avoir mis l'accent sur la notion de système d'information organisationnel en s'appuyant sur les trois idées suivantes [TaT 87] :

- mettre les acteurs de l'organisation en situation de formuler eux-mêmes les représentations collectives et stables de leur activité,
- établir un schéma de référence basé sur un langage accessible par les utilisateurs et les constructeurs (conception, développement, maintenance) du système d'information,
- désaccoupler les processus opérationnels, informationnels et décisionnels.

J. Melèse critique l'approche proposée par J.-L. Le Moigne [Mel 79], en s'appuyant sur deux considérations. D'une part, il considère que la notion de « système d'information » concerne uniquement les parties automatisées du système opérant, du système d'information et du système de décision et que, de ce fait, l'information sur leurs « variables essentielles » a quasiment disparu. D'autre part, il montre qu'il existe une confusion entre système d'information et système de donnée du fait de « présupposer qu'il existe un niveau conceptuel indépendant de tout besoin des utilisateurs, donc invariant par rapport aux acteurs internes et

[5] Synchronie : Période durant laquelle nous pouvons considérer le système sans évolution de structure pouvant avoir une influence sur son fonctionnement ou son comportement (étude de la structure et des fonctions).
[6] Diachronie : Période permettant d'observer les évolutions de structure (étude de l'évolution) [LeG 92].

externes, renvoie à la notion de données (objectivité, mesure, répétabilité) et non à celle d'information (représentation, cognition, signification) car l'information est inséparable du sujet et du contexte ».

Il propose une nouvelle approche du système d'information en identifiant les modalités d'organisation, de comportement, d'apprentissage qui favorisent la capacité pour un individu, ou un groupe d'individus, de percevoir, de reconnaître et de traiter l'information. Il aboutit donc à la structuration suivante pour caractériser le fonctionnement informationnel d'une unité d'organisation :

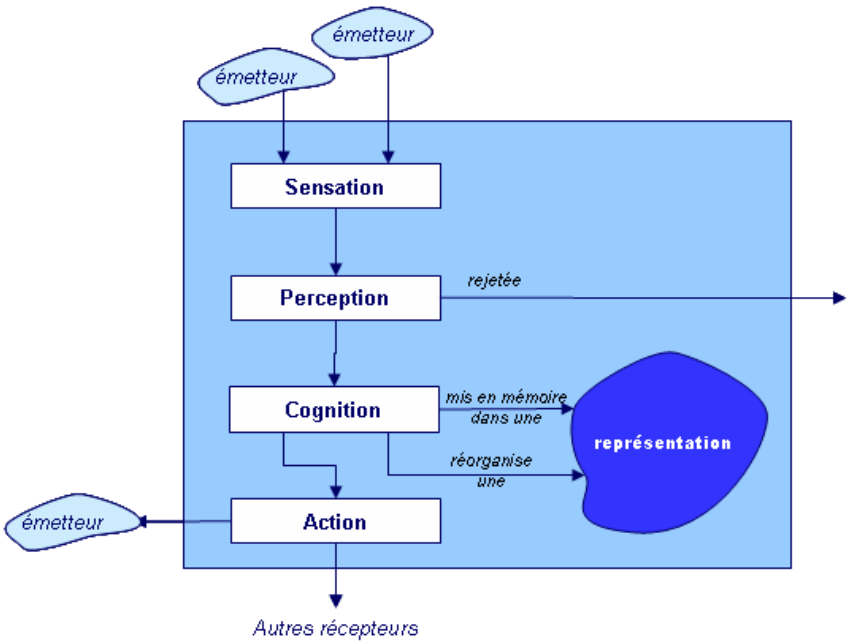



*Figure I.12. : Modèle proposé par J. Melèse*

Dans le schéma précédemment présenté :

- La sensation correspond à l'exposition aux signaux.
- La perception extrait des signaux perçus comme une représentation.
- La cognition reconnaît, comprend, interprète la représentation ; Elle peut conduire à mémoriser la représentation, réorganiser une représentation ou à déclencher une action.
- L'action (ou réaction à la sensation) peut être propre au récepteur ou conduire à diffuser les signaux à d'autres récepteurs.

En d'autres termes, J. Melèse considère que chaque unité organisationnelle doit être susceptible d'extraire de l'ensemble des sensations perçues (images, sons, données, etc.), les informations nécessaires à la régulation des différents flux circulant au sein de l'entreprise.

Les travaux sur la planification et la compréhension [Wil 83] ont ouvert une nouvelle voie dans l'étude des couplages entre système d'information et pilotage. Ces travaux ont montré que le système de pilotage doit traiter les aspects cognitifs au même niveau que les aspects téléologiques. Aussi, nous nous appuyons sur cette approche pour identifier les principales caractéristiques d'un système d'information.

#### *2.2.3.3 Caractéristiques du système d'information*

Nous nous appuyons sur les résultats issus de la génération de plans pour identifier les principales caractéristiques du système d'information. Cette approche montre qu'il existe un lien entre [TaT 87] :

- la résolution de problèmes mal structurés,
- la compréhension,
- la décision,
- la planification.

Un point commun à ces activités est l'utilisation d'une mémoire qui contient les représentations que l'acteur (ou le groupe d'acteurs) s'est forgées avec son expérience. Ces représentations mélangent à la fois des plans d'actions[7] et des situations existantes, elles sont appréhendées dans la génération de plans en robotique à travers [BHM 92] :

- le modèle d'actions qui est basé sur une typologie des actions permettant de représenter les potentialités des ressources en intégrant les aspects fonctionnels, organisationnels et comportementaux dans un environnement donné,
- le modèle de l'univers qui permet de représenter les éléments en fonction :
  - des actions qu'ils peuvent subir,
  - de leur utilisation durant la réalisation d'une action,
  - des contraintes qu'ils exercent sur la réalisation d'une action.

De plus, cette approche considère que le pilotage doit à la fois résoudre des problèmes mal structurés, décider et planifier. En conséquence, piloter, c'est identifier des situations mal structurées auxquelles sont associés des buts. Ces buts donnent lieu à l'établissement de plans d'actions qui sont mis en œuvre par le déroulement de processus. La capacité de résolution de problème doit être permanente et autonome car les plans d'actions doivent pouvoir être révisés lorsque les situations perçues sont identifiées comme ne permettant pas d'atteindre le but initial.

Enfin, les travaux menés dans cet axe montrent que le pilotage doit également être capable d'expliciter un plan d'actions à d'autres acteurs. Ainsi, ces autres acteurs pourront inférer des prédictions en fonction de ces plans d'actions et ils seront capables d'identifier les compétitions (interactions négatives) ou les coopérations (interactions positives) avec leurs propres buts. En d'autres termes, la connaissance de l'organisation se construit par l'élaboration de représentations s'appuyant sur la compréhension qu'ont les acteurs de cette organisation.

Ce constat nous a conduit à nous appuyer sur un nouveau paradigme du système d'information d'une entreprise [BHB 95]. Le **système d'information** doit gérer des processus en s'appuyant d'une part, sur un système de **gestion de ressources** (SGR) qui fournit les données nécessaires au déroulement des processus et, d'autre part, sur un **système de gestion des connaissances** (SGO) pour tenir compte du savoir-faire de l'entreprise. Cette vision est illustrée par la figure suivante.

[7] Ces plans peuvent être linéaires (totalement ordonnés), non linéaires (partiellement ordonnés), hiérarchiques ou non hiérarchiques [Mou 90].

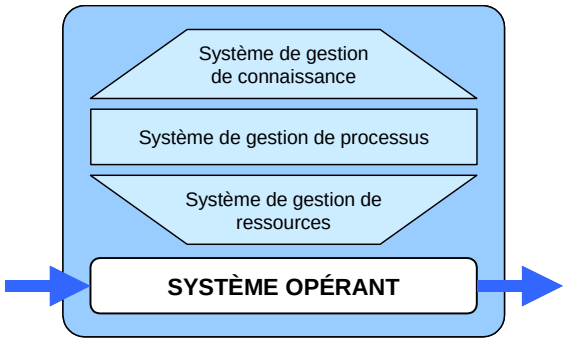



*Figure I.13. : Le paradigme du système d'information*

*2.2.3.3.a. Le système de gestion de ressources*

Ce système est la mémoire nécessaire au pilotage de l'entreprise (pilotage des processus). La description des ressources de l'entreprise définit le modèle de l'univers tandis que les actions supportées par ces ressources constituent le modèle des actions.

*2.2.3.3.b. Le système de gestion de processus*

Ce système gère le déroulement et l'évolution des processus de l'entreprise. Ce système nécessite un modèle de processus pour les représenter et les piloter. Il s'appuie d'une part, sur le système de gestion des connaissances pour définir les éléments nécessaires à l'évolution d'un processus et au paramétrage des activités à réaliser et, d'autre part, sur le système de gestion de ressources pour appréhender les situations existantes et connaître les potentialités de l'entreprise.

*2.2.3.3.c. Le système de gestion de connaissance*

Ce système s'appuie sur les connaissances de l'entreprise pour gérer les objectifs de chaque processus en accord avec la stratégie de l'entreprise. De plus, ce système doit définir l'enchaînement des objectifs propres aux activités d'un processus.

Ayant présenté les différents systèmes composant un système d'information, nous allons maintenant nous intéresser à leur modélisation.

# 3. Le paradigme d'ambivalence

L'évolution économique et industrielle requiert de la part de l'entreprise une agilité lui permettant d'une part, de faire face aux évolutions de l'environnement et, d'autre part, de s'appuyer sur des applications logicielles communicant de manière efficace.

Ce constat induit la nécessité d'appréhender l'entreprise à travers deux perceptions :

- une perception organisationnelle qui s'appuie sur l'enchaînement des activités des processus métiers qui visent, grâce à leur déroulement, à l'atteinte des objectifs de l'entreprise. Cette perception que nous désignons **vision « métier »** fournit ainsi une représentation dynamique de l'entreprise,
- une perception des données qui sont créées, modifiées ou utilisées par les processus. Cette perception que nous désignons **vision « donnée »** fournit une vision unifiée des données[8] circulant au sein de l'entreprise.

[8] Dans le cadre de nos travaux, nous nous intéressons plus particulièrement aux données créées, transformées et transmises par des applications informatiques.

Ces deux visions concernent deux familles d'acteurs :

- les responsables métiers qui utilisent les applications informatiques pour piloter les grandes fonctions de l'entreprise (commercial, marketing, production, distribution, recherche et développement, ressources humaines, finances, etc.),
- les techniciens de l'information qui conçoivent et construisent les applications informatiques (architectes, concepteurs, analystes, programmeurs, etc.) supportant tout ou partie du système d'information de l'entreprise.

Le **paradigme d'ambivalence** considère que ces deux visions représentent la même réalité opérationnelle selon des points de vue et des concepts différents, mais ni disjoints, ni contradictoires [Che 03]. Nous considérons que les points de vue « métier » et « donnée » définissent un ensemble de concepts complémentaires et suffisants pour représenter la réalité opérationnelle de l'entreprise. Cette notion induit que la prise en compte de la réalité « totale » de l'entreprise nécessite l'introduction d'autres points de vue (financier, sociologique, etc.).

Il est à noter que, si le paradigme d'ambivalence s'applique aux points de vue « métier » et « donnée », il doit également s'appliquer aux représentations réalisées pour mieux analyser et agir sur chacune d'entre elles. Cela signifie, en conséquence, que toute modification réalisée sur une représentation se répercute immédiatement sur l'autre.

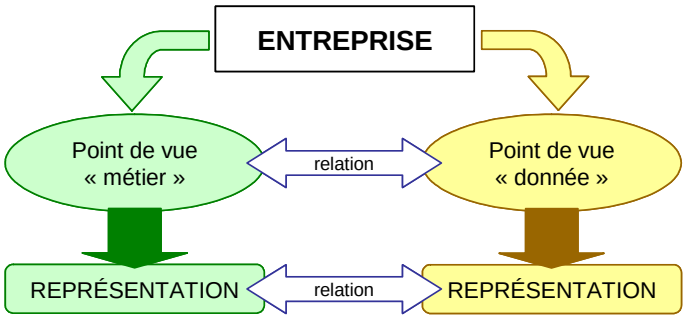



*Figure I.14. : Ambivalence de la réalité opérationnelle de l'entreprise et de ses représentations*

L'illustration de ce paradigme peut s'effectuer par analogie à la perception visuelle d'un être humain. Lorsque nous observons un paysage, nous recomposons mentalement sa réalité en superposant ce que notre œil droit et notre œil gauche nous transmettent via le nerf optique pour en saisir le relief, les distances, la position relative des choses, etc. En fait, chacun de nos yeux perçoit une image légèrement différente du paysage mais ces deux représentations ne sont pas dissociées et seule la juxtaposition de ces deux images nous permet d'approcher la réalité du paysage. A chaque modification dans l'image perçue par un œil, un changement équivalent est perçu également par l'autre œil. Autrement dit, ce qui est vrai dans ce que voit notre œil droit l'est aussi dans ce que voit notre œil gauche et inversement.

Selon la finalité, la représentation du paysage sera différente. S'il s'agit de fournir une représentation pour s'orienter dans une région, il est nécessaire d'identifier les points remarquables du paysage, de les situer les uns par rapport aux autres et d'indiquer les voies de communication qui les relient. Dans le cas de la description d'un site particulier, une représentation détaillée est nécessaire.

La vision globale est indispensable à la compréhension d'un point de vue, mais une vision détaillée est nécessaire pour intervenir en son sein.

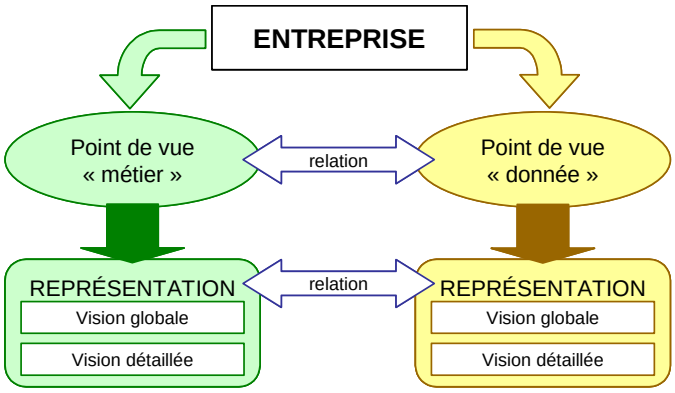



*Figure I.15. : Vision globale et détaillée d'un point de vue*

Le paradigme d'ambivalence se fonde sur l'intuition que les deux visions majeures que nous percevons de la réalité opérationnelle de l'entreprise sont liées et se correspondent comme les perceptions de chacun de nos yeux. Lorsqu'une entreprise décide de résoudre un problème selon un seul de ces points de vue, sans tenir compte de l'autre, le résultat est limité et provisoire. Par exemple, citons les échecs des approches BPR – *Business Process Reengineering* – lorsque la démarche s'est déroulée sans une implication du système d'information. De même, lorsque les besoins et les attentes de l'entreprise ne sont pas pris en compte, les applications informatiques sont la plupart du temps inadaptées.

Dans ce paragraphe, nous nous intéressons dans un premier temps aux conséquences de la séparation de ces deux points de vue. Dans un second temps, nous décrirons les trois principes issus du paradigme d'ambivalence.

## 3.1. La séparation des points de vue : conséquences et opportunités

L'évolution économique et industrielle requiert, de la part de l'entreprise, une efficacité opérationnelle. Cette efficacité opérationnelle est perçue par les clients à travers la qualité des produits et des services qu'ils achètent. Elle constitue l'une des mesures de la capacité managériale des entreprises. Elle est enfin surveillée par les actionnaires qui jugent, notamment à travers elle, du bon usage et de la rentabilité de leurs investissements.

Or, parmi les différentes composantes de cette efficacité, le délai de mise sur le marché de nouveaux produits ou services – *time to market* – fait aujourd'hui l'objet d'une attention particulière au sein des entreprises de façon à réduire les temps de réponse aux sollicitations de l'environnement. Cette évolution induit la mise en place de nouvelles méthodes et de nouveaux langages (Cf. § 2.2.4) pour décrire les relations avec les clients et les fournisseurs, les processus à piloter, les produits et les services et les facteurs de production. Parallèlement, le développement des technologies de l'information a nécessité des connaissances de plus en plus spécialisées pour construire des applications de plus en plus complexes. La maîtrise de ces développements requiert la connaissance de langages spécifiques. Cet accroissement de la complexité induit des temps de développement de plus en plus long. Il est donc apparu au fil du temps une séparation des cycles et des langages entre ces deux points de vue.

Il est à noter que cette séparation a été vécue de façon naturelle et progressive et sans contrainte. Elle n'a jamais été remise en cause car elle était potentiellement porteuse de gains importants de productivité [CHE 03].

### 3.1.1. La séparation des cycles

Au moment le plus fort du taylorisme qui a précédé la montée en puissance des technologies de l'information, les visions « métier » et « donnée » étaient liées et les individus chargés de contribuer aux processus étaient également chargés de traiter et véhiculer l'information non encore réellement automatisée. La parcellisation et l'extrême spécialisation des travaux se sont également appliquées au découpage des flux d'information le long des processus de fabrication et de gestion. L'évolution de l'environnement des entreprises et des technologies de l'information a progressivement engendré une séparation de ces cycles.

Le point de vue « métier » s'appuie sur différentes activités de conception, de développement et de distribution d'un nouveau produit ou d'un nouveau service. Dans la phase amont, les stratèges, les spécialistes du marketing et les chefs de produits traitent de l'émergence des idées, de la conception et de l'industrialisation des produits et des services, tandis que les responsables des services assurant la réalisation du produit ou du service (achat, fabrication, logistique, vente, distribution, etc.) interviennent dans la phase aval de création.

Le cycle d'évolution de la vision « métier » est court car le marché l'exige et la survie de l'entreprise en dépend. Depuis plusieurs dizaines d'années, la période de ce cycle est passée de plusieurs semestres à quelques semaines dans les entreprises à forte croissance et à forte pression concurrentielle.

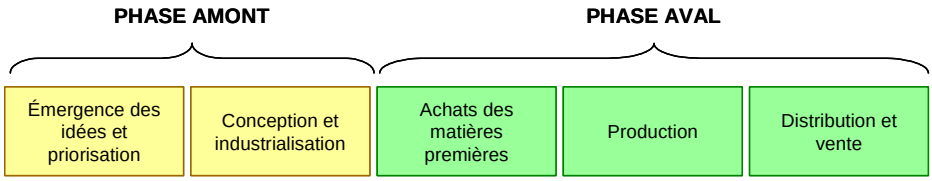



*Figure I.16. : Cycle de création d'un nouveau produit ou service au sein de la vision « métier »*

Le point de vue « donnée » s'appuie, quant à lui, sur des activités propres au développement ou à l'acquisition d'applications informatiques. La phase aval traite des besoins des utilisateurs en termes d'applications informatiques tandis que la phase amont assure la réalisation (conception et développement) ou l'acquisition de l'application, son intégration et son exploitation.

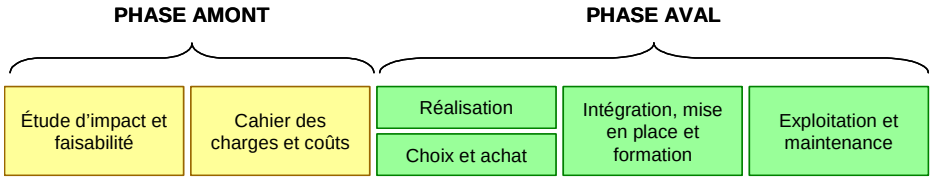



*Figure I.17. : Cycle d'évolution des applications de la vision « donnée »*

L'efficacité et la productivité des acteurs impliqués dans ce point vue a évolué du fait de l'utilisation de nouvelles méthodes d'analyse et de développement, de nouvelles technologies de traitement et de transmission de l'information, et de nouveaux outils de programmation. Malgré cette évolution, le cycle d'évolution du point de vue « donnée » est long du fait de la complexité des applications informatiques.

Ce décalage entre ces cycles est à la fois la cause et la conséquence de la séparation de ces deux points de vue. Il en est la cause car seul un découplage de ces cycles, c'est-à-dire leur désolidarisation et leur gestion asynchrone, permet de pallier les différences de période. Mais

ce décalage est également la conséquence de la séparation des points de vue puisqu'il a engendré, au fil des années, des logiques et des stratégies de plus en plus différenciées. La réduction du décalage entre les cycles nécessite un organe ayant certaines fonctionnalités spécifiques. Les caractéristiques de cet organe peuvent être décrites par analogie à un autre domaine technique. En mécanique automobile, il existe ainsi un mécanisme de transmission qui relie le moteur aux roues d'un véhicule. Ce mécanisme est composé :

- d'un embrayage qui assure la différence de vitesse de rotation entre la sortie du moteur (cycle à période courte) et les roues motrices (cycle à période longue),
- d'une boîte de vitesse qui permet d'ajuster, par paliers successifs, la vitesse de rotation du moteur à celle du véhicule.

Dans le cadre de ces travaux, nous nous appuyons sur cette analogie pour introduire un mécanisme de découplage des deux points de vue considérés.

### 3.1.2. La séparation des langages

La montée en puissance des technologies de l'information a nécessité l'intervention de techniciens extrêmement spécialisés et qualifiés qui ont pour mission l'automatisation de certains travaux répétitifs. De ce fait, les activités d'analyse et de programmation sont devenues incompréhensibles et non maîtrisables pour les utilisateurs à l'origine des demandes. L'incommunicabilité de ces deux populations a été maintes fois identifiée comme étant la cause probable de dysfonctionnements. En effet, la fourniture d'applications informatiques comporte très souvent de nombreuses lacunes, des erreurs et des non-sens. L'implication des utilisateurs dans le cycle de développement des applications n'a pas permis à ce jour de résorber les difficultés rencontrées. Ce constat montre la nécessité de disposer d'une base commune de dialogue. Les futurs utilisateurs d'un programme informatique et ceux chargés de le développer ont les mêmes difficultés de communication qu'un russe et un mexicain qui ne parleraient que leur langue maternelle. Ils ne pourraient résoudre leur incommunicabilité qu'en ayant recours :

- soit à une troisième langue, l'anglais par exemple, que l'un et l'autre auraient appris,
- soit au service d'un interprète qui posséderait une bonne maîtrise du russe et du mexicain.

Un langage commun doit donc, pour être compréhensible et partagé par les deux points de vue, fournir une correspondance entre les concepts de niveau macroscopique et ceux de niveau microscopique.

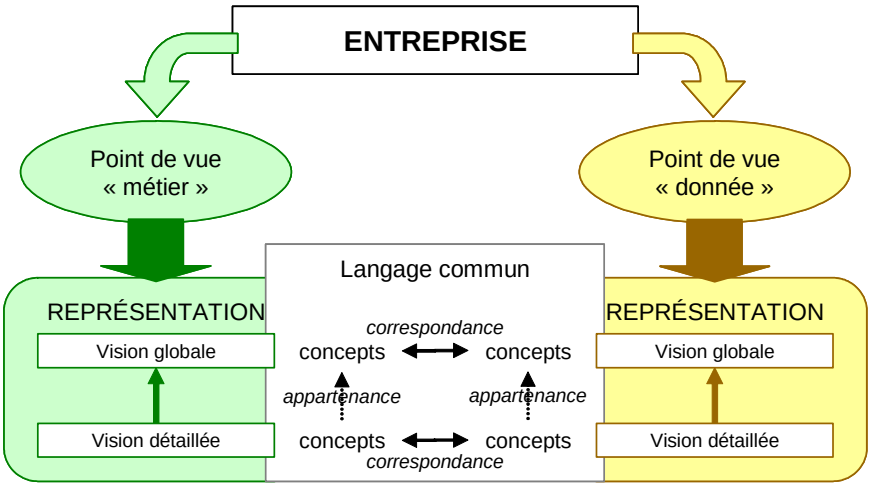



*Figure I.18. : Rôle et positionnement du langage commun*

## 3.2. Les trois principes issus du paradigme d'ambivalence

L'application du paradigme d'ambivalence aux points de vue « métier » et « donnée » de la réalité opérationnelle de l'entreprise a fait émerger trois principes qui doivent être respectés lors de la définition du langage commun [Che 03].

Le premier principe considère que : ***Ce qui s'applique au point de vue « métier » s'applique à celui du point de vue « donnée »***. La réalité ambivalente de l'entreprise conduit à considérer les deux points de vue comme deux perceptions différentes d'une réalité unique. En conséquence, ce qui s'applique à l'une s'applique à l'autre et inversement. Il s'agit donc d'un principe de correspondance entre les concepts de chacun des points de vue. Ce principe est une contrainte car il induit que tout concept retenu pour décrire un point de vue a une correspondance dans l'autre. Si ce n'est pas le cas, le concept est considéré comme non pertinent pour la réalité opérationnelle.

Le second principe peut se résumer par la phrase suivante : ***La représentation de chaque point de vue doit se faire à la fois à un niveau global et à un niveau détaillé***. Comme pour notre perception visuelle, la représentation de la réalité opérationnelle de l'entreprise est structurée en deux niveaux. Le niveau global fournit un schéma général construit avec une famille spécifique de concepts, tandis que le niveau détaillé affine le schéma général en s'appuyant sur une autre famille de concepts.

Enfin, le dernier principe considère que : ***Au sein d'un point de vue, tout concept de représentation détaillée a pour vocation de décomposer le contenu d'un unique concept de représentation globale***. Ce principe nous incite donc à initialiser la recherche de concepts de représentation par le niveau global puis d'identifier les concepts du niveau détaillé qui seront les plus à même de les décrire de manière plus précise et plus adaptée. Cette correspondance est assurée par un mécanisme spécifique désigné **raffinement par partition** [Bol 04] qui établit une correspondance entre un concept du niveau global et des concepts du niveau détaillé. Ce mécanisme permet donc de traduire toute évolution du niveau global dans le niveau détaillé. De plus, le paradigme d'ambivalence suggère de rechercher en permanence une correspondance entre les représentations des deux points de vue à chacun des niveaux. Dans le cadre de nos travaux, cette correspondance est assurée par un mécanisme spécifique désigné **raffinement par réécriture** [Bol 04] qui établit une correspondance entre les concepts des deux points de vue. Ce mécanisme permet donc de traduire toute évolution d'un point de vue dans l'autre point de vue.

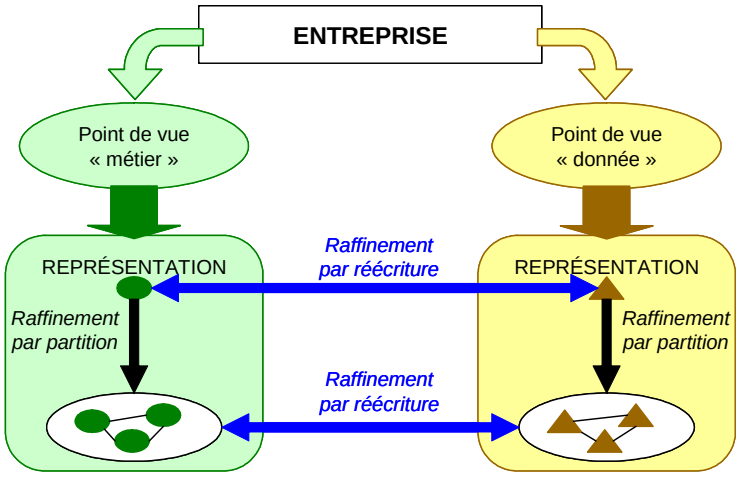



*Figure I.19. : Les mécanismes de raffinement*

La représentation de la réalité opérationnelle à travers deux points de vue induit l'implication de deux domaines d'études spécifiques que sont le génie industriel et le génie logiciel. De plus, la vision détaillée (du point de vue « métier » ou « donnée ») engendre un nombre important d'éléments à représenter pour caractériser la réalité opérationnelle d'une entreprise. Ces deux constats nous conduisent à représenter les deux points de vue par des architectures qui structurent ces éléments. Aussi, dans le cadre du paragraphe suivant, nous présentons le concept d'architecture et nous identifions les architectures spécifiques à chacun des points de vue.

## 3.3. Des représentations centrées « architecture »

L'architecture d'un système est un terme qui est fondé sur une analogie qui nous semble être d'un emploi justifié quand les points suivants sont vérifiés [Dar 04] :

- Il n'est pas possible de représenter entièrement le système du fait d'une part, du nombre d'éléments à appréhender et, d'autre part, des multiples façons de le représenter. La représentation du système est alors basée sur la définition, puis l'assemblage de composants.
- Il est nécessaire de faire des ajustements lors des différentes étapes de réalisation pour corriger des lacunes ou des incompréhensions tout en respectant des propriétés attendues telles que la robustesse, la facilité d'utilisation, la performance, la modifiabilité, etc.
- Le système traduit des besoins spécifiques liés au domaine d'intervention, au contexte d'utilisation, aux moyens mis en œuvre, etc. et nécessite une approche progressive décrite par des représentations variées.
- La réalisation nécessite la participation de techniciens différents pour la réalisation des différents éléments composant le système et induit donc la présence d'un coordinateur.

Compte tenu de ce qui précède, une architecture peut être définie comme une configuration de composants et de connecteurs [PeW 92] [BCK 99]. Un ***composant*** correspond à une unité de traitement ou de stockage, il est le constituant de base d'une architecture. Un composant dispose de ***ports*** pour s'interfacer avec son environnement. Un ***connecteur*** formalise les protocoles d'échange entre les différents composants. Un connecteur dispose également de ports. Une ***configuration*** précise l'organisation des composants et des connecteurs en identifiant les connexions entre leurs ports.

Aussi, en nous référant au paradigme d'ambivalence de la réalité opérationnelle de l'entreprise, et notamment au paragraphe relatif à la séparation des points de vue « métier » et « donnée », nous identifions deux types d'architecture :

- les architectures industrielles qui correspondent au point de vue « métier »,
- les architectures logicielles qui correspondent au point de vue « donnée ».

L'application des principes issus du paradigme d'ambivalence se traduit donc de la façon suivante. D'après le premier principe, toute modification ou changement réalisé au sein d'une architecture doit avoir un impact immédiat sur l'autre. Ainsi, toute évolution de l'architecture industrielle pour prendre en compte de nouveaux besoins doit pouvoir être impactée sur l'architecture logicielle. De même, toute évolution de l'architecture logicielle pour prendre en compte de nouvelles fonctionnalités doit pouvoir être impactée sur l'architecture industrielle.

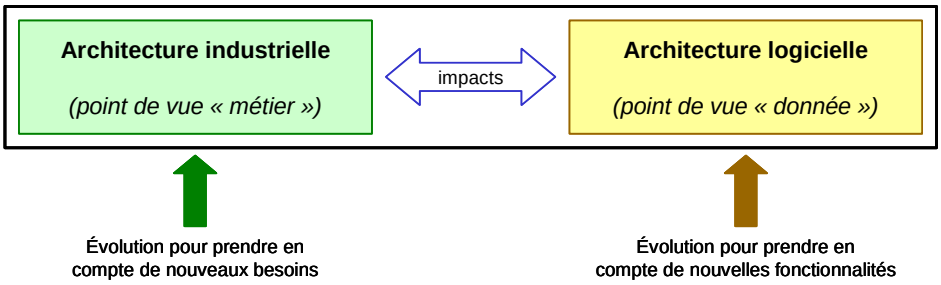



*Figure I.20. : Impact d'un changement au sein d'une architecture*

Le second principe nous conduit à disposer d'un langage d'architecture capable de représenter d'une part, l'entreprise dans son environnement et, d'autre part, les différents éléments participant à son fonctionnement.

Enfin, le dernier principe nécessite l'établissement de correspondance entre les représentations globales et détaillées, et entre chacun des points de vue. Nous utilisons un langage de raffinement centré architecture pour réaliser ces correspondances.

Ayant introduit les concepts d'appréhension de l'entreprise que nous utilisons dans le cadre de nos travaux, nous allons maintenant nous intéresser à la représentation de la réalité opérationnelle d'une entreprise fournie par le cadre de modélisation OLYMPIOS.

# 4. Notre approche modélisante de l'entreprise

Une entreprise peut être perçue comme un système est complexe[9] ce qui implique qu'il est nécessaire de la modéliser afin d'une part, de la rendre intelligible et, d'autre part, d'autoriser le raisonnement d'un acteur ayant un projet bien défini en son sein.

## 4.1. Notion de modèle

Un modèle (d'un phénomène) est une construction intentionnelle, obtenue par composition de symboles, qui permet de [LeM 90] :

- rendre compte de toutes les observations faites,
- prévoir le comportement du phénomène considéré dans des conditions plus variées que celles qui ont donné naissance aux observations.

Un modèle est toujours construit sur la base d'un langage que celui-ci soit informel (langage naturel, par exemple), semi-formel[10] (langage au formalisme essentiellement graphique par exemple) ou formel (langage mathématique). La plupart du temps, les modèles basés sur un langage informel sont utilisés pour décrire une situation existante tandis que les modèles basés sur un langage formel permettent la vérification des propriétés fixées dans un projet donné [CLL 99]. De plus, un modèle est toujours associé à une finalité et il doit, suivant les besoins, être capable de prendre en compte les aspects structurels, fonctionnels et comportementaux. Outre ces aspects, il doit également être capable d'appréhender des points de vue particuliers liés à un individu ou à un groupe d'individus.

[9] Cette notion de complexité induit d'une part, que le système est soumis à des phénomènes incertains et aléatoires et, d'autre part, que l'organisation fait émerger de nouvelles propriétés [LeM 90].

[10] Un langage semi-formel s'appuie généralement sur un vocabulaire et une syntaxe formels (les éléments du langage et les règles syntaxiques sont parfaitement définis) mais ne dispose pas de sémantique lui permettant de vérifier les propriétés comportementales.

## 4.2. Modélisation de l'entreprise

Ces différentes constatations expliquent le fait que les travaux menés dans le cadre de la modélisation d'entreprise aient donné lieu à de nombreux langages et outils de modélisation. Aussi, il est très difficile d'en fournir une liste exhaustive d'autant plus que la plupart des disciplines ayant à traiter des systèmes complexes artificiels ont, peu ou prou, apporté leur pierre à l'édifice. Nous citerons notamment le génie logiciel, l'automatique ou le génie industriel. Dans le cas du génie logiciel, il faut remarquer que celui-ci a toujours été source d'innovation en modélisation d'entreprise, que ce soit au travers de méthodes déjà anciennes telles que MERISE [CHL 89] [NEC 92], plus récentes telles que PROCESS UNIFIED [Mul 97] [FaL 99] ou des travaux en cours tels que ceux menés par l'OMG – *Object Management Group* (http://www.omg.org) ou le WfMC – *Workflow Management Coalition* (http://www.wfmc.org/index.html), qui ont une incidence sur la modélisation d'entreprise, en particulier dans le cadre de l'intégration des applications industrielles. Dans le cadre du génie industriel, nous pouvons citer, parmi les méthodes couramment utilisées, IDEFx [MeM 98], ARIS [Sch 98], GRAI [Rob 93]. Face à cette disparité, des travaux sont actuellement en cours pour définir un langage commun ou générique permettant aux modèles de communiquer. Cette fusion devrait permettre l'émergence d'un langage unifié (UEML) dont la couverture finale sera supérieure à celle assurée individuellement par chacun des langages actuels [VBC 03] [Ver 02]. Une étude détaillée des différents modèles d'entreprise est proposée par le projet UEML – *Unified Enterprise Modelling Language* [UEM 02].

Enfin, une démarche spécifique explicite souvent les différentes étapes nécessaires à la construction et à l'exploitation du modèle (cycle de vie) ; Chaque étape étant souvent caractérisée par un niveau d'abstraction (niveaux conceptuel, organisationnel ou technique).

Nous venons de montrer que la compréhension d'une entreprise nécessite une approche modélisante. Cette approche doit :

- permettre la perception du monde réel (analyse de l'existant) par l'énumération des ressources (produits, acteurs, moyens), des fonctions et des processus,
- représenter l'entreprise comme une entité fonctionnant et se transformant dans un environnement en fonction d'une finalité.

En d'autres termes, toute approche modélisante de l'entreprise doit combiner une double démarche ascendante et descendante [BrH 95].

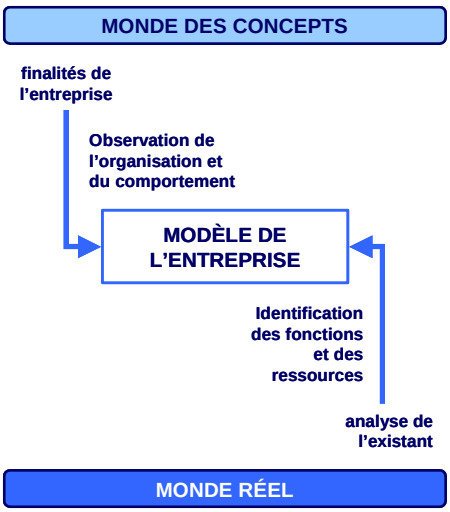



*Figure I.21. : Approche modélisante*

Dans le cadre de nos travaux, nous nous appuyons sur le cadre de modélisation OLYMPIOS qui permet d'appréhender l'entreprise à travers différents points de vue.

## 4.3. Le modèle OLYMPIOS

Le modèle OLYMPIOS [Din 98] [Thé 02] [Bra 02] vise à modéliser le système d'information d'une entreprise en s'appuyant sur la notion de point de vue. Dans un premier temps, cette section présente les concepts de base du modèle OLYMPIOS et décrit ensuite les différents systèmes qui caractérisent le système d'information d'une entreprise.

### 4.3.1. Les concepts d'OLYMPIOS

Avant de présenter les concepts de ce modèle, il nous semble important de rappeler le contexte du modèle OLYMPIOS. La maîtrise de l'entreprise passe par la maîtrise de son système d'information et de décision. Aussi, ce modèle vise à fournir aux industriels une démarche cohérente pour spécifier et construire leur système d'information. De plus, ce modèle intègre différentes vues de l'entreprise. En effet, une entreprise manufacturière peut être modélisée de différentes manières selon la vision qui intéresse le modélisateur et en fonction des problèmes à résoudre. Ainsi, un comptable ayant un objectif tel que la valorisation des stocks d'en-cours et un gestionnaire de production ayant un objectif tel que la diminution de ces mêmes stocks n'ont pas la même vision de l'entreprise et n'utilisent pas le même modèle de représentation.

#### *4.3.1.1 Domaine de visibilité*

Chaque individu (ou groupe d'individus) perçoit l'entreprise à travers l'ensemble des actions qu'il est capable de réaliser. Cette perception définit la notion de domaine de visibilité d'un individu. En effet, un opérateur sur une machine-outil et un régleur (de cette même machine-outil) n'ont pas la même vision de la ressource car ils l'appréhendent à travers des actions différentes. Dans le cadre de nos travaux, nous nous appuyons sur ce concept pour caractériser les points de vue « métier » et « donnée » introduits précédemment (Cf. § 3.1).

#### *4.3.1.2 Relation d'échange*

Un autre concept du modèle OLYMPIOS est celui de la relation d'échange entre les intervenants de l'entreprise. L'entreprise est reliée à son environnement (clients, fournisseurs, institutions) par des relations d'échanges où nous pouvons, en général, identifier un client et un fournisseur. Le modèle OLYMPIOS généralise ce paradigme client-fournisseur aux relations internes de l'entreprise, cette relation se traduit par :

- une expression des besoins de l'utilisateur d'une ressource (client) au fournisseur,
- la fourniture d'une ressource du fournisseur à l'utilisateur.

Ce concept de relation d'échange repose sur les postulats suivants :

- tout utilisateur sait localiser un fournisseur capable de lui fournir la ressource dont il a besoin,
- tout utilisateur sait définir et exprimer ses besoins et les transmettre au fournisseur,
- un processeur est un utilisateur de ressource en vue de fournir une ressource à un autre processeur. Ainsi, aucun processeur ne produit pour produire (réalisation d'une ressource sans expression des besoins) ou ne consomme pour consommer (appropriation d'une ressource non suivie de réalisation).

#### *4.3.1.3 Objectif et niveau de satisfaction*

Le modèle OLYMPIOS considère que chaque relation d'échange est réalisée pour satisfaire un objectif fixé à l'utilisateur. La représentation de cet objectif, au niveau de l'utilisateur, est désignée *ressource attendue*. Suite à une relation d'échange, la représentation de cet objectif, au niveau du fournisseur, est désignée *ressource de référence*. Lorsqu'une ressource est réalisée, le fournisseur décide de la conformité de cette ressource en comparant les caractéristiques de cette dernière à celles de la ressource de référence. Suite à l'appropriation d'une ressource, l'utilisateur est capable d'émettre un niveau de satisfaction en comparant les caractéristiques de la ressource appropriée à celles de la ressource attendue.

#### *4.3.1.4 Représentation des ressources*

Le modèle OLYMPIOS considère que chaque ressource de l'entreprise est représentée par une description informationnelle qui fournit les caractéristiques nécessaires à leur gestion. Ces caractéristiques constituent le modèle d'actions et le modèle de l'univers de l'entreprise (Cf. § 2.2.3.3). Ces caractéristiques doivent intégrer les aspects temporels pour appréhender les problèmes de planification, de traçabilité, de partage, de synchronisation et de coopération.

Ayant présenté les différents concepts du modèle OLYMPIOS, nous allons maintenant décrire les différents systèmes issus de ce modèle.

### 4.3.2. Les divers systèmes du modèle OLYMPIOS

Le système d'information construit à partir du modèle OLYMPIOS s'articule autour des systèmes suivants :

- le système d'information fournisseur – utilisateur (SIFU),
- le système de gestion des objectifs (SGO),
- le système de gestion de ressources (SGR),
- le système d'activation (SA).

#### *4.3.2.1 Le système d'information fournisseur-utilisateur (SIFU)*

Le Système d'Information Fournisseur-Utilisateur [Bra 89], noté SIFU, est le système capable de gérer le concept de relation d'échange. Le SIFU représente l'atome d'organisation du système d'information et de décision. Ce système considère que chaque processeur est capable de mettre en œuvre trois fonctions qui sont à la base des relations d'échange :

- une fonction « *Réalisation* » qui réalise une transformation,
- une fonction « *Fourniture* » qui est capable de fournir une ressource,
- une fonction « *Utilisation* » qui est capable de s'approprier une ressource.

Suite à ces remarques, un SIFU est constitué par :

- un système S1, désigné Fournisseur, mettant en œuvre ses fonctions « *Réalisation* » et « *Fourniture* »,
- un système S2, désigné Utilisateur, mettant en œuvre la fonction « *Utilisation* »,
- une relation d'échange établie entre S1 et S2.

Le fonctionnement d'un SIFU est décrit par le schéma suivant :

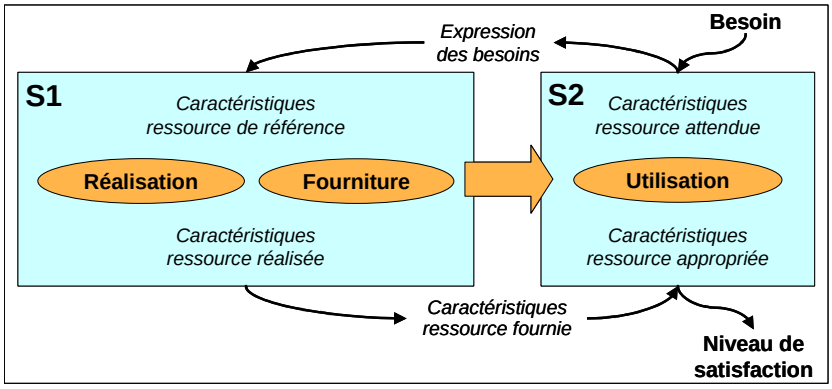



*Figure I.22. : Fonctionnement d'un SIFU*

#### *4.3.2.2 Le système de gestion des objectifs (SGO)*

Le rôle du système de gestion des objectifs est de créer un graphe d'objectifs dont les nœuds sont les besoins des SIFU. Pour ce faire, le SGO s'appuie sur différents graphes existants dans l'entreprise, entre autres :

- le graphe hiérarchique,
- le graphe décrivant les rôles des intervenants de l'entreprise,
- le graphe décrivant les transformations subies par les différents flux,
- etc.

Ce système identifie les interactions positives et négatives entre les différents objectifs.

#### *4.3.2.3 Le système de gestion de ressources (SGR)*

Le système de gestion de ressources est chargé du management de l'ensemble des ressources (logiques ou matérielles) nécessaires à la réalisation des activités. Rappelons que, dans le modèle OLYMPIOS, ces ressources sont perçues à travers un ensemble de caractéristiques dont certaines permettent au SGR de gérer les différents modes de partage (Cf. § 4.3.1.4).

#### *4.3.2.4 Le système d'activation (SA)*

Le système d'activation construit, exécute et modifie les plans d'actions en fonction des objectifs et des disponibilités des ressources. En d'autres termes, ce système contrôle le déroulement des processus mis en œuvre pour atteindre les objectifs définis par le SGO en tenant compte de la disponibilité des ressources transmises par le SGR. La mise en œuvre de ce système peut être concrétisée par un système de gestion de workflow [Bla 01] [BlB 01].

#### *4.3.2.5 Une nouvelle vision du système d'information*

Ces différents systèmes sont à la base de la vision du système d'information introduite précédemment (Cf. § 2.2.3.3). En effet, le SGR gère les ressources de l'entreprise tandis que le système de gestion des objectifs correspond au système de gestion des connaissances. Le SA est, quant à lui, le système de gestion des processus. Nos travaux étant principalement centrés sur ce système, nous allons maintenant décrire le modèle de pilotage des processus proposé par le modèle OLYMPIOS.

### 4.3.3. Le modèle de pilotage des processus du modèle OLYMPIOS

L'objectif du modèle OLYMPIOS est de permettre la formalisation du système d'information d'une entreprise pour lui permettre de réagir aux contraintes qu'elle subit. Cette recherche d'amélioration constante se traduit par la mise en place de **mécanismes d'adaptation** et **d'anticipation** qui sont mis en œuvre par le système d'activation. Ces mécanismes sont à la base du modèle pilotage proposé par le modèle OLYMPIOS [Thé 02].

Un système de pilotage doit permettre de faire évoluer un système vers un but initialement fixé, ceci via différentes situations intermédiaires. A un instant donné, le système évolue d'une situation initiale vers une situation intermédiaire. Pour signifier cette évolution, nous considérons qu'il est nécessaire de s'appuyer sur un **référent** (Cf. § 2.2.2), afin de pouvoir évaluer, en termes de plus ou moins bonne conformité, une situation obtenue, à un instant donné, à une situation espérée. Dans le cas où une non-conformité serait significative, il est nécessaire de construire un nouveau référent capable d'atteindre l'objectif fixé à partir de la situation atteinte.

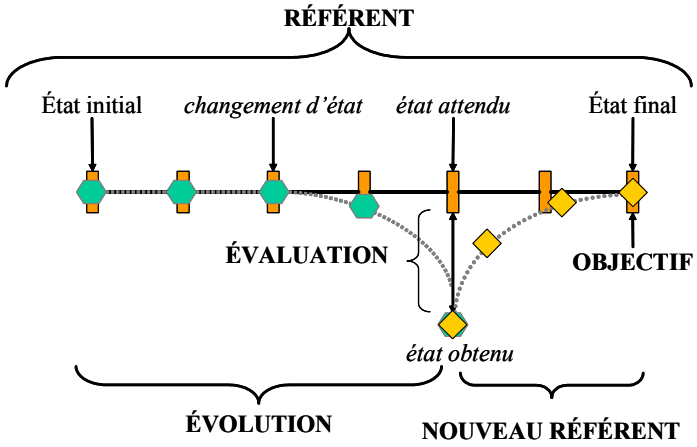



*Figure I.23. : Le pilotage d'un système*

Cette vision permet d'appréhender le pilotage grâce à quatre phases mettant en œuvre différentes fonctions. Ces phases, qui définissent le **schéma de contrôle** du modèle de pilotage, sont les suivantes :

- la **phase initialisation** qui construit le référent capable d'atteindre l'objectif fixé, en définissant les activités et objectifs à réaliser,
- la **phase d'évolution** qui exécute les activités du référent et permet la détection d'anomalie de fonctionnement s'appuyant sur le résultat atteint et l'objectif attendu,
- la **phase de correction** qui, après avoir stoppé l'évolution, corrige les problèmes constatés en définissant un nouveau référent,
- la **phase de terminaison** qui évalue globalement le processus lorsque son état final est atteint.

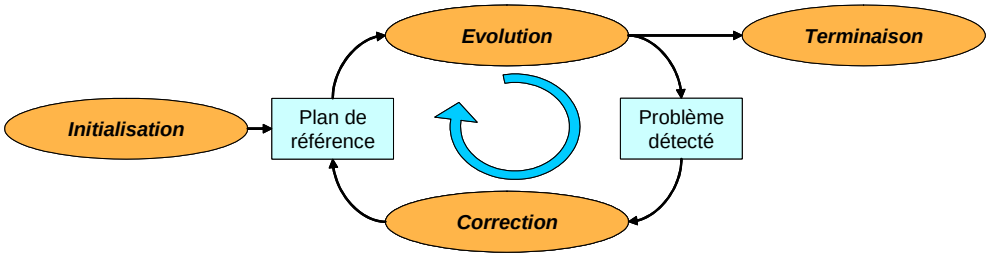



*Figure I.24. : Les phases du modèle de pilotage*

Dans le cadre de nos travaux, nous nous appuyons sur ce modèle pour formaliser le pilotage d'un processus. L'objectif du pilotage est de contrôler le déroulement d'un processus et de déterminer si ce dernier converge ou non vers le but fixé. Dans ce cas, la phase d'**initialisation** construit le référent (plan d'actions de référence) qui précise l'enchaînement des activités à réaliser pour atteindre l'objectif fixé. Cette construction peut s'effectuer de plusieurs façons : planification, apprentissage, adaptation de plans existants ou recherche d'une meilleure pratique. La phase d'**évolution** assure l'exécution des activités et mesure leur résultat en comparant la situation obtenue à la situation espérée qui est définie dans le référent. Suite à la détection d'un dysfonctionnement (non-conformité significative exprimée par la phase d'évolution), la phase de **correction** construit un nouveau référent (plan d'actions) permettant au processus d'atteindre le but fixé à partir de la situation obtenue. Cette construction peut s'appuyer sur les techniques de la phase d'initialisation ainsi que sur les techniques de réparation de plans [MyB 98]. Au terme de la phase d'évolution du processus, la phase de **terminaison** détermine sa performance, c'est-à-dire une valeur de conformité traduisant la plus ou moins bonne atteinte du but.

### 4.3.4. Le composant industriel

Rappelons que nous nous appuyons sur la notion d'architecture pour appréhender la réalité opérationnelle de l'entreprise. Nous introduisons de plus le concept de **composant industriel** pour structurer tout ou partie de l'entreprise [BTH 00] [Cha 02]. En conséquence, un composant industriel est une entité de structuration définie pour satisfaire un besoin unique. Il a un comportement qui lui permet d'atteindre un but qui satisfait plus ou moins le besoin. Le besoin et le but sont les ports du composant, ils sont décrits dans le même univers du discours.

Un composant industriel est caractérisé par les propriétés suivantes :

- il est **atomique** : il doit être appréhendé globalement,
- il est **générique** : il permet la représentation de différentes parties (il tient simultanément compte des aspects structuraux, fonctionnels et comportementaux d'un secteur donné),
- il exprime le **pilotage du processus** mis en œuvre pour réaliser le besoin,
- il est **connectable** à d'autres composants industriels.

Les connecteurs entre ces composants industriels définissent les différents mécanismes d'assemblage. Nous distinguons trois types de mécanismes. Le premier mécanisme d'assemblage est l'assemblage par ports (besoin ou but). Il permet de caractériser différents niveaux au sein d'un domaine en fournissant :

- un assemblage par le besoin : le besoin d'un composant industriel peut être décomposé en plusieurs « sous-besoins ». Ce sont les besoins des composants industriels d'un autre niveau. L'opérateur qui lie les besoins dépend de la décomposition du besoin initial,
- un assemblage par le but : des buts de plusieurs composants industriels, définis au même niveau, peuvent être agrégés pour constituer le but d'un composant industriel d'un autre niveau ; Cet assemblage est réalisé par des opérateurs spécifiques (agrégation, par exemple).

Le second mécanisme d'assemblage considère que le but d'un composant industriel est une ressource nécessaire à un autre composant industriel. Ce mécanisme fait émerger la notion de rôle (client ou fournisseur, par exemple) et traduit une relation d'échange entre composants industriels.

Enfin, il existe des mécanismes d'assemblage par les ressources, qui incluent :

- une coopération entre les composants industriels (synchronisation et envoi de messages),
- une délégation de tout ou partie d'un comportement d'un composant industriel.

Ces éléments sont à la base des représentations construites au sein du point de vue « métier ». Dans le cas d'une vision globale, le composant industriel représente l'entreprise et l'architecture définit alors les interactions entre l'entreprise et son environnement. Dans la vision détaillée, le composant industriel représente un processus métier et l'architecture identifie les coordinations entre ces processus.

Ayant présenté le cadre de modélisation OLYMPIOS, nous allons maintenant nous intéresser à un environnement logiciel centré architecture capable de concrétiser ce cadre de modélisation.

# 5. Le projet ArchWare

Les travaux présentés au sein de cette thèse ont été financés par la Commission Européenne dans le cadre du projet IST ArchWare – *Architecting Evolvable Software* – référencé sous le numéro IST-2001-32360 [Arc 02]. L'objectif principal de ce projet est de répondre à la demande de plus en plus importante des systèmes logiciels qui doivent s'adapter aux différents changements durant leur cycle de vie, et en plus être évolutifs. Afin d'atteindre cet objectif, le projet ArchWare développe un ensemble intégré de langages et d'outils centrés architecture pour l'ingénierie des systèmes logiciels évolutifs en se basant sur un cadre d'exécution persistant. Dans le cadre de ce projet, nos travaux ont plus particulièrement concerné la validation des résultats de ce dernier par le développement d'un environnement capable d'intégrer différentes applications logicielles en vue d'exécuter et de piloter les processus métier.

Le projet ArchWare a donc pour objectif de concevoir, développer, et disséminer :

- un langage innovant centré architecture,
- des styles architecturaux,
- un langage de raffinement,
- des environnements logiciels personnalisables,
- des outils dédiés à l'ingénierie de systèmes logiciels évolutifs.

Le langage centré architecture permet la description formelle d'une architecture. Pour ce faire, il supporte la définition de style architectural et la définition des différents éléments de l'architecture. Un style caractérise une architecture donnée à travers différentes facettes. La première facette précise les propriétés vérifiées par l'architecture. Une deuxième facette fournit les concepts, les éléments de construction et les contraintes que doit vérifier l'architecture du domaine étudié. Enfin, la dernière facette définit une configuration (organisation des composants et des connecteurs).

Les propriétés et les contraintes définies au sein d'un style fixent les règles et les limites de construction de l'architecture. Il existe trois classes de contraintes :

- Les contraintes ***topologiques*** précisent le type des éléments de construction, le nombre d'occurrences possibles au sein de l'architecture et les règles de configuration

contraignant leurs interactions (par exemple, deux composants sont toujours liés par l'intermédiaire d'un connecteur).
- Les contraintes ***comportementales*** définissent d'une part, le comportement des éléments de l'architecture et, d'autre part, les modalités d'évolution de l'architecture (par exemple, il n'y a pas d'interblocage entre les différents composants).
- Les contraintes ***d'attributs*** concernent les aspects non structurels et non fonctionnels d'une architecture. Les attributs apportent des informations complémentaires sur les éléments architecturaux en contraignant leur type, leur nom et leur plage de valeurs (par exemple, la durée de traitement réalisée par un composant ne doit pas dépasser 0.5 seconde). Elles peuvent également spécifier la manière dont les valeurs des attributs sont liées avec d'autres aspects architecturaux (topologie et comportement).

Concernant la notion de raffinement, nous rappelons que le raffinement d'architectures peut être de deux types (Cf. § 3.2). Le premier type est un raffinement par partition qui permet de passer d'un certain niveau de description à un niveau plus détaillé. Le second est un mécanisme de raffinement par réécriture qui permet de passer d'une description architecturale suivant un certain point de vue, vers une description architecturale suivant un autre point de vue. Par conséquent, le langage de raffinement devra permettre la mise en œuvre de ces deux mécanismes.

Le projet ArchWare s'appuie sur la notion de style architectural pour construire des environnements personnalisables. Ces environnements doivent permettre d'appréhender les spécificités d'un domaine (styles architecturaux, cycle de vie de développement, etc.).

Ces différents éléments s'appuient sur des outils spécifiques qui doivent permettre l'ingénierie de systèmes logiciels évolutifs. Ces outils fournissent des supports pour :
- la définition de styles architecturaux,
- la description visuelle d'architectures en s'appuyant sur des profils UML,
- l'animation du comportement des architectures,
- l'analyse des propriétés des architectures,
- le raffinement de descriptions d'architectures,
- la génération de code,
- la personnalisation des environnements.

En résumé, l'objectif principal du projet ArchWare est de permettre le développement de logiciels évolutifs en tenant compte d'une part, des styles architecturaux spécifiques à un domaine et, d'autre part, des composants existants. L'approche développée au sein du projet ArchWare s'appuie sur une vue globale du système représenté par une description formelle de l'architecture en utilisant des styles spécifiques à un domaine. Cette description peut ensuite être raffinée de façon incrémentale jusqu'à obtenir une description concrète qui peut être utilisée pour générer une application.

# 6. Conclusion

Nous avons vu au sein de ce chapitre, que l'entreprise est une entité complexe qu'il est tout à fait légitime de considérer comme un système. Dans le cadre de nos travaux, nous nous intéressons à la formalisation du système d'information permettant de rendre l'entreprise agile en proposant un nouveau paradigme du système d'information qui est centré sur un système de gestion de processus. Le pilotage de ces processus s'appuie sur un système de gestion de

ressources et un système de gestion des connaissances. De plus, nous avons montré qu'il est nécessaire d'appréhender l'entreprise à travers deux points de vue : le point de vue « métier » et le point de vue « donnée ».

Il est donc essentiel de pouvoir formaliser la gestion des processus à travers ces deux points de vue. Aussi, nous présentons dans le chapitre suivant, les concepts de base de l'approche processus. Cette approche induit une vision unifiée des données qui est réalisée, à l'heure actuelle, par des solutions technologiques capables d'intégrer les applications logicielles supportant les activités d'un processus. L'étude de ces solutions est développée dans le chapitre suivant.

Enfin, nous avons présenté le projet ArchWare qui fournit le cadre applicatif de ces travaux. Au sein de ce projet, nous nous appuyons sur le modèle OLYMPIOS pour formaliser les styles architecturaux nécessaires à la définition des architectures industrielles et logicielles capables de supporter une approche processus au sein des deux points de vue d'une réalité opérationnelle.

# Chapitre 2 :
# Etat de l'art autour de notre approche de modélisation

# Chapitre 2 :
# Etat de l'art autour de notre approche de modélisation

## 1. Introduction

Comme nous l'avons précisé dans le chapitre précédent, nous considérons que l'entreprise doit être appréhendée à travers un point de vue « métier » et un point de vue « donnée ». Aussi, ce chapitre est consacré à la présentation des différents états de l'art liés aux concepts et aux éléments associés à ces deux perceptions. Dans un premier temps, nous rappelons les différentes structures supportant les organisations industrielles. Nous décrivons ensuite les principaux éléments liés à une approche processus ; cette approche étant à la base du point de vue « métier ». Dans un second temps, nous nous focalisons sur le point de vue « donnée » en présentant les concepts liés à l'unification des données et à l'intégration des applications d'une entreprise. Nous présentons ensuite l'approche BPM – *Business Process Management* – qui constitue, à notre avis, une première étape pour l'intégration de ces points de vue. Nous illustrons cette approche en présentant différents langages utilisés pour formaliser les processus. Dans le cadre de nos travaux, nous nous appuyons sur des représentations centrées architecture (Cf. Chapitre 1, § 3.3). Aussi, nous concluons ce chapitre en montrant que la prise en compte de l'approche BPM nous conduit à nous appuyer sur de nouvelles démarches d'ingénierie des systèmes d'information : la démarche d'ingénierie centrée architecture.

## 2. L'évolution du point de vue « métier »

Cette partie présente l'approche processus mise en place au sein des entreprises pour leur permettre de faire face aux évolutions de l'environnement. Au sein des entreprises, de nouvelles organisations industrielles sont mises en place et leurs architectures facilitent la mise en œuvre de l'approche processus. Aussi, avant de décrire les principaux concepts de cette approche, nous rappelons la définition d'une organisation industrielle. Nous montrons ensuite que l'entreprise est passée d'une organisation fonctionnelle à une organisation transversale s'appuyant sur une structuration matricielle.

### 2.1. Définition et objectif d'une organisation industrielle

Rappelons que l'entreprise est un système compliqué et complexe. En effet, une entreprise est une structure compliquée du fait des nombreux flux de différentes natures qui doivent être synchronisés en fonction d'une stratégie donnée (Cf. Chapitre 1, § 2.2.1). La transformation de ces flux est assurée par les différentes ressources de l'entreprise (hommes, machines, applications). La complexité de l'entreprise est liée à la présence de facteurs aléatoires (pannes, ruptures de stock, absentéisme, données non fiables ou manquantes, retard de livraison, annulation de commandes, etc.) et à l'émergence de nouvelles propriétés liées aux types d'organisation.

Une organisation peut être perçue comme une unité de coordination dotée de frontières repérables, fonctionnant de manière relativement continue en vue d'atteindre un objectif partagé par ses membres (Cf. Chapitre 1, § 2.2.3). E. Morin [Mor 77] considère pour sa part que « *l'organisation est l'agencement de relations entre composants ou individus qui produit une unité complexe ou système, dotée de qualités inconnues au niveau des composants ou*

*individus* ». A travers cette définition, nous pouvons remarquer que le concept d'organisation ne peut être dissocié de la notion d'architecture. Aussi, au sein de nos travaux, nous considérons que l'organisation d'une entreprise nécessite la définition et la création de :

- composants dotés de fonctions spécifiques,
- connecteurs capables d'assurer les liaisons entre les composants.

Nous retrouvons dans cette définition la notion d'architecture introduite dans le chapitre précédent (Cf. Chapitre 1, § 3.3). Les paragraphes suivants présentent l'évolution des différentes configurations mises en place au sein des entreprises pour faire face à l'évolution du contexte industriel.

## 2.2. Les grands modèles d'organisation et leurs conséquences

Selon leurs stratégies, leurs marchés, leurs objectifs et leurs modes de management, les entreprises sont susceptibles d'adopter divers types d'organisation [Gia 91] ; chaque type d'organisation correspondant à une architecture industrielle particulière.

### 2.2.1. Les modèles classiques

Les modèles classiques d'organisation, toujours très présents au sein des entreprises, sont fondés sur le principe de hiérarchie (Cf. Chapitre 1, § 2.2.3.1.b) qui valorise le rôle des cadres dirigeants selon des spécialités bien définies. Cette structuration clarifie le rôle de chacun mais elle ne peut cependant pas éviter des risques de cloisonnement qui sont préjudiciables à l'efficacité.

Ces modèles sont plus particulièrement adaptés aux contextes suivants :

- marché stabilisé (par exemple, le marché automobile),
- marché de demande (par exemple, le marché lié à la grande distribution),
- secteurs à haute intensité capitalistique (par exemple, le domaine pharmaceutique).

Les différents modèles présentés dans le cadre de nos travaux sont :

- la structure fonctionnelle,
- la structure territoriale,
- la structure divisionnelle par produit,
- la structure par marché.

Les principales caractéristiques de ces structures sont fournies dans les paragraphes suivants.

#### *2.2.1.1 La structure fonctionnelle*

Cette structure organise l'entreprise de manière très hiérarchique suivant les grandes fonctions (ou services) présentes en son sein. Chacune de ces fonctions est sous la direction d'un responsable qui reçoit directement ses ordres de la Direction de l'entreprise. Ces différents responsables ont pour mission de diriger un ensemble de personnes pour que la fonction réponde aux objectifs fixés.

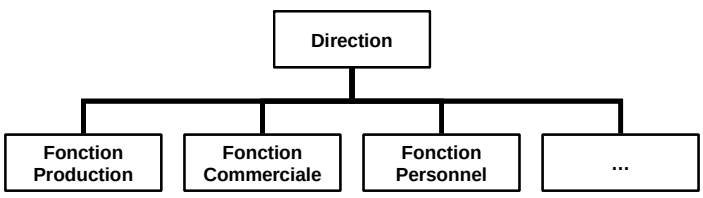



*Figure II.1. : La structure fonctionnelle*

La mise en place d'une telle structure clarifie les responsabilités associées à chaque fonction par la définition des zones de pouvoir, spécialise les compétences requises pour chaque poste et centralise le contrôle. Cependant, malgré ces avantages, ce type de structure induit des inconvénients. En effet, cette organisation « sur-spécialise » les compétences de certains cadres, affaiblit le rôle des « généralistes » ainsi que la coordination entre les différentes fonctions de l'entreprise. Ce manque de coordination peut même aller jusqu'à entraver la croissance économique de l'entreprise.

#### *2.2.1.2 La structure territoriale*

Ce type d'organisation est généralement déployé lorsque l'entreprise est désireuse de prendre en compte les spécificités des différentes régions au sein desquelles elle fournit ses produits ou services. La Direction Générale de l'entreprise délègue ainsi une grande partie des pouvoirs de décision aux responsables de région. L'entreprise est donc structurée sous forme d'unités quasi autonomes (micro-entreprises) contrôlées par un responsable. Ces derniers organisent alors leur unité telle une entreprise indépendante.

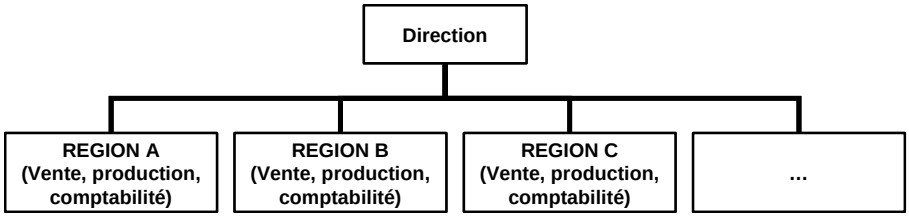



*Figure II.2. : La structure territoriale*

Cette structure permet de déléguer une part importante des responsabilités et privilégie ainsi les marchés et les problèmes locaux. Elle induit ainsi une bonne coordination au sein des différentes régions pour prendre en compte les intérêts locaux. Du fait de l'importance des aspects managériaux, des compétences spécifiques sont requises pour les personnes en charge de ces unités. Ce type de structure rend cependant difficile le contrôle de l'entreprise par la Direction et, généralement, il ne permet pas une bonne répartition des rôles fonctionnels entre les services centraux et régionaux.

#### *2.2.1.3 La structure divisionnelle par produit*

Ce type d'organisation est déployé lorsque la direction est désireuse de focaliser le fonctionnement de l'entreprise sur les produits ou services qu'elle délivre sur le marché. Il s'agit ici d'organiser l'entreprise autour des produits ou services qu'elle vend. La Direction contrôle donc différentes unités (micro-entreprises) spécialisées dans la délivrance de produits ou services spécifiques.

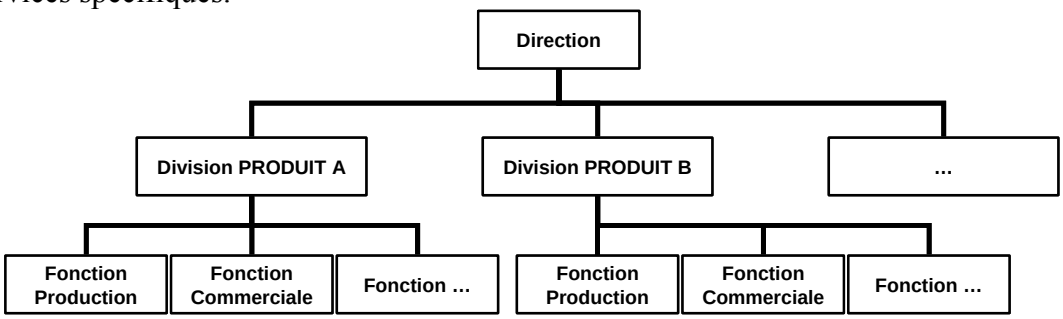



*Figure II.3. : La structure divisionnelle par produit*

La mise en place d'une telle structure permet d'attirer l'attention sur les produits, elle favorise ainsi la croissance et la diversité des produits. Cette structure responsabilise les niveaux

produits et coordonne de manière efficace les activités fonctionnelles de l'entreprise qui leur sont associées. Cependant, une telle structure requiert des personnes formées au management. Les principaux inconvénients liés à la mise en place d'une telle structure sont le risque de faiblesse des services communs (en particulier les services financiers) et la limite des possibilités de contrôle de la part de la direction.

#### *2.2.1.4 La structure par marché*

La logique de cette structure est strictement identique à celle définie dans le paragraphe précédent. Au lieu de focaliser sur les produits, l'entreprise s'appuie sur des marchés spécifiques. La Direction ne dirige plus des unités spécialisées dans la délivrance de produits, mais des unités (micro-entreprises) répondant à des besoins de marchés spécifiques.

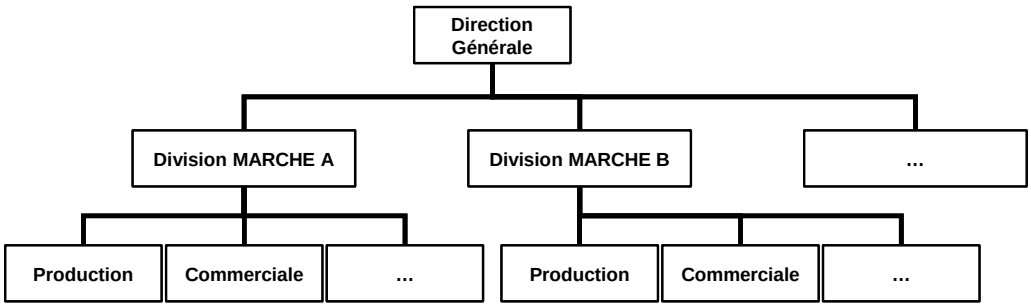



*Figure II.4. : La structure par marché*

Dans certains cas particuliers, par exemple lorsque la production est destinée à un unique client, la structure est alors organisée en fonction des clients (chaque client est alors identifié à un marché et il est géré spécifiquement). Ce type d'organisation est également mis en place dans les approches B2B – *Business To Business*.

### 2.2.2. Les tendances actuelles

Les principaux facteurs influençant les organisations industrielles actuelles peuvent être résumés par les cinq points suivants :

- Les nécessités de la conduite de projet car, pour pouvoir satisfaire ses clients, l'entreprise doit prendre en compte de nombreuses contraintes relatives à ses différentes ressources. La conduite de projet permet de planifier et coordonner la gestion de ses ressources en fonction de leurs disponibilités.
- Les implications des démarches « Qualité » (incluant la maîtrise des processus) en tant que philosophie de l'organisation.
- La recherche du déplacement de la valeur ajoutée le long de la chaîne de production, ce qui induit quelquefois un recours plus ou moins important à la sous-traitance (les constructeurs automobiles, par exemple, externalisent auprès de sous-traitants qualifiés, une grande partie de la fabrication des pièces nécessaires à la construction de leurs modèles, se focalisant ainsi sur une activité d'assemblage.
- La recherche de la différenciation retardée avec ses influences en matière de marque et de système de distribution (par exemple, dans une entreprise de fabrication de meuble, les produits finis sont personnalisés au dernier moment et sont assemblés à partir de composants standards).
- La diminution du nombre de niveaux hiérarchiques débouchant sur de nouvelles définitions du rôle des cadres.

Ces tendances ont conduit les entreprises à définir de nouveaux modèles d'organisation dont les plus courants s'inspirent des structures matricielles. Aussi, avant de décrire en détail les principales caractéristiques liées à une organisation par processus, nous décrivons succinctement les structures suivantes :

- la structure matricielle Produit/Fonction,
- la structure matricielle Projet/Fonction,
- la structure matricielle Marché/Produit,
- la structure matricielle mixte.

#### *2.2.2.1 La structure matricielle Produit/Fonction (orientée produit)*

Au sein de cette structure, la Direction a pouvoir d'autorité sur les différents responsables des principales fonctions de l'entreprise (désignés responsables « fonction ») mais également sur les responsables chargés de la réalisation des produits (désignés responsables « produit »). Chaque responsable « produit » s'appuie sur les différentes fonctions de l'entreprise pour constituer une équipe capable de mettre en œuvre les différentes phases liées à la réalisation et à la fourniture des produits. Pour ce faire, les responsables « produit » identifient, en coordination avec les responsables « fonction », les différents acteurs qui sont affectés à la réalisation de ces phases.

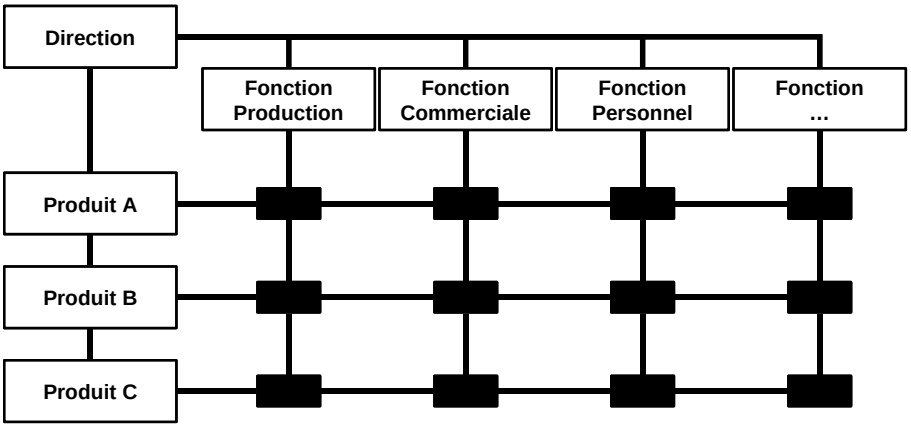



*Figure II.5. : L'organisation matricielle Produit/Fonction*

#### *2.2.2.2 La structure matricielle Projet/Fonction (orientée projet)*

Dans cette structure, la Direction a pouvoir d'autorité sur les responsables « fonction » de l'entreprise mais également sur les chefs de projet qui ont pour mission de mener à bien un projet clairement identifié. Pour ce faire, les chefs de projet s'appuient sur une équipe projet constituée de personnes issues des différentes fonctions de l'entreprise. Ce type d'organisation se rencontre généralement dans les entreprises ayant une fabrication de type unitaire (par exemple, le secteur du Bâtiment et Travaux Publics ou les Sociétés de Services en Informatique Industrielle).

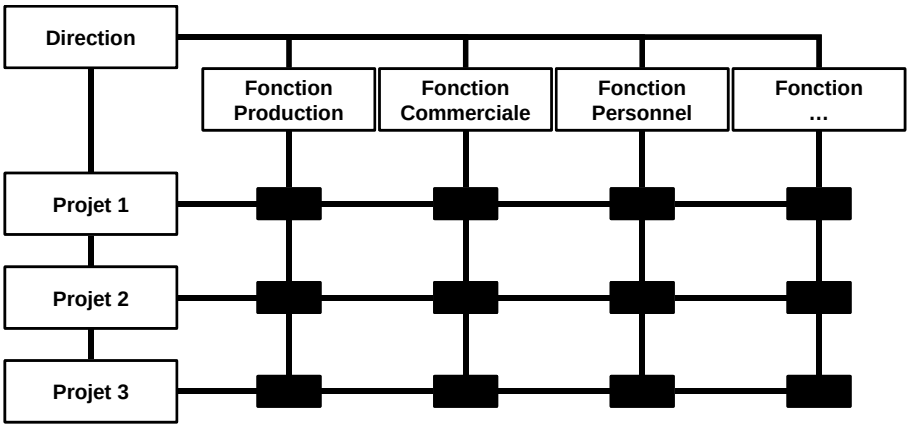



*Figure II.6. : L'organisation matricielle Projet/Fonction*

#### *2.2.2.3 La structure matricielle Marché/Produit*

Au sein de cette structure, les principales fonctions de l'entreprise n'apparaissent pas explicitement. En effet, les personnes affectées à ces fonctions de l'entreprise sont regroupées en équipe. Les responsables « produit » et les responsables « marché » s'appuient ensuite sur ces équipes pour définir une organisation apte à réaliser un objectif. Cette structure est très répandue dans le secteur de l'agroalimentaire.

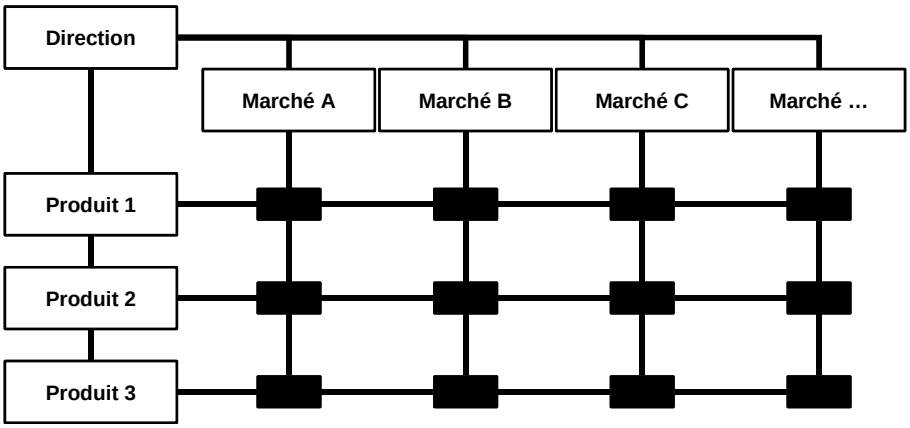



*Figure II.7. : L'organisation matricielle Marché/Produit*

#### *2.2.2.4 La structure matricielle mixte (orientée produit et projet)*

Cette structure combine les organisations matricielles « produit/fonction » et « projet/fonction ». Les responsables « produit » et les chefs de projet constituent des équipes en s'appuyant sur des personnes affectées aux différentes fonctions de l'entreprise. Il est à noter que cette structure est généralement déployée au sein de grandes entreprises telles que l'EDF.

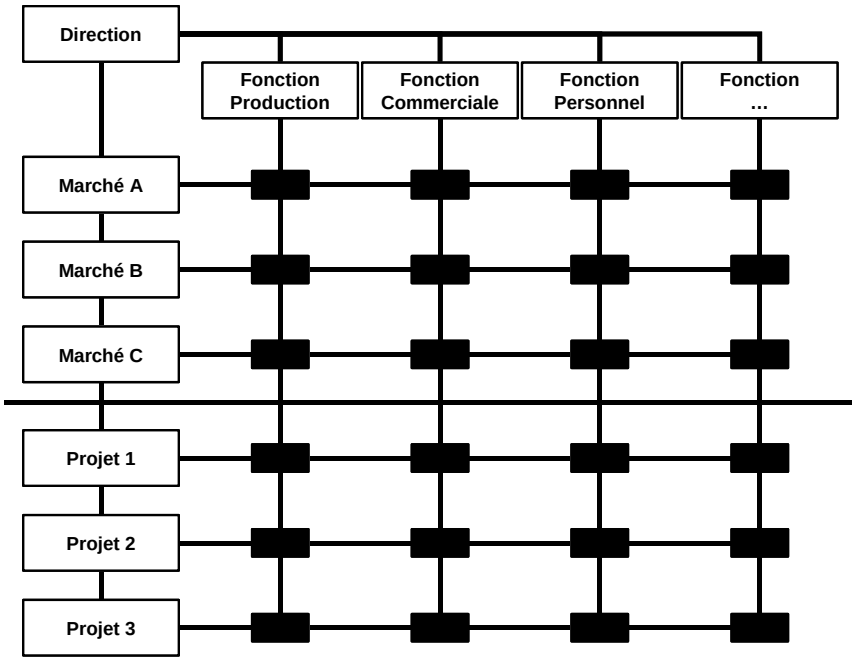



*Figure II.8. : L'organisation matricielle mixte*

## 2.3. L'approche par processus

Ces nouvelles organisations facilitent l'approche processus car elles sont plus adaptées à une vision « transversale » et dynamique des activités de l'entreprise. Avant de présenter l'approche processus, il nous paraît important de rappeler le concept de « processus » car il existe de nombreuses définitions dans la littérature [Per 87] [Lor 95] [Sch 96] [Din 98] [Ver 99] [ISO 00].

M. Périgord [Per 87] définit un processus comme étant « *une succession de tâches réalisées à l'aide de moyens tels que le personnel, les équipements, le matériel, les informations, les procédures. Le résultat final escompté est un produit* ». Nous considérons que cette définition ne doit pas être limitée à l'élaboration d'un produit tangible mais elle doit également concerner la réalisation de services. La norme ISO 9001 version 2000 [ISO 00] définit, quant à elle, le processus comme « *un ensemble d'activités corrélées ou interactives qui transforment des éléments d'entrée en éléments de sortie* ». Cette définition intègre la réalisation des produits ou des services mais elle n'explicite pas l'objectif associé au processus. Suite à ces remarques, nous considérons qu'un processus est défini comme « *un enchaînement partiellement ordonné d'exécution d'activités qui, à l'aide de moyens techniques et humains, transforme des éléments d'entrée en éléments de sortie en vue de réaliser un objectif dans le cadre d'une stratégie donnée* » [Thé 02].

Un processus est donc caractérisé par :

- un **objectif** qui représente l'expression de la finalité du processus,
- une **performance** qui représente le résultat de la comparaison entre l'objectif et sa réalisation,
- des **activités** qui sont les éléments d'action atomique du processus ; une activité exprime la transformation d'une ressource en s'appuyant sur la relation Fournisseur / Utilisateur propre au modèle OLYMPIOS (Cf. Chapitre 1, § 4.3.1.2),
- un **graphe d'activités** qui représente l'enchaînement des activités nécessaires à la réalisation de l'objectif,
- une **fonction de transition** qui contrôle le déroulement du processus.

### 2.3.1. Les avantages d'une approche processus

L'organisation par processus définit une approche transversale au sein de l'entreprise qui permet de :

- **maîtriser les demandes des clients :** moyens (méthode et organisation) pour bien identifier et bien comprendre les exigences des clients dans une politique qualité,
- **mettre en place un ensemble de processus transversaux :** mise en œuvre dans l'entreprise des processus cohérents depuis les exigences du client jusqu'à l'obtention du produit et/ou du service requis,
- **maîtriser les ressources :** détermination et affectation des ressources nécessaires au bon déroulement des processus,
- **assurer le pilotage des processus :** mise en œuvre d'un cycle PDCA (*Plan/Do/Check/Act*) [Dem 82] [ChT 02] en s'appuyant sur des informations mesurables : indicateurs, tableaux de bord.

Appréhender l'entreprise par ses processus permet d'appréhender l'entreprise comme un système unique, et non comme un conglomérat de fonctions, de services, de métiers, de responsabilités, d'objectifs et de stratégies, aussi morcelé que rigide dont il suffit de bien « gérer les interfaces » pour le rendre cohérent.

Cependant, identifier, décrire, formaliser et cartographier les processus n'est pas suffisant pour mettre son entreprise sur le chemin de l'excellence. En effet, cette approche ne doit pas se limiter à une démarche de (re)certification ISO 9001 version 2000 [ISO 00] qui viserait à simplement décrire les processus à travers un référentiel. Appréhender l'entreprise à travers ses processus n'est pas une fin en soi, mais un moyen de construire une organisation capable de s'adapter aux besoins de l'environnement. Cette approche n'a de sens que si elle est prolongée par une étape qui vise à superviser, piloter, améliorer, voire automatiser les processus. Les concepts de pilotage définis dans le chapitre précédent (Cf. Chapitre 1, § 2.2.2 et § 2.2.3.3) fournissent les moyens d'atteindre cette finalité.

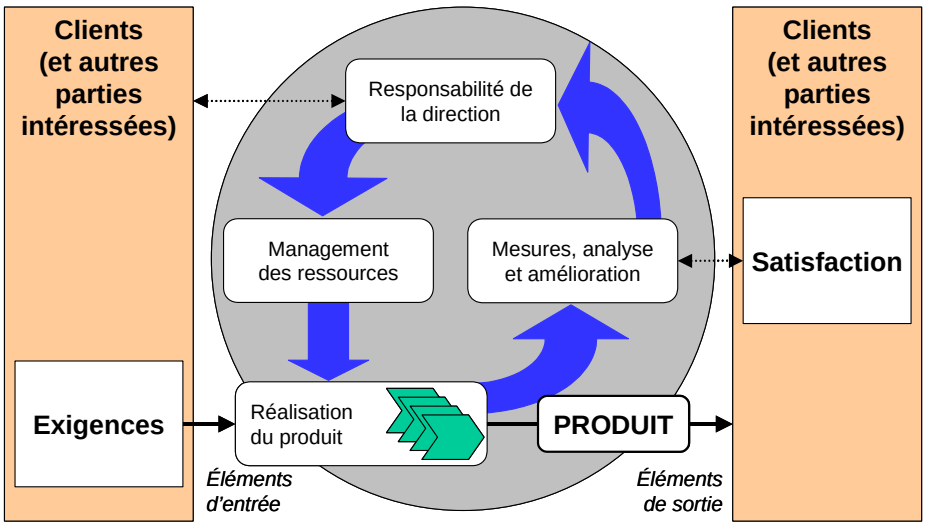



*Figure II.9. : Le management des processus de l'entreprise*

### 2.3.2. Les différents types de processus

Différents travaux ont pour but de proposer une typologie des processus [Mur 02]. Citons par exemple :

- une typologie par enjeux qui identifie :
  - les processus clés qui contribuent directement à la réalisation des objectifs stratégiques,
  - les processus clients qui impactent directement la satisfaction des clients,
  - les processus critiques qui ont un niveau de risque important,
  - les processus normaux qui sont sans enjeu spécifique mais dont la maîtrise doit être conservée au sein de l'entreprise,
  - les processus externalisables qui peuvent être sous-traités,
- une typologie par niveau de maturité qui caractérise les processus en fonction de leur niveau de maîtrise :
  - les processus aléatoires qui ne sont pas maîtrisés,
  - les processus soumis aux bonnes pratiques professionnelles,
  - les processus sous assurance qualité,
  - les processus maîtrisés,
  - les processus en amélioration permanente,
  - les processus « benchmarkables ».

Outre ces différentes typologies, nous avons retenu une typologie qui distingue trois grandes catégories de processus :

- les processus de réalisation, ou opérationnels,
- les processus support, ou de soutien, ou d'appui,
- les processus de management, ou de direction.

Les processus de réalisation sont les processus qui contribuent à la réalisation des produits ou des services. Ce sont les processus qui intègrent les différentes étapes du cycle de vie d'un produit ou d'un service, à savoir la définition, la conception, la production, la vente, la maintenance.

Les processus support concernent la fourniture des facteurs de production nécessaires à la réalisation des produits ou des services (Cf. Chapitre 1, § 2.2.1). Par exemple, pour les ressources humaines, ces processus traitent de l'implication du personnel, de sa formation et des qualifications.

Les processus de management élaborent la politique de l'entreprise et déploient les objectifs liés à celle-ci. Ces processus sont essentiels au fonctionnement de l'entreprise, ils doivent coordonner le déroulement des processus de réalisation et des processus support. Ces processus intègrent par exemple la revue de direction, le pilotage de l'amélioration continue, le management de la qualité, etc.

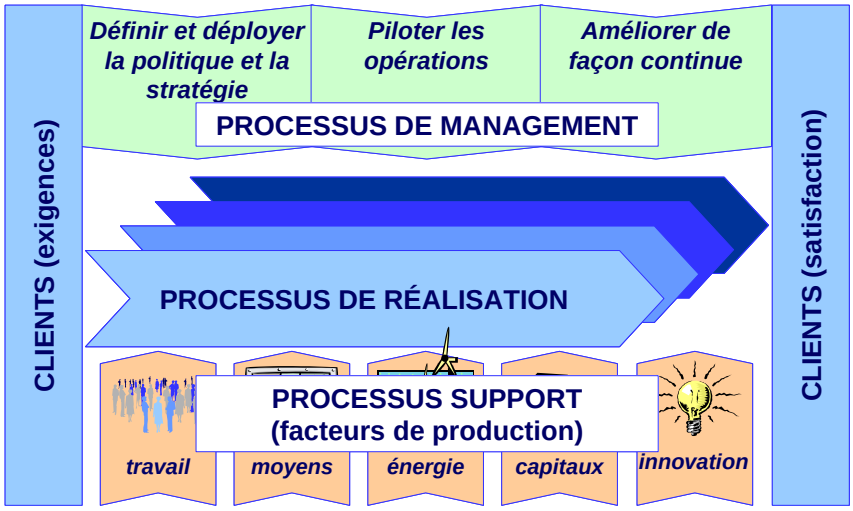



*Figure II.10. : Les trois types de processus dans l'entreprise*

Il est important de noter que la nature des processus est dépendante de l'activité de l'entreprise. Par exemple, dans une organisation dédiée à l'enseignement, la gestion de la formation est assurée par des processus de réalisation tandis que cette gestion est liée à des processus support dans le cadre d'une organisation réalisant des produits tangibles.

Nous avons choisi de retenir cette typologie au sein de nos travaux car elle est parfaitement adaptée au paradigme du système d'information que nous avons présenté dans le chapitre précédent (Cf. Chapitre 1, § 2.2.3.3). Nous rappelons qu'un système d'information doit gérer des processus en s'appuyant d'une part, sur un système de gestion de ressources qui fournit les données nécessaires au déroulement des processus et, d'autre part, sur un système de gestion des connaissances pour tenir compte du savoir-faire de l'entreprise. Aussi, nous référant à cette typologie :

- les processus de management concrétisent le fonctionnement du système de gestion de connaissances (ce système s'appuie sur les connaissances de l'entreprise pour gérer les objectifs de chaque processus en accord avec la stratégie de l'entreprise),
- les processus de réalisation sont les processus gérés par le système de gestion des processus (ce système gère le déroulement et l'évolution des processus de l'entreprise),
- les processus support concrétisent, quant à eux, le système de gestion des ressources (ce système est la mémoire nécessaire au pilotage des processus de l'entreprise).

Dans le cadre de ces travaux, nous nous intéressons plus particulièrement aux processus de réalisation qui définissent le point de vue « métier » et aux processus support dédiés aux traitements des données nécessaires au déroulement des processus de réalisation.

### 2.3.3. La notion de pilotage au sein de cette approche

Comme nous l'avons évoqué au sein du chapitre 1, nous nous appuyons sur la notion de pilotage basée sur la théorie des plans d'actions (Cf. Chapitre 1, § 2.2.2) qui fournit un modèle explicite décrivant l'enchaînement des activités à réaliser (plan d'actions). Ce modèle caractérise le processus de référence, désigné référent (Cf. Chapitre 1, § 4.3.3), qui définit la situation finale attendue (réalisation de l'objectif) et les différentes situations intermédiaires.

Le pilotage d'un processus consiste alors à faire évoluer le processus vers son objectif via différentes situations intermédiaires. Il s'appuie sur les fonctions suivantes [Mei 00] :

- la **planification** qui, à partir d'un besoin et de procédures standard, construit un plan d'actions que nous désignons référent,
- la **mise en place** de l'organisation par la mobilisation et l'allocation des moyens humains et matériels nécessaires à la réalisation du plan,
- l'**exécution** du plan,
- le **suivi** qui contrôle le bon déroulement du plan,
- la **clôture** qui vérifie que le résultat obtenu est conforme à l'objectif fixé.

Par conséquent, cette approche du pilotage peut se décomposer en trois grandes étapes :

- L'étape de **mise en œuvre** qui construit le référent en définissant les actions à réaliser. L'établissement de ce référent nécessite la connaissance d'une part, des actions et de leurs effets et, d'autre part, des ressources capables de supporter ces actions.
- Les étapes de **réalisation / suivi**. Il s'agit de mettre en œuvre le référent précédent en vérifiant qu'il n'existe pas de dérive. Il est donc nécessaire de disposer de points de mesure pour permettre le déclenchement d'actions correctives. Ces étapes intègrent les fonctions suivantes [Tre 96] :
  - la fonction de **lancement** déclenche la réalisation des actions en fonction du référent et de l'état courant,
  - la fonction **commande** exécute les instructions nécessaires à la réalisation des actions,
  - la fonction **conduite** assure d'une part, le suivi du déroulement des actions et, d'autre part, définit les actions à mettre en œuvre pour traiter certains dysfonctionnements.
- L'étape d'**adaptation** qui cherche à mettre en œuvre une solution lorsqu'un aléa est détecté.

En conclusion, cette approche doit, à terme, amener la Direction à gérer son entreprise par les processus pour la rendre plus performante et plus apte à s'adapter et à anticiper les changements de l'environnement. Cette approche est donc nécessaire pour rendre l'entreprise agile. L'approche par processus définit ainsi une nouvelle organisation pour déployer et piloter la stratégie à tous les niveaux de l'entreprise, et à destination de tous les acteurs [PPR 03].

# 3. L'évolution du point de vue « donnée »

L'évolution des organisations industrielles a été possible du fait du formidable essor des techniques liées aux traitements de l'information. Cette partie présente l'évolution des concepts relatifs au point de vue « donnée ». Rappelons que ce point de vue vise à fournir une vision unifiée des données (Cf. Chapitre 1, § 3). Ce paragraphe définit dans un premier temps l'unification des données et montre que cette dernière nécessite une intégration des fonctions manipulant ces données. Comme dans le cadre de nos travaux, nous nous sommes plus particulièrement intéressés aux applications informatiques supportant ces fonctions au sein d'une entreprise, nous terminerons cette partie par un état de l'art des concepts relatifs à l'EAI – *Enterprise Application Integration*. Ces solutions visent à faire interagir divers progiciels hétérogènes de l'entreprise afin de proposer une vision unifiée de l'information.

## 3.1. Définition et objectif d'une vision unifiée des données

Il n'est plus à démontrer que l'entreprise d'aujourd'hui doit appréhender de plus en plus de données pour continuer à survivre. Ces données, essentielles pour l'entreprise, proviennent de différentes sources hétérogènes (bases ou entrepôts de données, applications, système d'exploitation). Aussi, les entreprises doivent unifier ces données, tant au niveau syntaxique (format) que sémantique pour faciliter leur gestion.

L'unification au niveau syntaxique garantit la faisabilité de l'échange de données entre les différentes fonctions de l'entreprise. En effet, il est nécessaire au sein de chaque fonction de connaître les données manipulées. L'unification sémantique permet la bonne utilisation de ces données au sein de chaque fonction. En d'autres termes, l'unification des données nécessite :

- un dictionnaire recensant l'ensemble des données manipulées par les différentes fonctions,
- une description du format de ces données,
- une description de l'utilisation de ces données.

L'unification des données a donc pour objectif de construire et de fournir l'ensemble des informations nécessaires aux différents acteurs de l'entreprise. Elle permet donc de rendre cohérent le fonctionnement d'une entreprise perçue à travers les points de vue de ses acteurs en intégrant les différentes fonctions qui leur sont associées.

Nous avons vu dans le chapitre précédent (Cf. Chapitre 1, § 2.1) que les entreprises sont de plus en plus souvent amenées à s'appuyer sur différentes applications logicielles. Les évolutions des technologies de traitement de l'information ont conduit l'entreprise à intégrer différentes applications hétérogènes qui n'ont généralement pas été conçues à cet effet. L'objectif de cette intégration vise d'une part, à réaliser un système d'information s'appuyant sur des applications commercialisées sur le marché par des fournisseurs différents et, d'autre part, à faire interopérer des applications déjà en service en vue d'obtenir de nouvelles propriétés dues à leur coopération.

Cette intégration induit un problème d'hétérogénéité qui ne peut être maîtrisé que si les applications sont capables de coopérer et si le système d'information est capable de masquer cette hétérogénéité à son environnement.

Pour que les différentes applications interopèrent correctement malgré leur hétérogénéité, il est nécessaire qu'elles présentent une bonne complémentarité fonctionnelle, qu'elles se synchronisent correctement et qu'elles interprètent de la même manière les contenants et les contenus des structures de données échangées. Ceci suppose également la résolution des problèmes techniques à tous les niveaux d'interfaces, tant des applications que des infrastructures technologiques qui les supportent (plates-formes, systèmes d'exploitation, systèmes de gestion de bases de données, réseaux). En d'autres termes, les adaptations nécessaires à l'interopérabilité doivent être réalisées à différents niveaux. Elles doivent rendre les applications complémentaires et cohérentes dans leurs fonctions et leur dynamique. En ce sens, la signification des données dans les différentes bases de données et leur utilisation dans les applications doivent être cohérentes. Les échanges de données doivent s'appuyer sur des syntaxes adaptées et les infrastructures technologiques doivent être intégrées. Par exemple, la structure d'une commande n'est pas nécessairement identique dans une application de gestion commerciale et dans une solution de type ERP. Dans ce cas, l'intégration doit transformer la commande pour qu'elle puisse être interprétée par chaque application.

L'intégration est, par ailleurs, un vrai défi lancé aux spécialistes informatiques car elle induit des exigences parfois antinomiques. Par exemple :

- augmenter l'efficacité des liens entre applications tout en diminuant leur interdépendance pour faciliter les évolutions,
- rendre accessible le système d'information pour autoriser le partage d'informations avec des partenaires tout en préservant son savoir et sa sécurité.

Le système d'information global doit masquer son hétérogénéité interne aux différents utilisateurs. L'interface utilisateur doit donc être unique et homogène quelle que soit l'hétérogénéité des applications auxquelles elle donne accès, et doit permettre à l'utilisateur d'intégrer par lui-même, différentes applications au cours de sa tâche.

## 3.2. L'intégration des applications d'entreprise

L'intégration d'applications d'entreprise – *Enterprise Application Integration* [Man 99] [BFW 00] [Hos 00] [Lin 00] [Sch 00] rationalise l'intégration des applications pour rendre le système d'information plus flexible et donc plus réactif [Bor 02]. Cette solution technologique relativement jeune s'appuie sur des concepts cependant connus depuis un certain temps [JoP 01], mais son développement est seulement en plein essor. Aussi, en plus d'un fort développement dans l'industrie, cette notion fait l'objet de nombreuses recherches dans divers domaines [KTK 03] [EYR 03] [Evg 02] [ITL 03] :

- gestion de la chaîne logistique,
- entreprise agile,
- commerce électronique,
- conception des systèmes d'information.

Ce paragraphe présente les principaux éléments liés à l'intégration d'applications d'entreprise en précisant les objectifs attendus par la mise en place d'une solution EAI. Dans un deuxième temps, nous décrivons le modèle de maturité de cette solution et les différentes couches d'une architecture EAI. Une étude plus détaillée décrivant les différentes technologies, l'évolution du modèle EAI et les différents formats d'échanges est fournie en annexe de ce document (Cf. Annexe 2).

### 3.2.1. Les objectifs de l'EAI

Les solutions EAI cherchent à faire interagir divers systèmes hétérogènes de l'entreprise afin de [Sch 00] [Sto 99] :

- constituer un support efficace au déroulement des processus de l'entreprise en résolvant les problèmes de cohérence entre des systèmes qui interagissent,
- fournir une structure capable d'évoluer en fonction des changements du marché (économiques, sociologiques, technologiques),
- présenter à l'utilisateur final une vision unifiée de l'information gérée par les différentes applications afin de l'aider dans ses prises de décision.

Les solutions EAI sont actuellement considérées comme l'**épine dorsale** de l'entreprise [Hos 00] car elles forment un ensemble d'outils, de progiciels d'intégration qui offrent un moyen de connecter et de faire communiquer entre elles les applications de l'entreprise (existantes ou à venir).

### 3.2.2. Le modèle de maturité pour l'intégration d'applications

Un organisme américain (*American Management Systems*) a proposé un modèle [Sch 00] pour valider le degré d'intégration des applications des entreprises. Ce modèle a pour but de guider les entreprises dans les étapes conduisant à la gestion optimale de leurs capacités d'intégration d'applications en vue de satisfaire les objectifs de l'entreprise (économiques, sociologiques, technologiques). Ces étapes sont caractérisées par des différents niveaux dans le modèle de « maturité », elles structurent de façon progressive l'entreprise en définissant les différentes technologies capables de décrire, de supporter et de piloter les différents processus de l'entreprise.

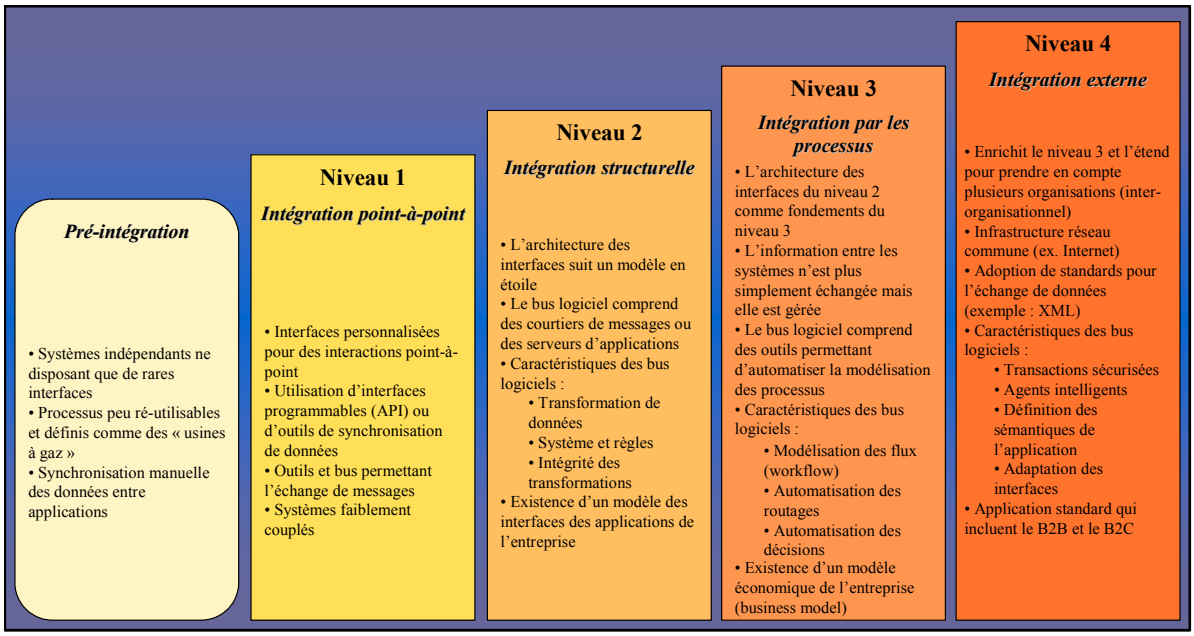



*Figure II.11. : Le modèle de maturité*

Il est à noter que le dernier niveau de maturité (le niveau 4 du modèle de maturité) prend en compte la chaîne logistique d'une manière globale, ce qui implique l'adoption de standards et des protocoles de communication pour structurer les échanges.

### 3.2.3. Les différentes couches d'une architecture EAI

Comme nous l'avons vu auparavant, une plate-forme EAI fait interagir divers progiciels hétérogènes de l'entreprise afin de proposer une vision unifiée de l'information, en s'appuyant sur l'approche processus. Pour permettre ceci, une plate-forme EAI s'articule autour des couches suivantes :

- la couche « Connexion et Transport »,
- la couche « Transformation – Interprétation des données »,
- la couche « Routage »,
- la couche « Modélisation ».

#### *3.2.3.1 Couche « Connexion et Transport »*

Afin de faire communiquer diverses applications via une plate-forme EAI, il faut être en mesure d'accéder physiquement à ces applications, c'est-à-dire interfacer ces applications. Ainsi, la première couche d'une plate-forme EAI fournit un accès physique aux données des applications et assure le transport de ces données. Cette couche « connexion et transport » comprend d'une part, les différents **connecteurs** (progiciels ou standards) qui permettent

d'interfacer les applications pour accéder, modifier ou créer des données et, d'autre part, les **protocoles de transport** qui permettent d'acheminer et de distribuer les données.

#### *3.2.3.2 Couche « Transformation – Interprétation des données »*

La couche « Transformation – Interprétation des données » sert à structurer le format des données véhiculées au sein d'une plate-forme EAI. Le but de cette couche est donc de transformer le format des messages (données des diverses applications) dans un format pivot propre à la plate-forme EAI. Ce format pivot est à l'heure actuelle généralement basé sur le format XML – *eXtensible Markup Language*.

#### *3.2.3.3 Couche « Routage »*

La couche « Routage », sert à aiguiller les différents flux d'information vers les applications. Cet aiguillage s'appuie sur une représentation des processus, il est à l'heure actuelle basé sur un moteur de workflow. D'un point de vue technique, cet aiguillage peut notamment être rendu possible grâce à la mise en place d'outils de gestion de files d'attentes de messages.

#### *3.2.3.4 Couche « Modélisation »*

La couche « Modélisation » est très importante car elle permet de représenter les processus des entreprises. Ces représentations décrivent les tâches, les organisations et les personnes impliquées (en particulier leur rôle), les informations nécessaires ou fournies et les outils requis. Cette modélisation permet à l'entreprise de maîtriser les différents flux d'information (échange de données entre les applications de l'entreprise) circulant en son sein via les processus métier. C'est à ce niveau que la plupart des offres EAI ont un manque car, la plupart du temps, elles ne proposent que des outils capables de décrire un enchaînement d'activités.

### 3.2.4. L'architecture d'une plate-forme EAI

Pour mettre en œuvre les fonctionnalités définies dans les différentes couches présentées précédemment, la plate-forme EAI s'appuie sur une architecture construite autour d'un courtier de messages – *Message broker* – et d'un bus orienté messages – *Message Oriented Middleware (MOM)*. Le courtier de messages transforme les données et les diffuse vers les différentes applications (via des connecteurs) grâce au MOM. Ce sont ces deux éléments qui différencient les offres en termes de performance.

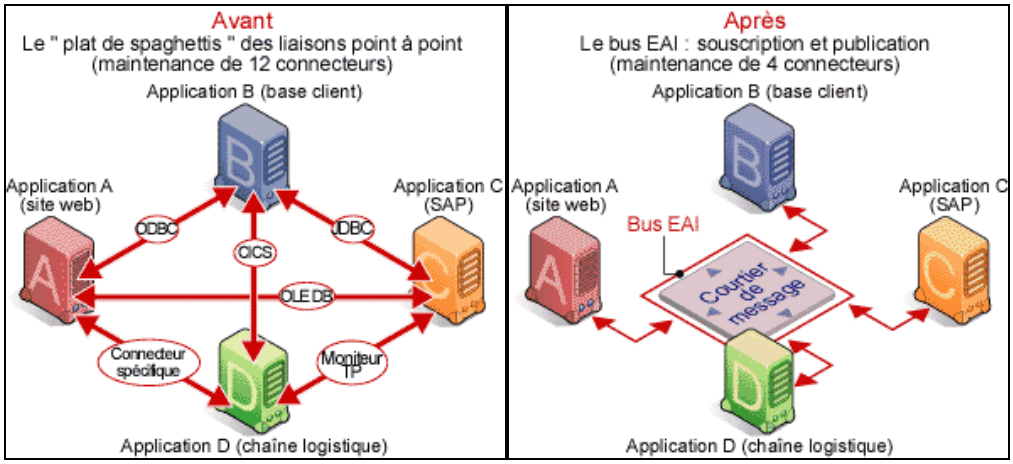



*Figure II.12. : Du « plat de spaghettis » au bus EAI*

Les outils EAI concentrent les connexions autour d'un bus asynchrone pour diminuer le couplage entre les applications plutôt que de développer des connexions point à point (plat de spaghettis). Par conséquent, cette approche réduit de manière importante le nombre de connecteurs et fournit une vision homogène des informations.

La plupart des solutions disponibles sur le marché s'appuient sur des bus propriétaires. Il existe toutefois de nouvelles approches basées sur des standards parmi lesquels nous pouvons citer les solutions s'appuyant sur les normes Java telles que JMS – *Java Messaging Service* – et JCA – *J2EE Connector Architecture*. A travers ces approches, les traitements sont réalisés par des appels EJB – *Enterprise JavaBean*, des composants COM ou des Services Web.

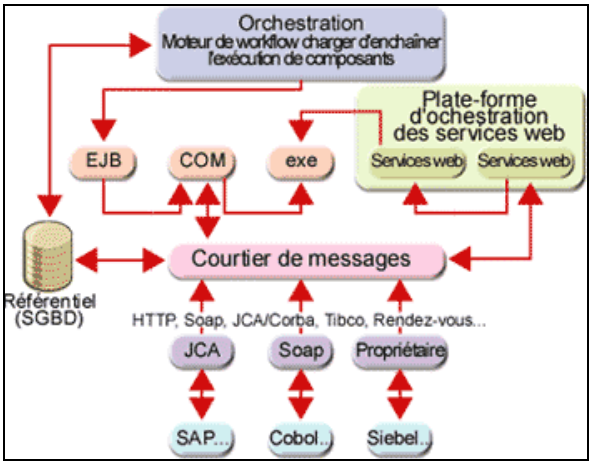



*Figure II.13. : Architecture d'une solution EAI*

Dans cette architecture, toutes les opérations de routage et de transformation sont concentrées dans le courtier de messages, ce qui facilite son administration, mais induit un goulet d'étranglement potentiel. Aussi, ce composant délègue de plus en plus les transformations aux différents connecteurs et s'appuie sur un bus de type « *publish & subscribe* » (Cf. Annexe 2, § 1.1.2, § 1.1.3 et § 1.1.4) pour diffuser les informations par types d'abonnement (Cf. Annexe 2, § 1.2.5). Cette nouvelle approche diminue la complexité du routage. En effet, en s'abonnant à une catégorie de messages, les applications reçoivent automatiquement toutes les informations qui les concernent. Le fonctionnement du courtier de messages se limite à une diffusion des messages sur des canaux distincts plutôt qu'à travers des liaisons « point à point ».

### 3.2.5. De l'intégration à l'orchestration

« *Il ne suffit pas de posséder des outils d'intégration. Encore faut-il savoir quels processus intégrer et dans quel ordre le faire* » remarque François Tabourot, Directeur Général de Mega International [Bor 02]. Ce constat montre la nécessité de disposer d'une nouvelle couche capable d'orchestrer le déroulement des processus de l'entreprise. D'ailleurs, les principaux éditeurs de solutions EAI (Tibco, Vitria, Seebeyond, Mercator ou webMethods) ont ajouté cette couche, désignée BPM – *Business Process Management* – à leur offre. Cette couche permet de modéliser les processus en fonction d'une logique métier. Dans ce cas, le courtier de messages joue le rôle de chef d'orchestre en déclenchant un enchaînement d'activités suite à la réception d'un événement ou d'un message. A titre d'exemple, ces activités peuvent être la publication d'informations, la connexion synchrone à une application, l'exécution d'une règle métier encapsulée dans un composant COM, ou bien encore l'attente d'un message. Par conséquent, les plates-formes EAI évoluent de la « simple » intégration de données vers la gestion des processus.

# 4. Vers une intégration des deux points de vue

Ayant présenté les concepts liés aux points de vue « métier » et « donnée », nous allons maintenant nous intéresser à une solution technique qui vise à intégrer ces deux points de vue. En effet, il existe, à l'heure actuelle, des solutions BPM – *Business Process Management* – dont l'objectif est de contrôler le déroulement des processus « métier » d'une entreprise en s'appuyant sur des plates-formes EAI.

En effet, les processus définissent les squelettes des mécanismes techniques qui sont ensuite paramétrés au sein d'une plate-forme EAI pour coordonner l'exécution des différentes applications en vue de réaliser un objectif fixé. Le référentiel métier des solutions BPM permet de formaliser les processus d'une entreprise à travers un vocabulaire spécifique qui est ensuite utilisé par les plates-formes EAI pour traduire les échanges entre les applications dans un langage pivot. Par exemple, le concept « client » d'une entreprise qui circule entre les différents processus doit être traduit pour être exploité par les différentes applications qui le manipulent.

A l'heure actuelle, cette couche BPM est devenue indispensable pour orchestrer des transactions complexes, pouvant faire appel à des interactions humaines, exécutées au sein d'une entreprise ou entre plusieurs entreprises. Cette couche peut être représentée par le schéma suivant :

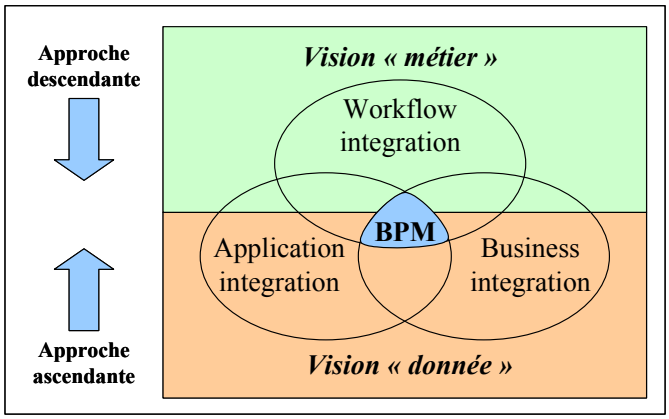



*Figure II.14. : Localisation du BPM*

Cette approche recouvre les idées suivantes [BGL 02] :

- une gestion globale des processus qui couvrent à la fois les échanges entre les applications, entre les entreprises et entre les personnes,
- l'accomplissement fiable de transactions simples et uniques comme celui de séquences complexes et longues,
- une structure permettant à l'entreprise d'être plus agile et capable de déployer plus rapidement de nouveaux modèles organisationnels, de nouvelles collaborations, de nouvelles manières de travailler et de nouvelles stratégies,
- le déploiement du modèle de processus conceptualisé pour aboutir à l'intégration et l'automatisation des activités.

L'objectif des solutions BPM peut dont être défini de la manière suivante : « *La capacité à identifier, conceptualiser, déployer, exécuter, interagir, opérer, optimiser et analyser les*

*processus de bout en bout, et de le faire au niveau de la stratégie d'entreprise, et non pas au niveau de la mise en place technique* »[11].

En étant explicite à un niveau métier et en s'appuyant sur l'intégration d'applications, l'approche BPM gère le déroulement des processus de l'entreprise.

## 4.1. Les concepts de BPI, BPA et BPRM

L'approche BPM s'articule généralement autour des trois éléments suivants :

- BPI – *Business Process Integration*,
- BPA – *Business Process Automation*,
- BPRM – *Business Process Repository Management*.

Les éléments BPA et BPI concernent respectivement l'automatisation et la coordination des processus tandis que l'élément BPRM gère les données caractérisant ces processus :

- **BPI :** cet élément intègre les connecteurs existant entre le moteur d'intégration (Cf. § 3.2.3.1) et les applications à coordonner. De plus, il prend en charge, sur de longues périodes, les échecs et les annulations requises par les conditions externes et il personnalise les échanges en fonction de l'activité de l'entreprise et de ses partenaires.
- **BPA :** cet élément assure la conceptualisation, la simulation, le déploiement et le déroulement des processus. Il s'appuie sur un moteur de « workflow » pour automatiser le déroulement des processus et pour intégrer les interactions humaines requises par certaines activités.
- **BPRM :** cet élément gère les descriptions des processus et assure un fonctionnement indépendant des deux éléments précédents (BPI et BPA). En exposant tous les processus sous forme de données explicites, l'entreprise gère son « capital process » d'une manière similaire à celle dont elle gère son « capital données ».

Ces différents éléments nécessitent des langages dédiés capables de formaliser les processus.

## 4.2. Les langages pour la formalisation des processus [Den 03]

La mise en œuvre d'une approche BPM nécessite de disposer de langages de formalisation de processus. A l'heure actuelle, il existe deux catégories de langage :

- les langages (généralement graphiques) utilisés pour la conception des processus,
- les langages utilisés pour dérouler les processus.

Parmi les langages de conception, nous présentons la proposition de l'OMG qui définit différents profils adaptés à des domaines spécifiques et les propositions issues de la BPMI[12] – *Business Process Management Initiative* – qui s'articule autour d'un langage de conception, la notation BPMN – *Business Process Modeling Notation* – [BPM 02]. Nous nous appuyons sur différentes études pour présenter les langages de déroulement de processus.

### 4.2.1. Les langages utilisés pour la conception des processus

Le profil EDOC proposé par l'OMG fournit un moyen pour modéliser les collaborations entre les processus en se basant sur la notation UML et en se focalisant sur les systèmes exécutables utilisant des « solutions technologiques » telles que les Services Web, ebXML, Corba, J2EE.

[11] BPM Conference 2003 – Business Process Management Group
[12] http://www.BPMI.org

La notation BPMN proposée par la BPMI s'appuie, quant à elle, sur les diagrammes de comportement UML pour définir une représentation graphique des processus.

#### *4.2.1.1 Le profil UML pour EDOC*

Il n'existe pas de définition normative d'un profil UML cependant, nous pouvons considérer qu'un profil UML est un package stéréotypé contenant des modèles adaptés à un domaine spécifique en utilisant des mécanismes d'extension [Bla 02]. Les profils UML fournissent ainsi un mécanisme capable de spécialiser le langage UML à un contexte particulier [Sof 99]. Dans le cadre de ces travaux, nous présentons le profil UML pour EDOC – *Entreprise Distributed Object Computing Systems* – [OMG 02a].

Ce profil est construit pour raffiner et étendre des concepts existants pour fournir à l'entreprise :

- un meilleur support pour la coordination et les compositions sémantiques des processus et de leurs composants, tout en incluant la réutilisation de COTS[13] mis en œuvre lors de l'exécution des processus métiers,
- un support explicite pour gérer les événements et les exceptions dans un processus,
- un support de spécification du comportement des entités.

Ce profil s'appuie sur des concepts suffisamment formels pour permettre la transformation des modèles en modèles d'exécution qui tiennent compte des différents choix d'implémentation.

Le concept clé de ce profil est l'architecture de collaboration d'entreprise (*Enterprise Collaboration Architecture*). Cette architecture ECA est destinée à formaliser des processus métiers coopératifs en s'appuyant sur cinq profils UML :

- L'architecture de collaboration de composant (*Collaboration Component Architecture*) s'appuie sur les concepts de classe, de collaborations et d'activités du langage UML pour décrire la structure et le comportement des composants d'un système à différents niveaux de granularité.
- Le profil des processus métier (*Business Process profile*) décrit un ensemble d'extensions du langage UML pour formaliser les processus « métiers ». Ce profil fournit les concepts permettant de composer des activités, de sélectionner des critères sur les entités qui supportent ces activités, de coordonner les processus (déclenchement sur événement, communication, etc.).
- Le profil entités (*Entities profile*) décrit un ensemble d'extensions du langage UML qui peuvent être utilisées pour modéliser des concepts propres au domaine d'application, et ce, afin de les définir en tant que composants composables.
- Le profil d'événement (*Event profile*) caractérise, quant à lui, les concepts nécessaires à la conception d'un système orienté événements.
- Le profil relationnel (*Relationships profile*) décrit les extensions apportées au langage UML pour formaliser l'enchaînement des actions.

Le profil UML pour EDOC fournit également la possibilité d'exploiter la capacité des patrons pour réutiliser des modèles et structurer l'approche de modélisation. Il s'appuie, pour ce faire, sur le profil BFOP – *Business Function Objects Patterns*.

[13] COTS – Component Of The Shelf : Progiciel du marché

L'OMG fournit également un profil UML pour l'EAI qui est une spécialisation du modèle de flot de composition du profil UML pour EDOC. Il existe donc une correspondance entre le profil CCA et les éléments du profil EAI et le profil UML pour l'EAI [OMG 02b].

#### *4.2.1.2 La notation BPMN*

La notation BPMN vise à fournir un langage symbolique utilisable par différents acteurs (analyste, développeur, etc.) pour formaliser les processus d'une entreprise.

Au sein de cette notation, le concept d'activité regroupe différents niveaux de perception :

- le processus est considéré comme un réseau de flots d'activités et de contrôles,
- le sous-processus représente une activité composée (définie comme un flot d'activités),
- la tâche est une activité atomique.

Outre ce concept d'activité, la notation BPMN définit d'autres concepts pour formaliser les processus :

- l'événement qui traduit le début, le changement d'état ou la fin d'une activité (*start*, *intermediate* et *end*),
- la décision qui définit des alternatives au sein d'une séquence,
- les connexions entre objets qui peuvent être de deux types :
  - connexion de type flot :

▪un flot de séquence pour préciser l'ordre d'exécution des activités,
▪un flot de messages pour signifier des échanges entre deux activités,

  - une connexion de type association qui permet d'associer des informations à des flots,
- les groupements (*pool* et *lane*) qui permettent de partitionner l'ensemble des activités.

La BPMI propose une correspondance entre la notation BPMN et le langage d'exécution BPML.

### 4.2.2. Les langages de déroulement des processus

L'émergence du BPM a introduit plusieurs langages de déroulement de processus. Dans le cadre de notre étude, nous nous sommes intéressés aux langages capables de supporter la technologie EAI en nous appuyant sur des études[14,15] résumées dans le tableau suivant (Cf. Figure II.15).

Selon cette étude, seuls BPML, BPEL4WS et EDOC supportent la technologie EAI. Cependant, dans le cadre de nos travaux nous considérons EDOC comme un langage utilisé pour la conception des processus.

Le langage BPML – *Business Process Management Language* – proposé par la BPMI, est un méta langage basé sur le langage déclaratif XML (il est donc indépendant d'une plate-forme spécifique) capable d'exprimer des processus exécutables de l'entreprise [Ark 02] en s'appuyant sur les concepts définis dans la notation BPMN.

BPEL4WS – *Business Process Execution Language for Web Services* – est né de la fusion du langage XLANG – *XML Language* (Microsoft) et du langage WSDL – *Web Services Description Language* (IBM). Il combine les avantages d'un langage structuré en bloc

[14] http://www.ebpml.org
[15] http://www.cis.unisa.edu.au/~cissh/research/webflow/bpdl.html

(XLANG) avec le support de graphe orienté processus (WSDL) [BPE 03]. Ce langage spécifie d'une part, comment les processus utilisent les Web Services et, d'autre part, les Web Services fournis par un processus.

| Specification | Control flow | Data flow | Message flow | Transaction | EAI friendly | B2B friendly | User friendly |
|---|---|---|---|---|---|---|---|
| XPDL | Transition | Process variables | Nested & chained process | No | No | No | No |
| XLANG | Block structured | XML | Web services& contracts | Yes | NO | No | No |
| WSFL | Transition | XML | Web services &global model | Yes | No | No | No |
| BPML 1.0 | Block structured | XML | Web services &global model | Yes | Yes | No | No |
| BPEL4WS | Block structured | XML | Web services | Yes | Yes | Kind of | No |
| EDOC | Event/ Notification | Entities | Events | No | Yes | YES | No |

*Figure II.15. : Langages d'exécution de processus*

### 4.3. Constat sur l'approche BPM

A l'heure actuelle, les solutions implémentant l'approche BPM au sein des entreprises se limitent à l'exécution des processus. Les modèles de processus utilisés permettent de :

- formaliser les processus à partir d'un langage commun,
- dérouler les processus pour exécuter les activités qui le composent.

Par contre, l'agilité d'une entreprise induit la capacité d'adapter le déroulement du processus au changement de l'environnement. Cette perspective n'existe pas dans les solutions actuelles implémentant l'approche BPM car seuls des choix correspondants à des situations prédéfinies peuvent être appréhendées par les langages présentés dans les paragraphes précédents. Dans le cadre de nos travaux, nous proposons d'intégrer dans le modèle de processus, les mécanismes nécessaires au pilotage des processus (Cf. § 2.3.3).

## 5. Vers un développement centré architecture

L'intégration des solutions informatiques induit une évolution de l'ingénierie du système d'information de l'entreprise. En effet, d'une logique d'ingénierie qui visait à réaliser des systèmes d'information « clés en main » répondant à un besoin spécifique, il est de plus en plus nécessaire d'intégrer des applications existantes qui n'ont pas été conçues en vue de cette intégration. Aussi, ce paragraphe présente les trois grandes phases qui ont marqué l'évolution de l'ingénierie des systèmes d'information :

- l'ingénierie « clé en main »,

- l'ingénierie par « développement de constituants »,
- l'ingénierie par « intégration de composants ».

## 5.1. L'ingénierie « clé en main » : approche Merisienne

Avant les années 80, la construction d'un système se basait essentiellement sur une ingénierie « clé en main », c'est-à-dire que nous construisions de manière spécifique le nouveau système désiré en fonction d'objectif(s) donné(s), le tout, en partant d'un système existant. Pour ce faire, les approches les plus courantes s'appuyaient sur la méthode MERISE [CHL 89] qui distinguait trois niveaux d'abstraction[16] :

- le ***niveau conceptuel*** concerne les choix relatifs à la gestion du système. La prise en compte de ce niveau permet de répondre à la question « Quoi ? »,
- le ***niveau organisationnel*** traite, quant à lui, les choix d'organisation. Il s'agit au sein de ce niveau de répondre aux questions « Qui ? », « Quand ? » et « Où ? »,
- le ***niveau physique*** aborde les choix purement techniques en permettant de répondre à la dernière question, à savoir « Comment ? ».

Cependant, bien que le développement de systèmes basé sur cette ingénierie « clé en main » ait largement contribué à la construction de nombreux systèmes, l'évolution des techniques de développement, notamment celles liées à la spécialisation des métiers d'ingénierie, a fait évoluer le mode de construction vers une ingénierie basée sur le développement de composants.

## 5.2. L'ingénierie par développement de constituants : approche Orientée Objet

Nous allons présenter, dans un premier temps, les notions générales liées à l'ingénierie et à l'intégration de système. Nous nous appuierons ensuite sur ces notions pour décrire les différentes phases qu'il est nécessaire de mettre en œuvre pour construire un système par développement de constituants.

### 5.2.1. Ingénierie et intégration systèmes [Mei 00]

Le concept d'ingénierie induit deux définitions complémentaires. L'ingénierie représente d'une part, les opérations de conception, de développement et de mise en œuvre des ouvrages faisant appel à plusieurs métiers et, d'autre part, l'ensemble des connaissances, des méthodes, des techniques et des outils permettant de concevoir ces ouvrages. Le concept d'ingénierie est généralement lié à la notion de système. L'ouvrage possède généralement les caractéristiques de complexité et d'hétérogénéité d'un système et il est analysé et conçu sous forme d'un ensemble de constituants réalisables par différents métiers.

Le terme d'intégration peut, quant à lui, être défini comme l'opération de réalisation d'un système par assemblage de ses constituants (ou d'un nouveau système à partir de systèmes existants). Ce concept est nécessairement lié à la notion de système : il s'agit d'assembler différents constituants de manière à ce que les relations confèrent à leur assemblage de nouvelles propriétés.

[16] La méthode MERISE 2 introduit un quatrième niveau (entre organisationnel et physique : le niveau logique qui concerne le choix des logiciels [NEC 92]).

### 5.2.2. Les phases de construction d'un système

Ayant précisé ces deux concepts, nous allons maintenant présenter le développement d'un système suivant trois phases :

- une *phase de définition du système* où les fonctions attendues sont décomposées et allouées à des constituants dont la réalisation relève d'un seul génie[17]. Elle débouche sur la spécification des constituants à faire réaliser par les métiers et celles des interactions qui précisent la manière dont ils devront être intégrés,
- une *phase de réalisation* des constituants par les différents métiers.
- une *phase de construction du système* qui assemble les constituants réalisés en vérifiant que leurs interactions confèrent bien les fonctions attendues au système intégré.

Un consensus semble s'établir pour désigner la première phase sous le nom d'ingénierie système, la troisième sous celui d'intégration système :

- l'**ingénierie système** représente la phase de définition d'un système spécifiant les propriétés attendues et débouchant sur la spécification de ses constituants et la manière de les intégrer,
- l'**intégration système** représente la phase de construction du système par assemblage de ses constituants préalablement réalisés en vérifiant que leurs interactions confèrent au système les propriétés attendues.

Ces différentes phases peuvent être illustrées par le schéma suivant :

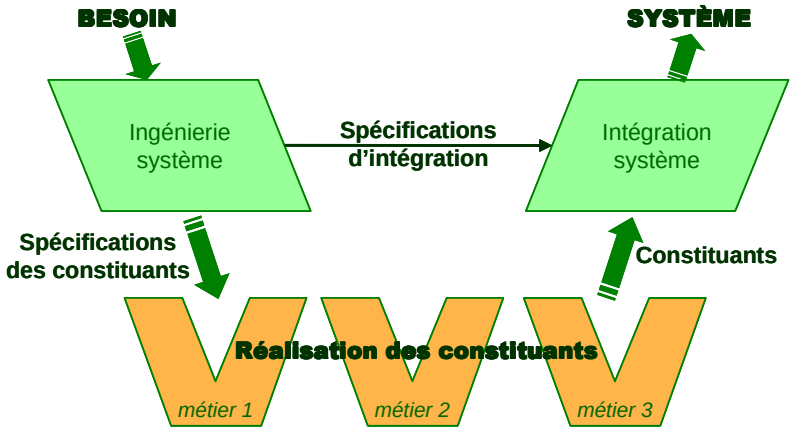



*Figure II.16. : Le développement d'un système*

Ce cycle de développement en trois phases (ingénierie, réalisation des constituants, intégration) s'applique à tout système, il peut donc être utilisé pour la réalisation du système d'information. Cette approche a été à la base de nombreux développements, en particulier dans les cas d'utilisation du paradigme « objet ».

L'utilisation de ce paradigme tout au long du cycle d'ingénierie semblait permettre des développements sans « raccord » favorisant ainsi le raffinement des spécifications. Il ne s'agissait plus par exemple de transformer un diagramme Entité-Association (ou MCD – Modèle Conceptuel des Données) en un schéma relationnel, mais bien de raffiner un diagramme de classes modélisant les objets métiers de l'utilisateur en un diagramme de classes représentant les objets techniques utilisés au sein des applications. Cependant, même si aujourd'hui la représentation des systèmes d'information dans des formalismes communs et

[17] Le terme génie représente l'ensemble des connaissances et des techniques liées à un métier.

adaptables aux différentes étapes du cycle de vie facilite les développements sans « raccord », elle ne les garantit pas. En d'autres termes, l'approche objet n'a en rien simplifié les processus de développement ; les transformations entre modèles ne sont pas plus simples ni plus naturelles qu'elles ne l'étaient dans des méthodes de développement traditionnelles telles que MERISE. Les principes fondamentaux de cette approche peuvent être résumés de la façon suivante [CHL 99] :

- l'importance d'inscrire la définition de l'architecture du système d'information dans un processus dynamique permettant de minimiser les ruptures en matière de conception et de réalisation lors de l'intégration des évolutions des besoins de l'entreprise et de celles des technologies de l'information,
- la nécessité d'analyser précisément les besoins pérennes avant de faire des choix technologiques,
- l'importance d'adopter le paradigme « objet » (pour son pouvoir d'abstraction et d'encapsulation) de manière systématique comme cadre pour structurer et modéliser le système d'information,
- l'importance de concevoir le système d'information comme un système ouvert décentralisé qui s'appuie sur des structures réutilisables métier correspondant directement aux processus importants de l'entreprise.

Comme dans la plupart des domaines d'ingénierie, les approches basées sur le paradigme « objet » vise à réduire les coûts et les délais des développements tout en améliorant la qualité des produits. Cependant, pour atteindre ces objectifs, il est nécessaire de mettre en œuvre les évolutions suivantes :

- la maîtrise, voire l'automatisation, des activités de développement,
- une gestion efficace de composants réutilisables.

Ces évolutions ont conduit à l'apparition de l'ingénierie par intégration de composants.

## 5.3. L'ingénierie par intégration de composants : approche orientée architecture logicielle

Toute véritable industrialisation se caractérise par la conception et le développement d'un ensemble de produits à partir d'éléments réutilisables. Cette affirmation s'applique à différents domaines industriels comme par exemple la construction automobile ou l'électronique. Dans le secteur du logiciel, il existe depuis fort longtemps des bibliothèques de programmes de calculs scientifiques, de statistiques, etc. Les informaticiens des banques ou des grandes entreprises ont été les premiers à appliquer ces principes de réutilisation en raison de la centralisation et de l'homogénéité de leur système d'information. Ce besoin de réutilisation est l'une des bases du succès de l'approche objet. En effet, les langages à objet ont été les premiers à répondre à ce besoin en mettant à la disposition du développeur des bibliothèques de composants logiciels. Ces bibliothèques constituent à l'heure actuelle de véritables boîtes à outils couramment utilisées et intégrées aux outils de développement.

Cependant, le plus souvent, de telles bibliothèques n'apparaissent qu'au niveau de la phase de développement et sont utilisées en vue d'améliorer la productivité des développeurs. De nombreux progrès restent donc à faire pour introduire cette notion de réutilisation dans les différentes étapes de spécification et de conception des systèmes d'information. L'objectif est en fait de permettre aux concepteurs de travailler à un niveau d'abstraction plus élevé que celui correspondant au niveau des classes logicielles. Depuis quelques années, des travaux dans les domaines du génie logiciel, des systèmes d'information et de l'intelligence artificielle

vont dans ce sens sans qu'une réelle harmonisation ne soit encore actuellement envisageable. Une grande variété de composants réutilisables a été proposée : les objets métiers [AnC 98], les modèles de domaines génériques [MaS 92] [MaS 93], les patrons de conception [GHJ 95], les patrons d'analyse [CNM 96], les ontologies [Gru 93], les *frameworks* [Joh 92] [Rev 96], etc. Dans tous les cas, un composant est vu comme une solution testée et acceptée pour résoudre un problème fréquemment rencontré lors du développement de logiciels.

Si les propositions concernant les types de composants sont nombreuses, les techniques de conception à base de composants restent, quant à elles, relativement embryonnaires. Il s'agit en effet de remplacer les démarches traditionnelles de développement par deux types de processus complémentaires :

- des processus de développement **pour** la réutilisation dont l'objectif est d'identifier, de spécifier, de concevoir et de développer des composants réutilisables (ce développement peut être réalisé par une approche orientée objet),
- des processus de développement **par** la réutilisation dont l'objectif est d'utiliser les composants réutilisables pour spécifier, concevoir et développer une architecture logicielle capable d'intégrer tout type de composants (les objets métiers, les patrons, les *frameworks*, etc.).

Aujourd'hui, un problème critique dans la conception et la construction d'un système logiciel complexe est son architecture, c'est-à-dire l'organisation des composants qui constituent ce système (Cf. Chapitre 1, § 3.3). Une architecture adaptée dote le système de propriétés de performance, de fiabilité, de portabilité et d'interopérabilité. Au contraire, une « *mauvaise »* architecture peut avoir des conséquences désastreuses sur le système [Gar 00]. C'est pourquoi, au cours de la dernière décennie, les architectures logicielles ont fait l'objet d'une attention croissante.

### 5.3.1. La notion d'architectures logicielles

Plusieurs définitions du terme « architectures logicielles » sont proposées dans la littérature [Boa 95] [BCK 99] [GSO 92] [PeW 92], mais il est largement accepté qu'une architecture est définie par un ensemble de composants (par exemple, filtres, objets, bases de données, serveurs, etc.) et leurs interactions (par exemple, appels de procédures, envoi de messages, émission d'événements, etc.) [GaS 93]. En d'autres termes, une architecture spécifie les caractéristiques d'un système en définissant le type des composants, leur nombre, la manière dont les composants interagissent entres eux et quelles sont les propriétés structurelles et comportementales qui doivent être respectées [RDT 97]. Par conséquent, la description architecturale d'un système spécifie :

- sa **structure** : composants et interactions,
- son **comportement** : fonctionnalités et protocoles de communication, dynamisme, évolution,
- ses **propriétés globales** : propriétés fonctionnelles ou non-fonctionnelles.

Les architectures de la plupart des systèmes logiciels ont longtemps été décrites de façon informelle (diagrammes où les composants logiciels sont des « boîtes » et les interactions des « lignes »). Cette description induit d'une part, des difficultés au niveau de son interprétation et, d'autre part, des limitations [AAG 95]. Aussi, des langages de description d'architectures (LDA) ont été définis. Ils offrent de nombreux avantages par rapport aux approches informelles, comme par exemple la précision, la capacité à prouver des propriétés, et la possibilité d'analyser la structure architecturale. Ainsi spécifiées, les architectures logicielles jouent un rôle important dans au moins six aspects du développement logiciel [Gar 00] :

1. **Compréhension** : les architectures logicielles rendent plus facile la compréhension du fonctionnement de systèmes complexes en les représentant à un haut niveau d'abstraction,
2. **Réutilisation** : les descriptions architecturales supportent la réutilisation à de multiples niveaux. Les travaux actuels, dans le domaine de la réutilisation, se concentrent généralement sur l'utilisation de bibliothèques de composants. La conception orientée architecture supporte, en plus, la réutilisation de composants complexes et des structures dans lesquelles ces composants peuvent être intégrés. Ceci est prouvé par de nombreux travaux existants dans les domaines des *Domain-Specific Software Architectures*, des architectures de référence, ou des patrons de conceptions [MeG 92] [BMR 96],
3. **Construction** : une description architecturale fournit un plan de développement en indiquant les composants principaux et les dépendances entre eux,
4. **Evolution** : l'architecture d'un système peut décrire la manière dont ce système est censé évoluer. La définition explicite des limites d'évolution d'un système permet de faciliter sa maintenance et d'estimer plus précisément les coûts des modifications. De plus, les descriptions architecturales distinguent les aspects fonctionnels des composants de la façon dont ces composants interagissent entre eux. Cette séparation permet de modifier facilement les mécanismes de connexion, ce qui favorise l'évolution en termes de performance, d'interopérabilité et de réutilisation,
5. **Analyse** : les descriptions architecturales fournissent des moyens d'analyse, tels que la vérification de la consistance d'un système [AlG 94] [LKA 95], la vérification de la conformité aux contraintes imposées par un style architectural [AAG 93], la vérification de la conformité à des attributs qualité [CBK 95], l'analyse de dépendance [SRW 93], ainsi que des analyses spécifiques au style à partir des architectures construites [CoS 93] [MDE 95] [GAO 94],
6. **Gestion** : l'expérience a montré que la définition précise d'une architecture logicielle est un facteur clé dans la réussite d'un processus de développement logiciel. L'évaluation d'une architecture mène à une meilleure compréhension des besoins, des stratégies d'implémentation, et des risques potentiels [BBH 94].

La figure suivante représente le processus de développement centré architecture ainsi que les acteurs : l'architecte d'application, l'ingénieur, ainsi qu'éventuellement l'analyste (cette tâche peut être automatisée).

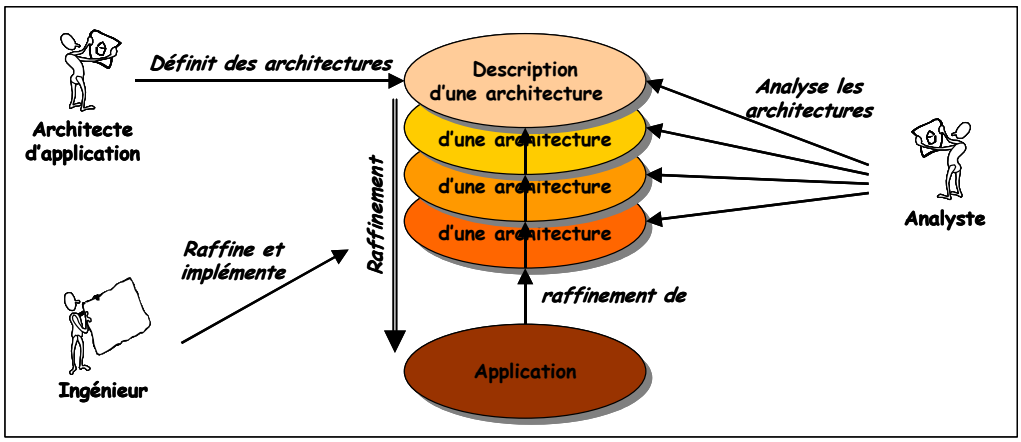



*Figure II.17. : Processus de développement architectural*

L'architecte a pour rôle de définir l'architecture qui servira de base au développement de l'application. L'ingénieur raffine cette architecture de façon à s'approcher, petit à petit, de l'application finale. Pour cela, il implémente les composants ainsi que leurs interactions (les connecteurs) en respectant la structure et les propriétés définies par l'architecture de départ. A chaque étape de raffinement, l'analyste doit être en mesure de vérifier que l'architecture raffinée est conforme à l'architecture du niveau d'abstraction supérieur. Ce processus permet de garantir que l'application obtenue respecte les propriétés fonctionnelles, structurelles et comportementales définies par l'architecte en accord avec le client et les utilisateurs.

Avant de présenter différents langages de description d'architectures, il nous semble important d'introduire la notion de style architectural car nous utilisons ce concept dans le cadre de nos travaux.

### 5.3.2. Notion de style architectural

Rappelons qu'un style architectural précise les propriétés et les contraintes qui fixent les règles et les limites de construction de l'architecture (Cf. Chapitre 1, § 5). Il caractérise donc une famille de systèmes qui ont les mêmes propriétés structurelles et sémantiques [AAG 93] (par exemple, « *pipe and filter* », « *client-serveur* », etc*.)*. L'objectif des styles architecturaux est de simplifier la conception des logiciels et la réutilisation, en capturant et en exploitant la connaissance utilisée pour concevoir un système [MKM 97]. Un style architectural est moins contraignant et moins complet qu'une architecture spécifique. Il spécifie uniquement les contraintes les plus importantes, au niveau par exemple de la structure, du comportement, de l'utilisation des ressources des composants et des connecteurs [Abd 96].

En d'autres termes, un style architectural fournit [Gar 95] :

- un vocabulaire pour spécifier les types spécifiques de composants utilisables,
- des règles de conception ou des contraintes pour spécifier les compositions d'éléments qui sont permises,
- une interprétation sémantique pour définir la signification des compositions d'éléments contraintes par les règles de conception,
- les analyses qui peuvent être appliquées sur les systèmes construits à partir de ce style.

D'une façon générale, les styles architecturaux permettent à un développeur de réutiliser l'expérience concentrée de tous les concepteurs qui ont précédemment fait face à des problèmes similaires [KlK 99]. En outre, l'utilisation des styles architecturaux comporte des intérêts précis :

- elle favorise la réutilisation dès la conception du système [MoG 96],
- elle autorise une réutilisation significative de code,
- elle favorise la normalisation des familles d'architecture, ce qui facilite la compréhension de l'organisation d'un système,
- elle autorise l'utilisation d'analyses spécifiques au style concerné [CiM 96].

La figure suivante présente un processus de développement général qui inclut la définition formelle et l'exploitation de styles architecturaux. Ce schéma reprend et étend le processus de développement architectural présenté auparavant (Cf. § 5.3.1).

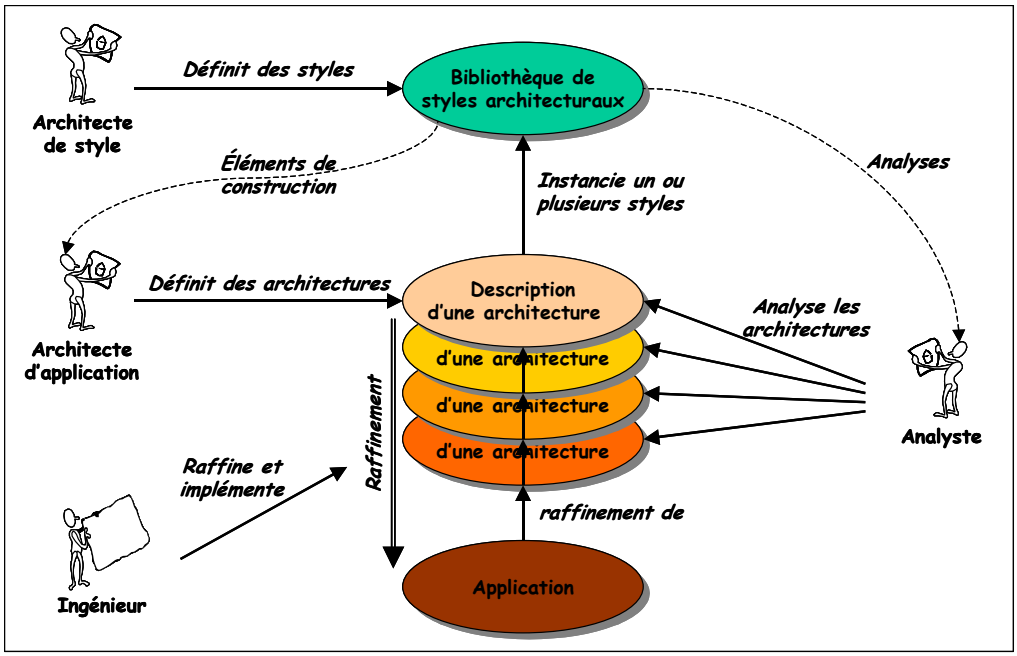



*Figure II.18. : Processus de développement architectural orienté styles*

La première étape de ce processus est la formalisation du style architectural ou des styles architecturaux. Au fur et à mesure des définitions de styles, une bibliothèque peut être définie. L'architecte d'application instancie alors le ou les styles qui correspondent à ses besoins spécifiques et les utilise comme éléments de construction pour définir une architecture de base. La suite du développement est identique au processus précédemment étudié : l'ingénieur raffine petit à petit l'architecture de base en architectures de plus en plus concrètes jusqu'à l'obtention de l'application finale. Il est toutefois à noter que l'utilisation d'un tel processus de développement permet à l'analyste d'appliquer aux différentes architectures des analyses définies par les styles utilisés. Les analyses disponibles ne sont donc plus uniquement des analyses génériques, applicables à toutes les architectures. Ces analyses peuvent aussi être des analyses spécifiques à un domaine d'application particulier, applicables uniquement aux architectures définies à l'aide des styles correspondant à ce domaine.

Les styles architecturaux ont longtemps été définis et utilisés comme des guides de conception informels, n'ayant pas de langages dédiés à leur exploitation. Les travaux menés, depuis lors, ont souligné l'intérêt de formaliser les styles [AAG 95] et de nombreux langages dédiés à cette tâche ont été développés.

### 5.3.3. Les langages de description d'architecture (LDA)

Les Langages de Description d'Architecture supportent le développement centré architecture à travers au moins un style (celui représentant la sémantique et les mécanismes du langage) [MeT 97] ; ce style est plus ou moins générique selon le domaine couvert par le langage. Nous illustrons ces propos en présentant rapidement différents langages de description d'architecture [LCO 02] [AAR 03] :

- ACME [GMW 97] et π-SPACE [ChO 01] ne sont pas spécifiques à un domaine car ils s'appuient sur un style générique,
- Adage [CoS 93] supporte la description de structures architecturales pour le guidage et la navigation dans le domaine de l'aviation,
- AESOP supporte l'utilisation de styles architecturaux,

- C2 [MOR 96] supporte la description de systèmes d'interfaces utilisateurs utilisant un style événementiel,
- Darwin [MDE 95] supporte l'analyse de systèmes de transmission de messages distribués,
- META-H [BiV 93] [BEJ 96] est un langage spécifique aux architectures des systèmes multiprocesseurs temps réel pour l'aviation,
- Rapide [LKA 95] permet de simuler les architectures et propose des outils pour analyser les résultats de ses simulations,
- SADL [MQR 95] fournit une base formelle pour le raffinement architectural,
- UniCon [SDK 95] possède un compilateur de haut niveau pour les architectures qui supportent l'utilisation de types de composants et de connecteurs hétérogènes,
- Wright [AlG 97] supporte la spécification formelle et l'analyse des interactions entre les composants architecturaux.

Il existe également des « méta-langage de description d'architecture ». Citons en exemple AML [Wil 99] qui permet de spécifier la sémantique des langages en caractérisant une architecture comme un ensemble d'éléments (*element*) liés par des relations (*relationship*).

Dans ce contexte, les LDAs qui ont été notamment étudiés dans le cadre du projet ArchWare sont [AAR 03] : AML [Wil 99], AESOP [GAO 94], ARMANI [Mon 01], UNICON-2[18] [DeL 96] et σπ-SPACE [LCO 01]. Ces langages proposent des formalismes et des outils permettant de définir et d'exploiter efficacement les styles architecturaux. Leur utilisation offre de multiples bénéfices, la description formelle de l'architecture rend possible l'utilisation de mécanismes pour représenter les architectures, leur analyse, leur raffinement, et la génération automatique du code correspondant.

# 6. Conclusion

L'agilité de l'entreprise face à son environnement est un facteur essentiel de succès dans le contexte industriel actuel. Elle nécessite la mise en place de nouvelles organisations capables d'identifier, de formaliser et de piloter les processus transversaux de l'entreprise. Cette maîtrise des processus nécessite un nouveau paradigme du système d'information (Cf. Chapitre 1, § 2.2.3.3) mais ce dernier n'est pas suffisant. En effet, la gestion des processus nécessite la maîtrise de l'évolution du système d'information à travers le point de vue « métier » et le point de vue « donnée ».

De plus, le déroulement des processus s'appuie sur les applications logicielles ayant leurs fonctionnalités propres. Il est donc nécessaire de s'appuyer sur une démarche d'intégration pour unifier les données exploitées par les différentes activités des processus. Les plates-formes EAI permettent d'une part, d'unifier les données partagées par les applications et, d'autre part, d'intégrer les fonctions supportées par ces mêmes applications.

Nous avons montré que l'approche BPM constitue une première étape d'intégration des points de vue « métier » et « donnée » en proposant des langages capables d'une part, de décrire les processus métiers et, d'autre part, de dérouler ces processus en coordonnant les activités supportées par les applications logicielles. Suite à cette présentation, nous avons constaté que les approches BPM actuelles ne permettent pas de piloter les processus. Aussi, nous

[18] Extensions de UNICON [SDK 95] et π-SPACE [ChO 01] pour les styles architecturaux

proposons, dans le chapitre suivant, un modèle capable de piloter les processus de l'entreprise. Ce modèle est à la base d'un langage capable d'intégrer les points de vue « métier » et « donnée ».

Enfin, cette nouvelle vision du système d'information induit une nouvelle démarche d'ingénierie. Dans le cadre du projet ArchWare, nous nous appuyons sur une ingénierie par intégration de composants pour décrire, concevoir, spécifier et développer le système d'information d'une entreprise. Cette démarche s'appuie sur un langage de description d'architecture supportant la définition de styles architecturaux. Dans les chapitres suivants, nous utiliserons cette notion de style pour formaliser notre modèle de pilotage et intégrer les deux points de vue. Nous nous baserons, pour ce faire, sur le concept de Composant Industriel (Cf. Chapitre 1, § 4.3.4) pour formaliser les processus de l'entreprise.

# Chapitre 3 :
# Notre cadre de modélisation

# Chapitre 3 :
# Notre cadre de modélisation

## 1. Introduction

L'objectif de ce chapitre est de présenter le cadre de modélisation que nous utilisons pour formaliser le fonctionnement du système d'information d'une entreprise évoluant dans un environnement concurrentiel. Nous avons montré que ce contexte industriel conduit les entreprises à s'organiser autour d'une approche processus (Cf. Chapitre 2, § 2.3). Aussi, nous considérons qu'un système d'information d'une entreprise est structuré en (Cf. Chapitre 1, § 2.2.3.3) :

- un système de gestion de connaissances (SGC),
- un système de gestion de processus (SGP),
- un système de gestion de ressources (SGR).

Nous allons donc présenter dans le cadre de ce chapitre, un cadre de modélisation, le modèle OLYMPIOS, nous permettant de formaliser le système de gestion de processus. De plus, nous avons montré que le paradigme d'ambivalence nous conduit à appréhender la réalité opérationnelle d'une entreprise à travers deux points de vue : le point de vue « métier » et le point de vue « donnée » (Cf. Chapitre 1, § 3.1). Le cadre de modélisation doit donc préciser les concepts liés à la gestion des processus dans chacun de ces points de vue. Ces concepts concernent la formalisation du processus et de son pilotage.

Nous considérons que pour appréhender la réalité opérationnelle de l'entreprise à travers ces deux points de vue, nous devons étudier d'une part, les architectures industrielles relatives au point de vue « métier » et, d'autre part, les architectures logicielles relatives au point de vue « donnée » (Cf. Chapitre 1, § 3.3). Nous proposons donc, dans le cadre de ce chapitre, de caractériser les architectures industrielles et logicielles en nous appuyant sur les concepts issus du modèle OLYMPIOS. Rappelons que ce modèle formalise le pilotage des processus qui peut induire une évolution de ces architectures. La caractérisation des architectures nécessite de caractériser les composants, les connecteurs et leur configuration (Cf. Chapitre 1, § 3.3).

En conséquence, la première partie de ce chapitre est consacrée à la définition des concepts propres à une architecture à travers les notions de composants, de connecteurs et de configuration. Dans un second temps, nous présentons le modèle de pilotage du modèle OLYMPIOS et, plus particulièrement, un modèle de pilotage des processus. La formalisation des architectures industrielles et logicielles évolutives est présentée dans la troisième partie de ce chapitre. Enfin, la dernière partie de ce chapitre présente les interactions entre ces différentes architectures.

## 2. Le modèle d'architecture « composant / connecteur »

Nos travaux s'appuyant sur la notion d'architectures industrielles et logicielles, ce paragraphe s'appuie sur différents résultats issus de travaux réalisés dans le domaine du génie logiciel pour définir le concept d'architecture [Boa 95] [BCK 99] [GSO 92] [PeW 92]. Rappelons qu'une architecture est définie par un ensemble de composants, un ensemble de connecteurs

et une configuration [GaS 93] (Cf. Chapitre 1, § 3.3). En d'autres termes, une architecture spécifie la structure, le fonctionnement et le comportement d'un système en définissant le type des composants, leur nombre, la façon dont ses composants interagissent et les propriétés (structurelles, fonctionnelles et comportementales) qu'ils doivent respecter. La formalisation d'une architecture nous conduit donc à définir les notions de [Cha 02] :

- composant,
- connecteur,
- configuration.

## 2.1. La formalisation d'une architecture

Nous rappelons qu'un ***composant*** est une unité de calcul ou de stockage de données dans une architecture (Cf. Chapitre 1, § 3.3), il correspond au constituant de base d'une architecture. Les composants sont indépendants de la structure dans laquelle ils sont placés, et possèdent une interface permettant de communiquer avec leur environnement. Il comprend deux parties distinctes : le **comportement** et les **ports**. Le comportement spécifie les traitements réalisés par le composant tandis que les ports formalisent son interface. Un composant répond à certaines propriétés [BTH 00], il est :

- autonome, c'est-à-dire indépendant de l'architecture dans laquelle il est placé,
- apte à être assemblé avec d'autres composants ; les mécanismes d'assemblage s'effectuent uniquement par les ports du composant,
- contraint par l'architecture dans laquelle il évolue,
- typé et instanciable au sein de l'architecture.

Un ***connecteur*** définit le protocole d'échange entre deux ou plusieurs composants (Cf. Chapitre 1, § 3.3). Il modélise donc les interactions entre les composants et spécifient les règles d'échange que les composants doivent respecter. Tout comme le composant, le connecteur comprend deux parties distinctes : le comportement qui spécifie le protocole d'échange et les ports qui formalisent les interfaces.

Le ***composite*** est un assemblage de composants disposant de ports. Les ports des composants qui ne sont pas utilisés dans l'assemblage sont les ports du composite. Le composite dispose des mêmes propriétés que le composant.

Une ***configuration*** précise la manière spécifique dont les composants et les connecteurs sont connectés.

Ayant précisé la notion d'architecture, il nous reste à définir les concepts liés à leur évolution. En effet, nous avons montré, dans le cadre du chapitre 1, que ces concepts sont primordiaux pour permettre à l'entreprise d'être agile (adaptation et anticipation) (Cf. Chapitre 1, § 2.1). Ces concepts sont à la base des propositions développées pour les points de vue « métier » et « donnée ».

## 2.2. Evolution d'une architecture

Au niveau du point de vue « métier », l'évolution d'une architecture permet de modifier le processus de référence (le référent) pour permettre aux entreprises de s'adapter à l'évolution de leur environnement (Cf. Chapitre 1, § 4.3.3). D'une façon similaire, l'évolution des technologies de traitement de l'information et de la communication nécessite la prise en compte des concepts d'évolution au niveau du point de vue « donnée ».

Pour ce faire, des travaux liés aux langages de description d'architectures proposent des concepts spécifiques aux architectures dynamiques [BCO 99] [BCO 00] [ChO 00] [ChO 01]. Une architecture dynamique est une architecture qui peut évoluer en cours d'exécution. Les manières communes selon lesquelles une architecture peut évoluer sont [Med 96] :

- l'ajout ou la suppression d'éléments dans l'architecture,
- la modification d'un élément (ajout ou suppression de port),
- la modification de la topologie (par exemple, nouvelle connexion de composants et de connecteurs).

Ce constat nécessite la définition de :

- port dynamique,
- composant dynamique,
- connecteur dynamique,
- configuration dynamique.

Le ***port dynamique*** représente une interface qui peut être créée ou supprimée en cours d'exécution. Il est donc nécessaire de disposer de mécanismes capables de manipuler ces ports (création, suppression).

Le ***composant dynamique*** représente un composant qui peut être créé, modifié ou supprimé en cours d'exécution. La modification d'un composant permet de lui attacher ou détacher des ports dynamiques. Il est donc nécessaire de disposer de mécanismes capables de :

- manipuler ces composants (création, suppression),
- manipuler les ports dynamiques d'un composant (attachement, détachement) pour définir la manière dont ces composants interagissent avec les connecteurs de l'architecture.

Le ***connecteur dynamique*** représente un connecteur qui peut être créé, modifié ou supprimé en cours d'exécution. Tout comme pour le composant, la modification d'un connecteur permet de lui attacher ou détacher des ports dynamiques. Il est donc nécessaire de disposer de mécanismes capables de :

- manipuler ces connecteurs (création, suppression),
- manipuler les ports dynamiques d'un connecteur (attachement, détachement) pour définir la manière dont ces connecteurs interagissent avec les composants de l'architecture.

Une ***configuration dynamique*** précise l'organisation des composants dynamique et des connecteurs dynamiques en identifiant les connexions entre leurs ports (dynamiques). De plus, elle précise les règles d'évolution de l'organisation (création, modification, suppression de composants, de connecteurs et de connexions)

En conséquence, les mécanismes d'évolution permettent de décomposer une architecture en cours d'exécution, de créer ou modifier des composants ou connecteurs et de recomposer une nouvelle architecture. Nous montrerons dans les chapitres suivants que ces mécanismes sont à la base de nos travaux. Cette évolution d'architecture en cours d'exécution peut être représentée par le schéma suivant.

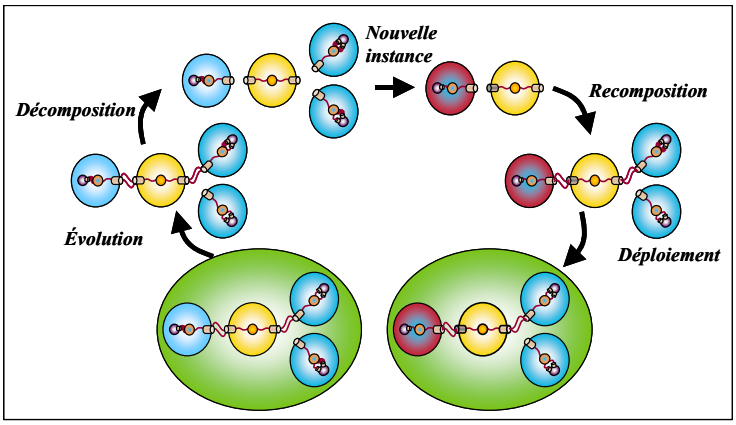



*Figure III.1. : L'évolution d'une architecture en cours d'exécution*

Suite à ces constats, nous pouvons dès à présent considérer qu'un langage de description d'architecture est un langage qui doit explicitement modéliser les composants, les connecteurs et leur configuration [MeT 00]. De plus, pour être vraiment utile et utilisable, il doit fournir des mécanismes supportant l'évolution des architectures. Le projet ArchWare propose un langage de description d'architecture, basé sur une approche formelle, qui nous permet de décrire des architectures capables d'évoluer tout en vérifiant des propriétés (fonctionnelles, structurelles, etc.).

### 2.3. Les conséquences sur notre approche

Suite à cette présentation du modèle « composant / connecteur » qui précise les différents éléments nécessaires à la modélisation d'une architecture, nous pouvons préciser quelques conséquences sur la construction du système d'information et, plus particulièrement, sur le système de gestion de processus.

Tout d'abord, il est nécessaire d'identifier les composants et les connecteurs pour « découpler » les unités de traitement (composants) de leurs interactions (connecteurs). De plus, la représentation d'une architecture à travers cette approche nous permet de spécifier clairement les limites du système que nous construisons et ses différentes règles d'évolution. D'autre part, l'utilisation de ce modèle « composant / connecteur » nous permet de formaliser les règles de construction qui précisent les liens entre les différents éléments de l'architecture.

## 3. Formalisation du pilotage

Ce paragraphe présente, dans un premier temps, les concepts de base d'un modèle de pilotage. Avant de décrire la manière dont ces concepts sont appréhendés au sein du modèle OLYMPIOS [Din 98] [Bra 02] [Cha 02] [Thé 02] pour formaliser le modèle de pilotage d'un processus, nous présentons les concepts de base d'un processus.

### 3.1. Modèle de pilotage d'un système [Thé 02]

Rappelons qu'à travers notre approche, le pilotage d'un système doit permettre de le faire évoluer vers un but initialement fixé, ceci via différentes situations intermédiaires (Cf. Chapitre 1, § 2.2.2). En effet, un système évolue d'une situation initiale vers une situation qui est évaluée plus ou moins conforme à une situation espérée (définie par le référent). Dans le cas d'une non-conformité significative, il est nécessaire de construire un nouveau référent du système capable d'atteindre le but fixé et la situation obtenue correspond alors à une nouvelle

situation initiale du processus. Cette démarche est renouvelée pour chaque situation identifiée par le référent.

Nous rappelons également que le pilotage d'un système nécessite quatre phases (Cf. Chapitre 1, § 2.2.2) :

- une phase d'**initialisation** qui fait appel à un générateur de plans pour construire un référent,
- une phase d'**évolution** qui contrôle l'évolution du système en s'appuyant sur un modèle de conduite,
- une phase de **correction** qui, suite à la détection d'une non-conformité, fait appel à un générateur de plans pour déterminer les actions correctives capables d'atteindre le but du système en définissant un nouveau référent,
- une phase de **terminaison** qui évalue la situation finale obtenue par rapport à la situation espérée.

En fonction du modèle de décision proposé par Simon (Cf. Chapitre 1, § 2.2.3.1.b), les non-conformités détectées peuvent être traitées de trois façons différentes :

- la non-conformité est anticipée et le traitement est intégré dans le plan d'actions (dans ce cas, la phase de correction doit simplement mettre en œuvre le traitement),
- la non-conformité n'est pas anticipée et il existe un traitement capable d'atteindre l'objectif fixé (la phase de correction doit alors construire un nouveau plan d'action pour intégrer le traitement),
- la non-conformité n'est pas anticipée et il n'existe pas de traitement capable d'atteindre l'objectif fixé (la phase de correction doit alors faire appel au système de gestion d'objectif pour qu'il lui fournisse un nouvel objectif).

Nous précisons que dans le cadre de nos travaux, nous ne traitons pas la remise en cause de l'objectif.

Dans le cadre du pilotage d'un processus, cette approche nous conduit à définir deux modèles :

- un **modèle de conduite** qui formalise la phase d'évolution d'un processus,
- un **modèle de pilotage** qui formalise les phases d'initialisation, de correction et de terminaison d'un processus et intègre le modèle de conduite.

Avant de présenter ces deux modèles, nous présentons le modèle de processus proposé par le modèle OLYMPIOS.

## 3.2. Modèle de processus [Din 98] [Thé 02]

Rappelons que nous définissons un processus comme « *un enchaînement partiellement ordonné d'exécution d'activités qui, à l'aide de moyens techniques et humains, transforme des flots d'entrée en flots de sortie en vue de réaliser un objectif* » (Cf. Chapitre 2, § 2.3). Son comportement est défini par un graphe d'état, qui précise les différents états du processus (état initial, états intermédiaires et état final) et les transitions entre ces états. De par ce graphe, une activité contribue à la transformation d'un état d'entrée (état initial ou état intermédiaire) en un état de sortie (état intermédiaire ou état final) en s'appuyant sur des ressources. La formalisation d'un processus nécessite donc, dans un premier temps, de préciser le concept d'activité qui, au sein du modèle OLYMPIOS, est caractérisé par des flots, une opération, des

ressources, un objectif, un résultat et un indicateur de performance. L'enchaînement des activités est formalisé par la notion de graphe d'ordres.

Dans le modèle OLYMPIOS, ces différents concepts sont formalisés par[19] :

- une **représentation ensembliste** qui précise les caractéristiques propres à un concept et les différentes fonctions définies sur ce concept ; cette représentation exprime les aspects structurels et fonctionnels d'un processus,
- des **diagrammes d'état** qui expriment les aspects dynamiques.

Une illustration de ces différents concepts est proposée dans le chapitre 5 de ce mémoire.

### 3.2.1. Les composantes d'une activité

Le modèle OLYMPIOS identifie les composantes suivantes pour une activité [Din 98] :

- les flots,
- l'opération,
- les ressources,
- l'objectif,
- le résultat,
- l'indicateur de performance.

Un **flot** caractérise un ensemble d'articles transformés par une activité. Un **article** est un produit ou un service auquel est associée une unité de mesure. Un **produit** est caractérisé par un identificateur et un ensemble d'attributs. Un **attribut** est caractérisé par un identificateur, une valeur définie dans un domaine de valeurs et un domaine de référence :

- le **domaine de valeurs** précise l'ensemble des valeurs possibles pour une propriété,
- le **domaine de référence** définit, quant à lui, les valeurs attendues pour cette propriété. Par construction, un domaine de référence doit toujours être inclus dans le domaine de valeurs.

Un **flot élémentaire** précise la quantité d'articles identiques transformés par l'activité. Il est donc caractérisé par un article (identificateur) et une valeur numérique qui précise la quantité. Un flot est constitué d'un ensemble de flots élémentaires pour décrire l'ensemble des articles transformés par l'activité.

L'**opération** caractérise la transformation réalisée par une activité. Dans le modèle OLYMPIOS, l'opération est considérée comme « **atomique** » à un niveau de description donnée, c'est-à-dire que seuls les articles en entrée et en sortie de l'activité sont observables. Par conséquent, une opération est caractérisée par un identificateur, des articles en entrée, des articles en sortie et un ensemble d'attributs permettant de représenter les caractéristiques spécifiques de l'opération.

Dans le modèle OLYMPIOS, une **ressource**, également désignée ressource de fonctionnement, caractérise le « processeur » support d'une activité. Par conséquent, une ressource est un acteur ou un moyen (matériel ou application logicielle). Rappelons que dans le cadre de nos travaux, nous nous intéressons plus particulièrement aux applications logicielles, aussi, dans la suite de ce travail, nous ne considérons que les ressources qui sont des applications logicielles.

[19] Une description détaillée de ces différents concepts est fournie dans [Din 98] [Thé 02].

L'**objectif** d'une activité précise ce que doit réaliser l'activité. Dans le cadre de nos travaux, l'objectif est défini par :

- la quantité d'articles à réaliser par l'activité,
- les contraintes de coût et de délai fixées pour l'activité.

En conséquence, un objectif est caractérisé par un flot, un attribut économique et un attribut temporel.

Le **résultat** de l'activité précise la quantité d'articles réalisés, le coût et la date d'obtention. Le résultat d'une activité est exprimé dans le même univers de discours que celui de l'objectif. Par conséquent, un résultat est caractérisé par un flot, un attribut économique et un attribut temporel.
Suite à l'exécution d'une activité, le résultat est comparé à l'objectif. Cette comparaison « mesure » l'écart entre une situation réelle et une situation espérée. L'**indicateur de performance** exprime un niveau de satisfaction qui traduit l'interprétation de la mesure dans un contexte donné [Ber 97].

### 3.2.2. L'enchaînement d'activités

Ayant présenté les différentes caractéristiques d'une activité, nous allons maintenant préciser les concepts permettant de décrire l'enchaînement de ces activités au sein d'un processus.

Un **ordre** contextualise la réalisation d'une activité. Il est donc caractérisé par un identificateur, une opération, un flot d'entrée et un flot de sortie. Par exemple, un ordre de fabrication est identifié par un numéro, il précise l'opération à réaliser, la quantité d'articles en entrée et en sortie de l'activité.

L'enchaînement des ordres est exprimé par trois opérateurs :

- le chaînage d'ordres,
- le pontage d'ordres,
- la divergence d'ordres.

Le **chaînage d'ordres**, noté $C_o$, exprime une séquence entre deux ordres. Dans ce cas, le flot de sortie de l'activité « amont » doit être égal au flot d'entrée de l'activité « aval ».

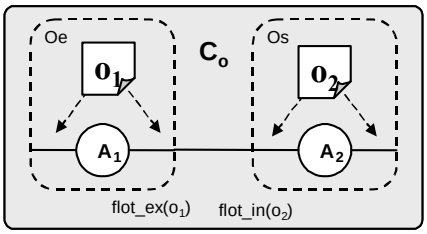



*Figure III.2. : Chaînage d'ordres*

Le **pontage d'ordres**, noté $P_n$, décrit l'enchaînement entre plusieurs ordres et un ordre. Le nombre d'ordres en « amont » est appelé **degré de pontage**. La composition des flots de sortie des activités « amont » est égale au flot d'entrée de l'activité « aval ».

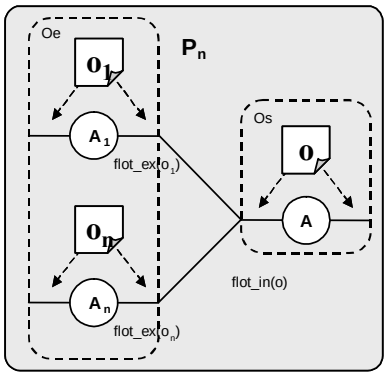



*Figure III.3. : Pontage d'ordres*

La **divergence d'ordres**, noté $\mathbf{D_n}$, décrit l'enchaînement entre un ordre et plusieurs ordres. Le nombre d'ordres en « aval » est appelé **degré de divergence**. Le flot de sortie de l'activité « amont » est égal à la composition des flots d'entrée de l'activité « aval ».

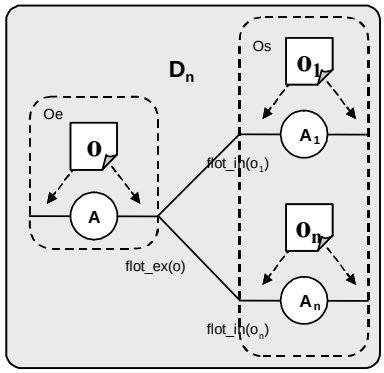



*Figure III.4. : Divergence d'ordres*

### 3.2.3. Définition d'un processus

Ayant défini les concepts liés à l'activité et au graphe d'ordres, nous pouvons maintenant caractériser un processus par une **fonction de transition** entre ses états, cette fonction s'appuie sur un graphe d'enchaînement d'ordres qui représente l'enchaînement des ordres à réaliser pour atteindre l'objectif du processus.

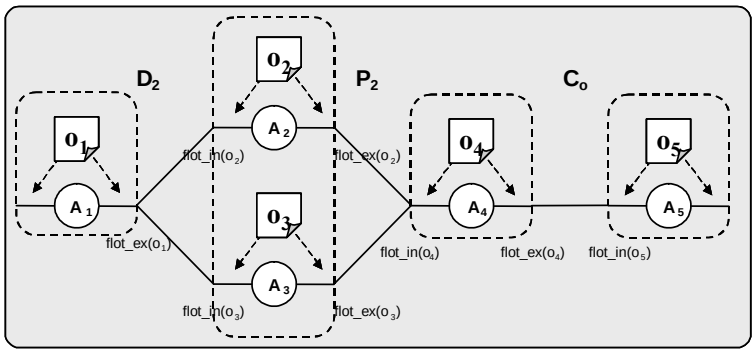



*Figure III.5. : Graphe d'enchaînement d'ordres*

## 3.3. Le pilotage des processus

Avant de présenter le modèle de pilotage de processus proposé par le modèle OLYMPIOS, nous définissons le modèle de conduite qui formalise la phase d'évolution d'un modèle de pilotage.

### 3.3.1. Le modèle de conduite d'un processus

Le modèle de conduite d'un processus s'appuie sur les deux fonctions suivantes :

- une **fonction de commande** qui assure l'exécution des activités,
- une **fonction d'évaluation** qui mesure le résultat des activités en comparant une situation obtenue à une situation espérée.

Dans le cadre du modèle OLYMPIOS, le modèle de conduite définit une **activité conduite** par :

- une activité opérante qui transforme des produits,
- une activité de conduite qui assure les fonctions de commande et d'évaluation,
- un point de conduite qui assure la coordination entre l'activité conduite et son environnement.

L'**activité opérante** réalise la transformation pour obtenir un résultat satisfaisant l'objectif fixé. Elle est donc caractérisée par :

- un flot d'entrée et un flot de sortie,
- une opération,
- des ressources.

L'**activité de conduite** commande et évalue l'activité opérante. Elle est donc caractérisée par :

- un objectif,
- un résultat,
- un indicateur de performance.

Le **point de conduite** assure la coordination entre l'activité de conduite et l'activité opérante et entre l'activité conduite et son environnement (c'est-à-dire d'autres activités conduites). En d'autres termes, un point de conduite assure les fonctions suivantes :

- coordonner l'activité de conduite et l'activité opérante,
- gérer la coopération entre les activités conduites d'un processus,
- gérer la coopération entre le processus et son environnement.

Un point de conduite doit donc être capable de mettre en œuvre des mécanismes de communication et de synchronisation.

Le comportement de l'activité conduite est caractérisé par les états suivants :

- **assignée** : les ressources possibles pour la réalisation de l'activité opérante sont précisées,
- **configurée** : les ressources pour la réalisation de l'activité opérante sont retenues,
- **en cours** : l'activité opérante réalise la transformation,
- **bloquée** : la transformation liée à l'activité est interrompue de façon définitive avant sa terminaison normale,
- **suspendue** : la transformation liée à l'activité est interrompue de façon temporaire,

- **terminée** : la transformation liée à l'activité est achevée,
- **évaluée** : la performance de l'activité est évaluée.

La figure suivante présente le graphe d'états d'une activité conduite.

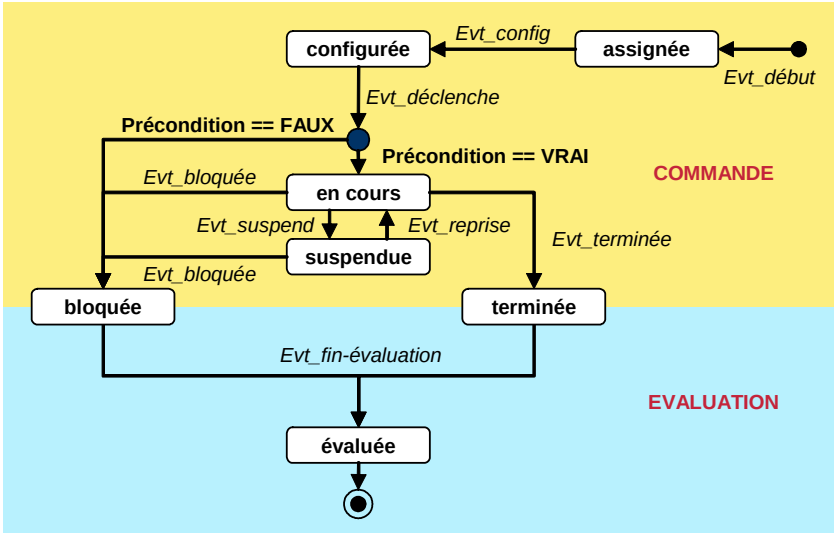



*Figure III.6. : Les différents états de l'activité conduite*

### 3.3.2. Le modèle de pilotage d'un processus

Ce modèle spécialise les quatre phases définies dans le modèle de pilotage d'un système. Le pilotage d'un processus nécessite de préciser ces états.

La **phase d'initialisation** du modèle de pilotage va construire le référent à partir des éléments suivants :

- un but à atteindre,
- des schémas de contrôle (gammes génériques) qui traduisent la connaissance des séquences de transformation à effectuer pour atteindre l'objectif fixé.

Dans le cadre du pilotage d'un processus, cette phase définit le graphe d'enchaînement d'ordres. Suite à cette définition, le processus est **existant** et toutes ses activités sont **assignées**. Dans un second temps, cette phase configure l'ensemble des activités du processus, le processus est alors dans un **état initial**.

Le modèle de conduite formalise la **phase d'évolution** du processus de son état initial vers son état final, via des états intermédiaires. L'état **intermédiaire** d'un processus signifie qu'au moins une activité du processus est dans l'état **en cours**, **bloquée**, **suspendue** ou **terminée**. La phase d'évolution est terminée lorsque toutes les activités du processus sont évaluées. Le processus est alors dans l'état **final**.

La **phase de correction** est déclenchée lorsque le modèle de conduite (fonction d'évaluation) détecte une non-conformité. Dans ce cas, le processus piloté passe dans l'état **adaptation** et le déroulement du processus piloté est interrompu. Cette phase recherche une solution corrective pour atteindre au mieux l'objectif fixé et aboutit à la définition d'un nouveau référent (graphe d'enchaînement d'ordres). Suite à cette phase de correction, le processus passe dans l'état **initial**.

La **phase de terminaison** évalue le processus en comparant le but (résultat obtenu) par rapport au besoin (objectif). Cette phase est déclenchée lorsque le processus est dans l'état final. Suite à cette activité, le processus passe alors dans l'état **terminé**.

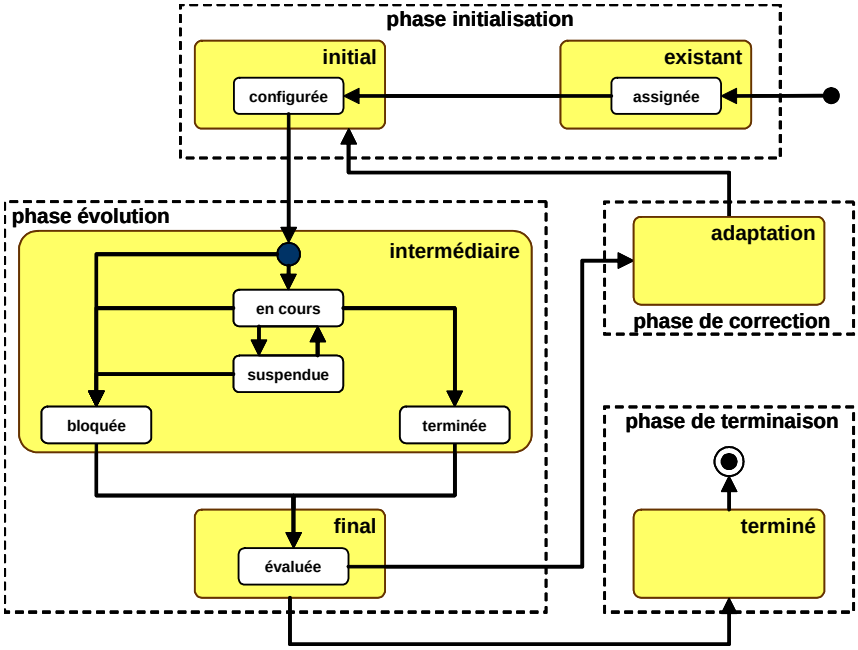



*Figure III.7. : Les quatre phases du modèle de processus*

Ayant présenté d'une part, les concepts relatifs à la formalisation d'une architecture à travers une vision « composant / connecteur » (Cf. § 2) et, d'autre part, les concepts de pilotage de processus liés au modèle OLYMPIOS, nous allons maintenant montrer comment nous formalisons les architectures industrielles et logicielles. En effet, l'utilisation du concept d'architecture pour formaliser le système d'information permet d'appréhender la réalité opérationnelle à travers différents niveaux d'abstraction, ce qui décomplexifie notre problématique. D'autre part, une vision architecturale fournit une représentation structurelle et comportementale des différents éléments composant le système d'information. Cette vision nous semble donc pertinente pour traiter de l'intégration de ces éléments. En conséquence, le modèle de pilotage a pour mission de faire évoluer les architectures. Rappelons que nous envisageons deux types de traitement pour corriger une non-conformité (Cf. § 3.1). La mise en œuvre de ces deux traitements nécessite la possibilité de construire des architectures évolutives. Le paragraphe suivant présente donc une description d'architecture industrielle et logicielle capable d'appréhender les concepts définis dans le modèle OLYMPIOS.

# 4. Formalisation des architectures

Nous allons maintenant nous appuyer sur le modèle d'architecture « composant / connecteur » et le modèle de pilotage pour formaliser les architectures industrielles et logicielles qui nous permettrons de représenter les points de vue « métier » et « donnée ». Ce paragraphe présente également les différentes possibilités d'évolution des architectures industrielles et logicielles.

## 4.1. Les architectures industrielles

La définition des architectures industrielles nécessite la définition des composants et des connecteurs spécifiques à cette architecture. Dans un premier temps, nous définissons les

éléments d'une architecture industrielle et nous présentons ensuite différentes architectures qui sont à la base de nos travaux dans le cadre du projet ArchWare.

### 4.1.1. Les composants d'une architecture industrielle

Les différents composants d'une architecture industrielle sont :

- le composant activité,
- le composant chorégraphe,
- le composant industriel,
- le composant système de performance,
- le composant système de gestion de ressources.

Le **composant activité** et le **composant chorégraphe** permettent de formaliser le concept de processus définit par le modèle OLYMPIOS (Cf. § 3.2).

Le **composant activité** est une entité de structuration définie pour satisfaire un objectif. Son comportement lui permet de fournir un résultat. Il encapsule une activité conduite. De par sa nature, ce composant dispose des propriétés suivantes, il est :

- atomique,
- connectable à d'autres composants activité, du fait qu'il dispose de ports lui permettant de communiquer.

Ce composant doit disposer des ports suivants :

- un port capable d'échanger des flots,
- un port capable de recevoir un objectif et d'émettre un résultat,
- un port capable d'échanger des données caractérisant des ressources,
- un port capable d'échanger des événements ou des messages pour se synchroniser avec son environnement.

Ce composant peut également être appréhendé comme un composite pour définir les différentes caractéristiques d'une activité (Cf. § 3.2.1). Dans le cadre de ces travaux, nous ne détaillerons pas cette architecture[20].

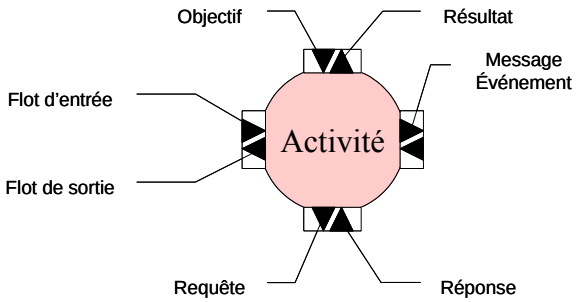



*Figure III.8. : Le composant activité*

Le **composant chorégraphe** orchestre l'exécution des activités conduites composant un processus. En d'autres termes, ce composant supporte les phases d'initialisation, de correction et de terminaison du modèle de pilotage (Cf. § 3.3). Ce composant s'appuie sur le graphe d'ordres pour réaliser cette coordination. Ce composant doit disposer d'un port spécifique pour s'interfacer avec chaque composant activité appartenant à un processus. Ce port doit être

[20] La description détaillée de ce composite est fournie dans [Cha 02].

capable de transmettre les messages et les événements nécessaires à la coordination des activités, il doit pouvoir être créé de façon dynamique. Ce composant peut également être appréhendé comme un composite mais, dans le cadre de ces travaux, nous ne détaillerons pas cette architecture[21].

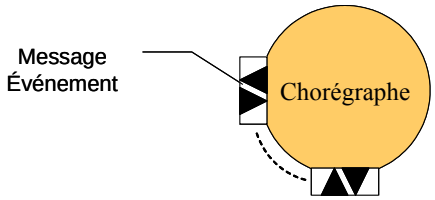



*Figure III.9. : Le composant chorégraphe*

Ayant défini les composants caractérisant un processus, nous allons maintenant nous intéresser à la description de la réalité opérationnelle d'une entreprise. En d'autres termes, nous allons préciser les composants nécessaires à la coordination des processus de réalisation d'une entreprise (Cf. Chapitre 2, § 2.3.2) : le composant industriel, le composant système de performance et le composant système de gestion de ressource.

Le **composant industriel** (CI) [BTH 00] [Cha 02] est une entité de structuration définie pour satisfaire un besoin unique (Cf. Chapitre 1, § 4.3.4). Il a un comportement qui lui permet d'atteindre son but qui vise à satisfaire un besoin. Il encapsule donc un processus piloté, ce qui induit que le composant industriel est un composant dynamique. Il est le constituant de base d'une architecture industrielle, il est créé lorsqu'un nouveau besoin est défini et, est supprimé suite à la réalisation du but. De par sa nature, un composant industriel dispose des propriétés suivantes, il est :

- générique, ce qui lui permet de représenter tout ou partie de l'entreprise en tenant compte des aspects structuraux, fonctionnels et comportementaux d'un secteur donné,
- connectable à d'autres composants industriels, du fait qu'il dispose de ports lui permettant de communiquer.

Ce composant doit disposer de trois ports pour s'interfacer avec son environnement :

- un port capable d'échanger des flots,
- un port capable de recevoir un besoin (objectif) et d'émettre un but (résultat),
- un port capable d'échanger des données caractérisant des ressources.

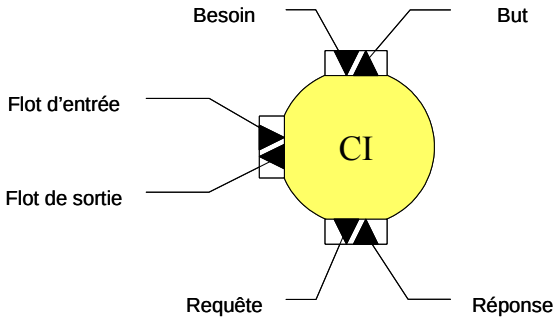



*Figure III.10. : Le composant industriel*

Nous montrerons dans le paragraphe 4.1.3 que ce composant est un composite.

[21] La description détaillée de ce composite est fournie dans [Cha 02].

Le **composant système de performance** (SP) définit les besoins des différents composants industriels et évalue les buts atteints par ces composants. En d'autres termes, le système de performance gère les objectifs de l'entreprise et évalue les résultats obtenus lors du déroulement des différents processus. Aussi, dans le cadre de nos travaux, nous considérons que le système de gestion de connaissance (Cf. Chapitre 1, § 2.2.3.3.c) est limité au fonctionnement précisé par ce composant. Ce composant doit disposer d'un port spécifique pour s'interfacer avec chaque composant industriel défini à un niveau d'abstraction donné. Ce port doit être capable de transmettre un besoin et de recevoir un but, il doit pouvoir être créé de façon dynamique. Ce composant peut également être appréhendé comme un composite mais, dans le cadre de ces travaux, nous ne détaillerons pas cette architecture[22].

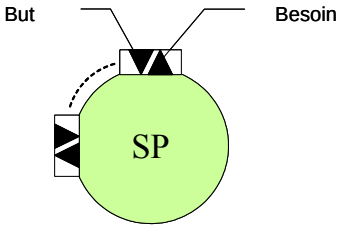



*Figure III.11. : Le composant système de performance*

Le **composant système de gestion de ressources** (SGR) gère l'ensemble des données caractérisant les ressources qui supportent les activités des processus. En particulier, ce composant gère leur état (disponible, occupé, hors service et hors d'usage [Thé 02]). Ce composant doit disposer d'un port spécifique pour s'interfacer avec chaque composant industriel défini à un niveau d'abstraction donné. Ce port doit être capable de recevoir une requête et de transmettre une réponse, il doit pouvoir être créé de façon dynamique. Ce composant peut également être appréhendé comme un composite mais, dans le cadre de ces travaux, nous ne détaillerons pas cette architecture[22].

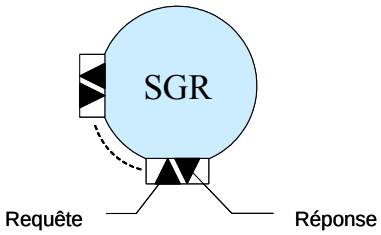



*Figure III.12. : Le composant système de gestion de ressources*

Nous montrons dans le paragraphe 4.1.3 des exemples d'architectures construites à partir de ces composants.

### 4.1.2. Les connecteurs dans une architecture industrielle

Le connecteur définit les différents mécanismes d'assemblage entre les composants industriels [BBD 03b]. Le premier mécanisme d'assemblage est l'assemblage par ports (le besoin ou but). Il lie les différents niveaux de décision d'un domaine donné en fournissant :

- un « *assemblage par le besoin* » : le besoin d'un composant industriel peut être décomposé en plusieurs sous-besoins. Ce sont les besoins des composants industriels d'un autre niveau. L'opérateur qui lie les besoins dépend de la décomposition du besoin initial,

[22] La description détaillée de ce composite est fournie dans [Cha 02].

- un « *assemblage par le but* » : des buts de plusieurs composants industriels, définis au même niveau, peuvent être agrégés pour constituer le but d'un composant industriel d'un autre niveau ; cet assemblage est réalisé par des opérateurs spécifiques (agrégation, par exemple).

Le deuxième mécanisme d'assemblage caractérise une relation d'échange entre deux composants industriels. Rappelons que le modèle OLYMPIOS ne perçoit pas l'activité comme une simple fonction de transformation mais il la considère comme une unité de réalisation liée à l'usage d'une ressource (ressource appropriée). Cette notion est formalisée par le concept de relation d'échange (Cf. Chapitre 1, § 4.3.1.2). Ce mécanisme fait donc émerger la notion de *rôle* qui permet de caractériser les composants industriels assemblés. Le composant industriel dont le but est utilisé par l'autre composant de la relation, a un rôle de ***fournisseur***, tandis que le composant industriel qui s'approprie le but a un rôle d'***utilisateur***.

Enfin, il existe des mécanismes d'assemblage de composants industriels par les ressources, qui supportent :

- la coopération entre les composants industriels (communication et synchronisation),
- la délégation de tout ou partie d'un comportement d'un composant industriel vers un autre composant.

Ces différents mécanismes d'assemblage permettent de construire une architecture représentant un secteur donné.

### 4.1.3. Les configurations d'une architecture industrielle

La configuration d'une architecture industrielle précise les composants, les connecteurs et leur organisation. Dans le cadre de ce paragraphe, nous proposons trois configurations [BBD 03a] [BBD 03c] :

- l'architecture d'un composant industriel,
- une architecture intra-entreprise,
- une architecture inter-entreprise.

Pour chaque architecture présentée, nous décrivons des possibilités d'évolution.

#### *4.1.3.1 Architecture d'un composant industriel*

Cette architecture s'appuie sur le modèle de processus (Cf § 3.2) pour construire l'architecture du composant industriel (composite). Cette architecture s'appuie sur les composants suivants :

- un ou plusieurs composants activité,
- **un et un seul** composant chorégraphe.

Les connecteurs entre les composants activités sont des connecteurs supportant des mécanismes mettant en œuvre une relation d'échange. Les connecteurs entre les composants activités et le chorégraphe sont des connecteurs supportant des mécanismes de coopération.

Au sein de cette architecture, les propriétés suivantes doivent être respectées :

- le composant chorégraphe doit disposer d'un port pour chaque composant activité présent au sein de l'architecture,
- les connecteurs entre composants activité sont toujours des connecteurs « relation d'échange »,

- un connecteur « relation d'échange » établit une relation entre une ou plusieurs activités « amont » et une ou plusieurs activités « aval », seuls les ports « flots » d'un composant activité peuvent être connectés à ces connecteurs,
- les ports « message événements » des composants activités sont connectés aux ports du chorégraphe via un connecteur « coopération »,
- les ports « objectif / résultat » et « requête / réponse » des composants activités sont les ports du composite.

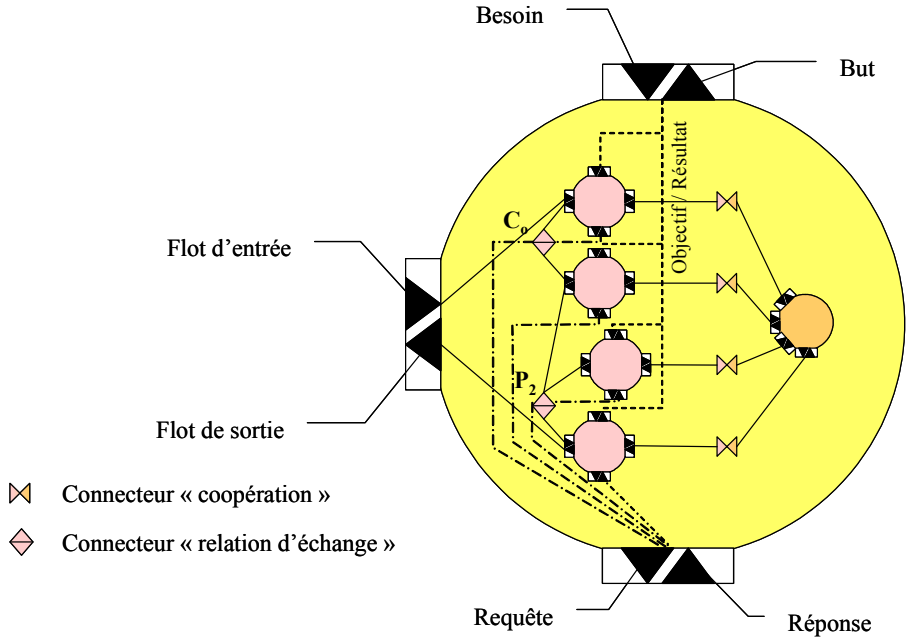



*Figure III.13. : Exemple d'architecture du composant industriel*

La phase d'évolution supportée par le chorégraphe peut modifier le référent, ce qui implique que les composants et les connecteurs formalisant un composant industriel sont des éléments dynamiques. En effet, la phase de correction peut être amenée à créer ou supprimer des activités, ce qui induit la possibilité de :

- créer ou supprimer des composants activité,
- créer ou supprimer des connecteurs « relation d'échange »,
- créer ou supprimer des ports sur les connecteurs « relation d'échange »,
- créer ou supprimer des ports sur le chorégraphe,
- créer ou supprimer des connecteurs « coopération »,
- créer ou supprimer des connexions entre les composants et les connecteurs.

#### *4.1.3.2 Architecture intra-entreprise*

Cette architecture formalise la coordination de plusieurs processus de réalisation (Cf. Chapitre 2, § 2.3.2) au sein d'une entreprise. Elle est composée de :

- un ou plusieurs composants industriels,
- **un et un seul** composant système de performance,
- **un et un seul** composant système de gestion de ressources.

Les connecteurs entre les composants industriels sont des connecteurs supportant des mécanismes mettant en œuvre une relation d'échange. Les connecteurs entre les composants industriels et le composant système de performance et entre les composants industriels et le

composant système de ressources sont des connecteurs supportant des mécanismes de coopération.

Au sein de cette architecture, les propriétés suivantes doivent être respectées :

- les composants système de performance et système de gestion de ressources doivent disposer d'un port pour chaque composant industriel présent au sein de l'architecture,
- les connecteurs entre les composants industriels sont toujours des connecteurs « relation d'échange »,
- un connecteur « relation d'échange » établit une relation entre un ou plusieurs processus « amont » et un ou plusieurs processus « aval », seuls les ports « flots » d'un composant industriel peuvent être connectés à ces connecteurs,
- les ports « objectif / résultat » des composants industriels sont connectés aux ports du composant système de performance via un connecteur « coopération »,
- les ports « requête / réponse » des composants industriels sont connectés aux ports du composant système de gestion de ressources via un connecteur « coopération »,
- les ports « relation d'échange » des composants industriels qui ne sont pas connectés à un connecteur « relation d'échange » sont les ports du composite.

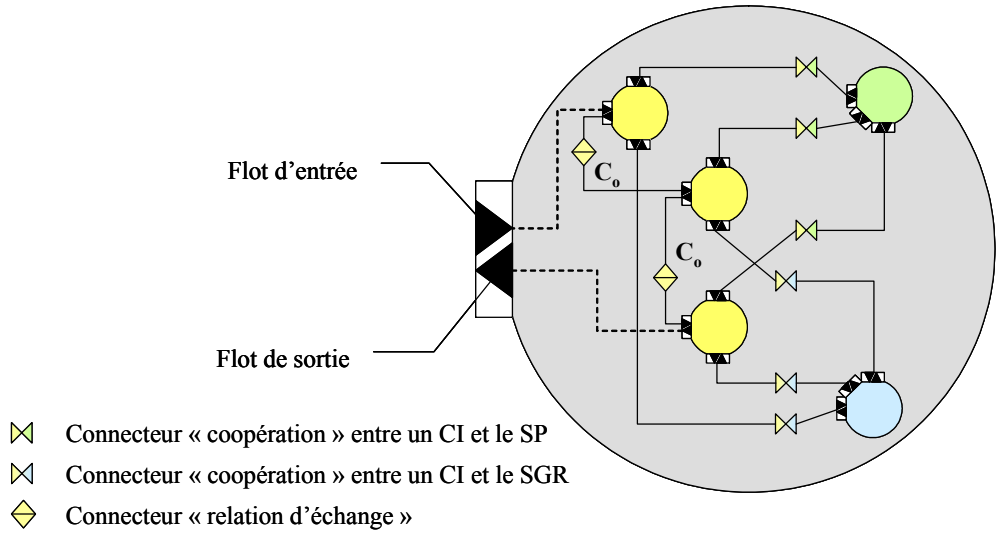



*Figure III.14. : Exemple d'architecture intra-entreprise*

Le composant système de gestion de performance peut modifier cette architecture en fonction de l'évolution des besoins. Ce constat implique que les composants et les connecteurs formalisant cette architecture sont des éléments dynamiques. En effet, le composant système de gestion de performance peut être amené à supprimer ou créer des processus, ce qui induit la possibilité de :

- créer ou supprimer des composants industriels,
- créer ou supprimer des connecteurs « relation d'échange »,
- créer ou supprimer des ports sur les connecteurs « relation d'échange »,
- créer ou supprimer des ports sur le composant système de gestion de performance et le composant système de gestion de ressources,
- créer ou supprimer des connecteurs « coopération »,
- créer ou supprimer des connexions entre les composants et les connecteurs.

Les architectures intra-entreprise permettent de représenter la vision détaillée du point de vue « métier ». Un exemple concret de cette architecture est fourni dans le chapitre 5, paragraphe 4.3.1.2.

#### *4.1.3.3 Architecture inter-entreprise*

Cette architecture formalise la coordination de plusieurs entreprises, elle permet donc de représenter la vision globale du point de vue « métier ». Elle est composée d'un ou plusieurs composants industriels. Chaque composant industriel représente le fonctionnement de l'entreprise par un processus global.

Rappelons que dans le cadre de nos travaux, nous nous focalisons sur les processus de réalisation pour caractériser la réalité opérationnelle d'une entreprise (Cf. Chapitre 2, § 2.3.2). De ce fait, nous ne considérons que la gestion des flux de production et nous limitons l'environnement de l'entreprise à ses clients et à ses fournisseurs. C'est pourquoi, seuls les connecteurs mettant en œuvre cette relation d'échange sont appréhendés dans le cadre de cette étude.

Au sein de cette architecture, les propriétés suivantes doivent être respectées :

- les connecteurs entre les composants industriels (entreprise) sont toujours des connecteurs « relation d'échange »,
- un connecteur « relation d'échange » établit une relation entre une ou plusieurs entreprise « amont » et une ou plusieurs entreprise « aval », seuls les ports « flots » d'un composant industriel peuvent être connectés à ces connecteurs.

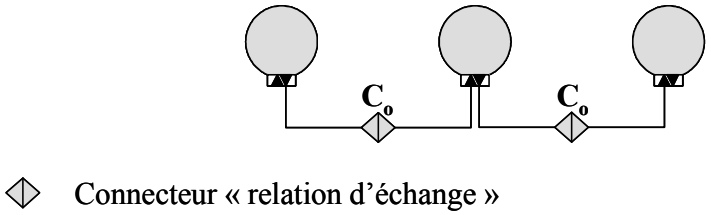



*Figure III.15. : Exemple d'architecture inter-entreprise*

La figure précédente formalise le concept de chaîne logistique entre plusieurs entreprises. Dans le cadre de nos travaux, nous considérons que les évolutions de cette architecture sont limitées. En effet, les entreprises représentées au sein de cette architecture sont identifiées. Il n'est donc pas possible de créer ou de supprimer des composants industriels représentant une entreprise. Par contre, les relations d'échange établies entre ces entreprises peuvent être modifiées. En effet, si une relation d'échange n'aboutit pas (niveau de satisfaction insuffisant) (Cf. Chapitre 1, § 4.3.1.2 et § 4.3.2.1), il doit être possible d'établir de nouvelles relations d'échange. Pour ce faire, nous considérons que chaque entreprise connaît la liste de l'ensemble de ses fournisseurs potentiels. Par conséquent, l'évolution de l'architecture inter-entreprise induit la possibilité de :

- créer ou supprimer des connecteurs « relation d'échange »,
- créer ou supprimer des ports sur les connecteurs « relation d'échange »,
- créer ou supprimer des ports sur les composants industriels,
- créer ou supprimer des connexions entre les composants et les connecteurs.

Un exemple concret de cette architecture est fourni dans le chapitre 5, paragraphe 4.3.1.1.

## 4.2. Les architectures logicielles

Nous nous appuyons sur la même logique pour formaliser les architectures logicielles.

La définition des architectures logicielle nécessite la définition des composants et des connecteurs spécifiques à cette architecture. Dans un premier temps, nous définissons les éléments d'une architecture logicielle et nous présentons ensuite différentes architectures qui sont à la base de nos travaux dans le cadre du projet ArchWare.

### 4.2.1. Les composants d'une architecture logicielle

Les différents composants d'une architecture logicielle sont :

- le composant application,
- le composant moteur de workflow.

Le **composant application** encapsule une application logicielle capable de réaliser un ensemble de fonctions définies. Ces fonctions sont caractérisées par leurs entrées et leurs résultats. Ces éléments constituent le port du composant application.

De par sa nature, un composant application dispose des propriétés suivantes :

- atomique, ce qui induit que les fonctions ne peuvent être perçues qu'à travers leurs entrées ou leurs sorties,
- générique, ce qui lui permet de représenter tout ou partie du système d'information (informatisé) de l'entreprise,
- connectable à d'autres composants application, du fait qu'il dispose de ports lui permettant de communiquer.

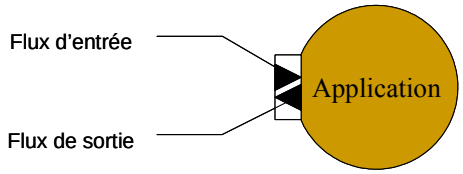



*Figure III.16. : Le composant application*

Le **composant workflow** transforme et interprète les données échangées entre les différents composants application. Ce composant doit donc structurer ces différentes données. De plus, ce composant aiguille les différents flux d'information vers les différents composants application (Cf. Chapitre 2, § 3.2.3.2 et § 3.2.3.3). Pour permettre l'évolution d'une architecture logicielle, ce composant doit être capable de mettre en œuvre les quatre phases définies au sein du modèle OLYMPIOS (Cf. § 3.3). Ce composant doit disposer d'un port spécifique pour s'interfacer avec chaque composant application. Ce port doit être capable de recevoir et de transmettre les différents flux d'information et il doit pouvoir être créé de façon dynamique.

Le nombre de ports du composant workflow dépend du nombre de composants application à interfacer.

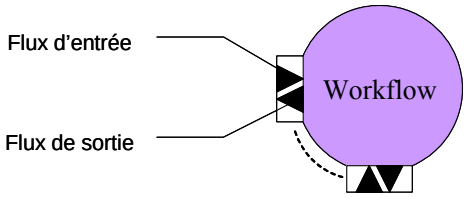



*Figure III.17. : Le composant workflow*

Nous montrons dans le paragraphe suivant (Cf. § 5.2), que le fonctionnement du composant workflow est défini par le composant SGR. En effet, ce composant est chargé de fournir les différentes ressources (données) nécessaires au déroulement des processus (composant CI).

### 4.2.2. Les connecteurs d'une architecture logicielle

Dans le cadre de nos travaux, nous ne considérons que les connecteurs supportant les mécanismes de communication. Ces connecteurs vérifient les propriétés définies lors de la présentation des différentes couches d'une architecture EAI (Cf. Chapitre 2, § 3.2.3.1).

### 4.2.3. Les configurations d'une architecture logicielle

La configuration d'une architecture logicielle précise les composants, les connecteurs et leur organisation. Dans le cadre de ce paragraphe, nous identifions deux configurations [BBD 03a] [BBD 03c] :

- une architecture EAI,
- une architecture de systèmes d'information distribués.

Pour chaque architecture présentée, nous décrivons des possibilités d'évolution.

#### *4.2.3.1 Architecture EAI*

Cette architecture formalise les couches d'une architecture EAI introduite précédemment (Cf. Chapitre 2, § 3.2.3). Elle est composée de :

- un ou plusieurs composants application,
- **un et un seul** composant workflow.

Les connecteurs entre les composants application et le composant workflow sont des connecteurs supportant des mécanismes de communication.

Au sein de cette architecture, les propriétés suivantes doivent être respectées :

- le composant workflow doit disposer d'un port pour chaque composant application présent au sein de l'architecture,
- il n'existe pas de connecteur entre deux composants application,
- les ports « flux » du composant application sont connectés aux ports « flux » du composant workflow via un connecteur « communication »,
- le composant workflow doit disposer d'au moins un port « flux » spécifique qui est le port du composite.

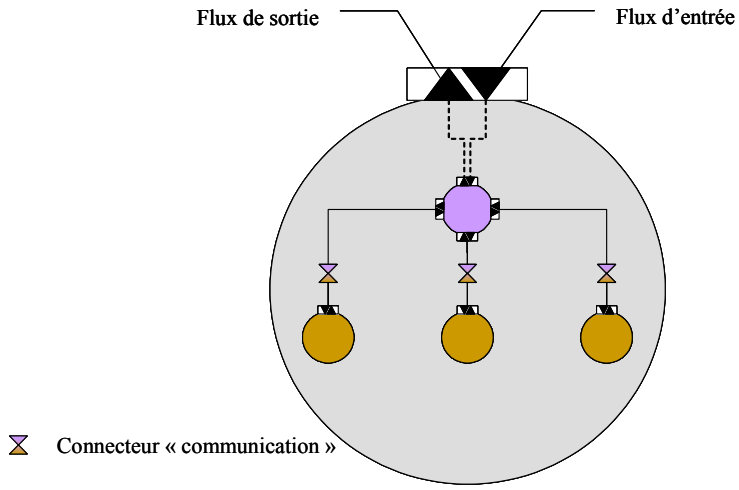



*Figure III.18. : Exemple d'architecture EAI*

Le composant workflow peut modifier cette architecture en fonction des dysfonctionnements identifiés lors de l'exécution des applications. Ce constat implique que les composants et les connecteurs formalisant cette architecture sont des éléments dynamiques. En effet, le composant workflow peut être amené à créer ou supprimer des applications, ce qui induit la possibilité de :

- créer ou supprimer des composants application,
- créer ou supprimer des ports sur le composant workflow,
- créer ou supprimer des connecteurs « communication »,
- créer ou supprimer des connexions entre les composants et les connecteurs.

Les architectures EAI permettent de représenter la vision détaillée du point de vue « donnée ». Un exemple concret de cette architecture est fourni dans le chapitre 5, paragraphe 4.3.1.3.

#### *4.2.3.2 Architecture de systèmes d'information distribués*

Cette architecture formalise la coordination de plusieurs systèmes d'information. Dans le cadre de nos travaux, ces systèmes d'information sont identifiés comme un ensemble d'applications logicielles coordonnées. Elle permet donc de représenter la vision globale du point de vue « donnée ». Elle est composée d'un ou plusieurs composants application. Chaque composant application représente le fonctionnement d'un système d'information d'une entreprise.

Les connecteurs entre les composants application sont des connecteurs supportant des mécanismes de communication.

Au sein de cette architecture, les propriétés suivantes doivent être respectées :

- les connecteurs entre les composants application (système d'information distribué) sont toujours des connecteurs « communication »,
- les ports « flux » d'un composant application sont connectés via un connecteur « communication ».

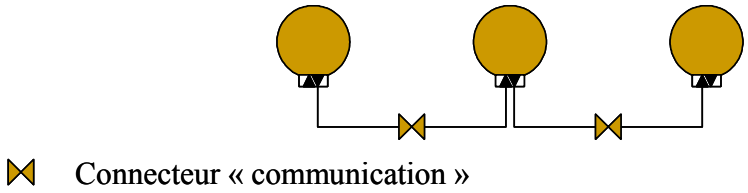



*Figure III.19. : Exemple d'architecture de systèmes d'information distribués*

Comme pour les architectures inter-entreprise, nous considérons que les évolutions de cette architecture sont limitées. Les entreprises représentées au sein d'une architecture inter-entreprise étant identifiées, les systèmes d'information le sont également. Il n'est donc pas possible de créer ou de supprimer des composants application représentant un système d'information. Par contre, les communications établies entre ces systèmes d'information peuvent être modifiées. Par conséquent, l'évolution de l'architecture de systèmes d'information distribués induit la possibilité de :

- créer ou supprimer des connecteurs « communication »,
- créer ou supprimer des ports sur les composants application,
- créer ou supprimer des connexions entre les composants et les connecteurs.

## 5. Interactions entre les différentes architectures

Le paragraphe précédent nous a permis de décrire les différentes architectures utilisées pour représenter les différentes visions des points de vue « métier » et « donnée ». D'après le paradigme d'ambivalence (Cf. Chapitre 1, § 3), il est nécessaire d'établir des correspondances entre les différentes visions (globale et détaillée) de ces deux points de vue. Nous allons donc, dans le cadre de ce paragraphe, présenter ces différentes interactions. Nous nous appuyons sur une spécialisation du schéma présenté lors de la description du paradigme d'ambivalence (Cf. Chapitre 1, § 3.1.2) pour illustrer les interactions entre les différentes architectures.

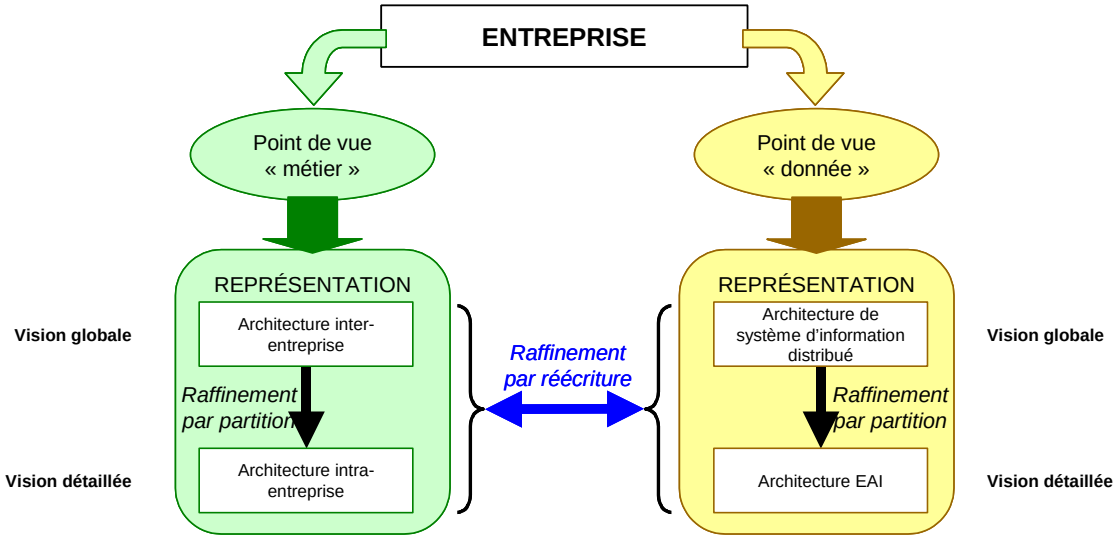



*Figure III.20. : Interactions entre les architectures*

Il est à noter que, dans le cadre de nos travaux, nous ne nous sommes pas intéressés aux interactions existant entre l'architecture inter-entreprise et l'architecture de système d'information distribué car notre application vise à rendre une entreprise agile au sein de son environnement.

## 5.1. Les interactions entre la vision globale et la vision détaillée

Suite à la remarque précédente, nous nous intéressons uniquement à la correspondance entre l'architecture inter-entreprise et l'architecture intra-entreprise. L'étude de l'agilité d'une entreprise nécessite de la représenter dans son environnement. Aussi, la vision globale du point de vue « métier » présente les interactions entre l'entreprise et les autres entreprises composant son environnement. Rappelons que nous nous focalisons sur la gestion du flux de production, nous avons donc limité l'environnement de l'entreprise à ses clients et à ses fournisseurs. C'est pourquoi, lors de la définition de l'architecture inter-entreprise, seul le connecteur relation d'échange est utilisé.

La vision détaillée du point de vue « métier » représente, quant à elle, le fonctionnement interne d'une entreprise. L'interaction existant entre ces deux points de vue est réalisée par un mécanisme de raffinement par partition (raffinement horizontal) [Bol 04].

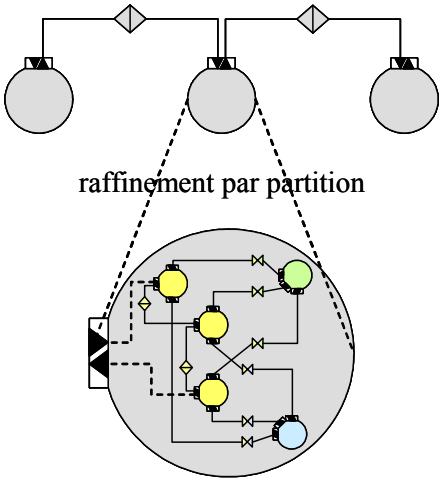



*Figure III.21. : Raffinement par partition*

## 5.2. Les interactions entre le point de vue « métier » et le point de vue « donnée »

Ce paragraphe présente les interactions entre le point de vue « métier » et le point de vue « donnée ». Dans le cadre de notre approche, nous considérons que chaque activité d'un processus est mise en œuvre par une fonction spécifique d'une application logicielle de l'entreprise.

L'exécution d'une activité correspond donc d'une part, à l'invocation d'une fonction d'une application et, d'autre part, à la récupération des résultats de la fonction pour calculer la performance de l'activité. Rappelons que le système de gestion de ressources (SGR) (Cf. Chapitre 1, § 4.3.2.3) est chargé du management de l'ensemble des ressources (logiques ou matérielles) nécessaires à la réalisation des activités. Par conséquent, l'invocation d'une fonction, la fourniture des données nécessaires à son exécution et la restitution du résultat sont réalisées par le SGR.

La fonction de transition contrôle, quant à elle, l'enchaînement des activités. Cette fonction précise donc, au SGR, l'ordre dans lequel les fonctions doivent être exécutées. En d'autres termes, cette fonction définit les échanges qui doivent être réalisées entre les différentes applications supportant les activités des processus [BBD 03d].

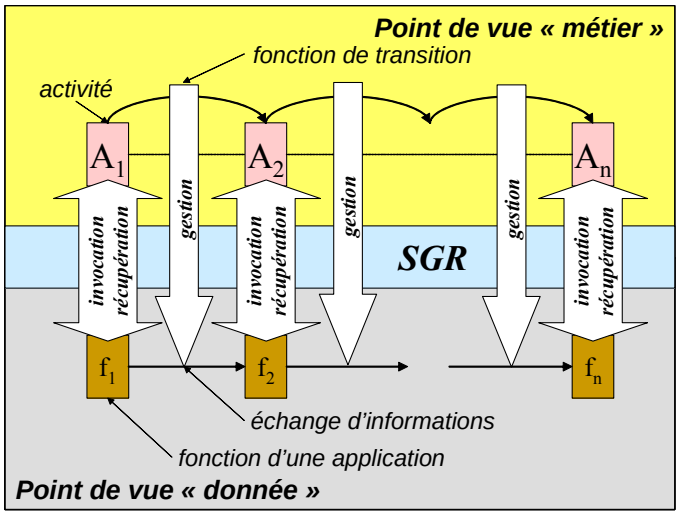



*Figure III.22. : Interactions entre activité et fonction*

Ayant précisé les interactions entre une activité d'un processus et une fonction d'une application logicielle, nous pouvons caractériser l'interaction entre l'architecture intra-entreprise et l'architecture EAI. Le mécanisme de raffinement par réécriture (raffinement vertical) [Bol 04] doit donc assurer une correspondance entre le fonctionnement du SGR et la coordination de l'ensemble des composants application coordonnés par le composant workflow (architecture EAI). Ce mécanisme précise donc l'invocation des fonctions supportant les activités d'un processus et la restitution de leur résultat. Il doit également assurer une correspondance entre l'enchaînement des activités (défini par la fonction de transition) et la coordination des applications logicielles. Rappelons que cette coordination est assurée par le composant workflow. Ce mécanisme peut être représenté par le schéma suivant :

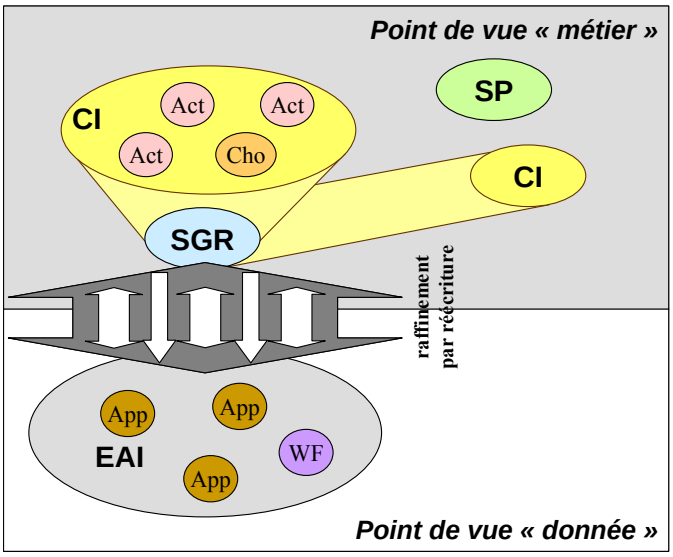



*Figure III.23. : Raffinement par réécriture*

# 6. Conclusion

Dans le cadre de ce chapitre, nous avons présenté différents concepts sur lesquels nous nous sommes appuyés pour formaliser les architectures industrielles et logicielles. En effet, ces architectures s'appuient sur le modèle « composant / connecteur » qui est un modèle

générique d'architecture. De plus, certains composants de ces architectures s'appuient sur un modèle de pilotage pour les faire évoluer.

Nous avons décrit les architectures permettant de représenter les visions globale et détaillée des points de vue « métier » et « donnée ». Toutefois, rappelons que nous ne nous intéresserons pas à l'architecture globale du point de vue « donnée » car nous nous focalisons sur l'agilité d'une entreprise dans son environnement. Cette vision pourra servir de base à des travaux ultérieurs portant sur l'étude des systèmes d'information distribués dans le cadre des réseaux d'entreprises.

Il est à noter que le modèle de pilotage est uniquement intégré dans l'architecture intra-entreprise (composant chorégraphe d'un composant industriel) et dans l'architecture EAI (composant workflow de l'architecture EAI). Ce modèle de pilotage doit permettre de faire évoluer ces architectures suivant des logiques déterminées. L'architecture inter-entreprise est également susceptible d'évoluer mais, dans ce cas, il est nécessaire de fournir à un acteur humain une interface spécifique pour modifier l'architecture. Dans le cadre du projet ArchWare, un outil spécifique (*Tower Browser*) dispose de ces fonctionnalités.

Enfin, nous avons caractérisé les mécanismes de raffinement permettant d'établir les correspondances entre ces différentes architectures. Nos travaux se déroulant dans le cadre d'un projet visant à développer un ensemble intégré de langages et d'outils centrés architecture pour l'ingénierie des systèmes logiciels évolutifs, nous nous appuierons sur les propositions réalisées dans le cadre de ce chapitre pour préciser d'une part les spécificités que doit respecter le langage de description d'architecture et, d'autre part, les spécificités que doit respecter le langage de raffinement proposé par la plate-forme du projet ArchWare.

# Chapitre 4 :
# Définition et utilisation de styles architecturaux

# Chapitre 4 :
# Définition et utilisation de styles architecturaux

## 1. Introduction

Ce chapitre présente les principaux concepts et mécanismes d'un langage de description de styles architecturaux, l'ASL – *Architectural Style Language*. Cette formalisation fournit deux avantages lors d'un processus de développement de logiciels centré sur les styles. Le premier avantage est de proposer une manière uniforme de décrire des styles. En effet, tous les acteurs peuvent interpréter une description de la même manière si des notations sont clairement définies et si elles sont identiques pour toutes les descriptions[23]. Le deuxième avantage est de pouvoir exploiter ces styles à l'aide d'applications logicielles. Cette approche favorise la définition d'un langage « commun » pour formaliser la vision globale et la vision détaillée des points de vue « métier » et « donnée ». Rappelons que cette notion de langage commun est un point important du paradigme d'ambivalence (Cf. Chapitre 1, § 3.1.2).

Un style caractérise, à travers différentes facettes, les architectures propres à un domaine particulier. Une des facettes concerne la définition des propriétés attendues pour une architecture. La vérification de ces propriétés induit l'utilisation d'un langage formel pour leur description. De plus, ce formalisme doit partager les mêmes concepts architecturaux que le formalisme de description d'architecture.

Dans un premier temps, ce chapitre rappelle la notion de style architectural et présente ses différentes facettes. Ensuite, nous présentons le langage utilisé pour formaliser les styles architecturaux. Enfin, nous illustrons cette description par un exemple issu de nos travaux.

## 2. La notion de style architectural

Dans le chapitre précédent, nous avons montré que nos travaux nécessitent la définition des différentes configurations architecturales. Aussi, nous nous appuyons sur la notion de style pour caractériser ces configurations. Dans un premier temps, nous présentons un langage de style architectural caractérisant les architectures propres à un domaine à travers trois facettes.

La première facette du style, désignée ***facette « spécification »***, précise d'une part, les propriétés attendues pour l'architecture et, d'autre part, les contraintes (topologiques, comportementales et attributs) qui fixent les règles et les limites de construction de l'architecture (Cf. Chapitre 1, § 5). En d'autres termes, cette facette fournit le cahier des charges d'une architecture d'un domaine spécifique.

La deuxième facette, désignée ***facette « domaine »***, précise les éléments de base d'une architecture. Ces éléments définissent :

- l'ensemble des concepts et des mécanismes propres au domaine étudié,
- les éléments de construction spécifiques (composants et connecteurs),
- un vocabulaire et une syntaxe adaptés à la description des architectures du domaine.

[23] Les descriptions informelles utilisent souvent des notations propres à un métier [ShC 96] [ShC 97].

La troisième facette, désignée ***facette « patron »***, définit une configuration qui peut être utilisée pour appréhender une problématique donnée. En d'autres termes, cette facette propose des solutions architecturales à des problèmes récurrents et, c'est en ce sens, qu'elle peut être considérée comme un « patron » d'architecture qui précise la topologie et le comportement à respecter.

Un style est moins contraignant qu'une architecture car il n'aborde pas les caractéristiques propres à une architecture particulière. Par exemple, la facette décrivant les propriétés d'un style définit les propriétés communes à toutes les architectures du domaine étudié. Pour les mêmes raisons, la description d'un style est moins complète que celle d'une architecture particulière. Il est à noter, que suivant les cas, toutes les facettes d'un style ne sont pas nécessairement définies.

## 2.1. L'utilisation d'un style architectural dans une démarche centrée architecture

Avant de présenter le langage ASL, nous rappelons l'utilisation des styles dans une démarche centrée architecture (Cf. Chapitre 2, § 5.3.2).

Les différentes étapes d'une démarche centrée architecture exploitant le concept de style architectural sont :

- définition du style pour construire une bibliothèque,
- instanciation du style pour construire une architecture,
- raffinement de l'architecture jusqu'à l'obtention de l'application finale.

L'utilisation des styles dans ces différentes étapes permet l'***instanciation*** du style qui consiste à générer automatiquement une architecture à partir d'un style donné. Le schéma suivant rappelle le processus de développement centré style.

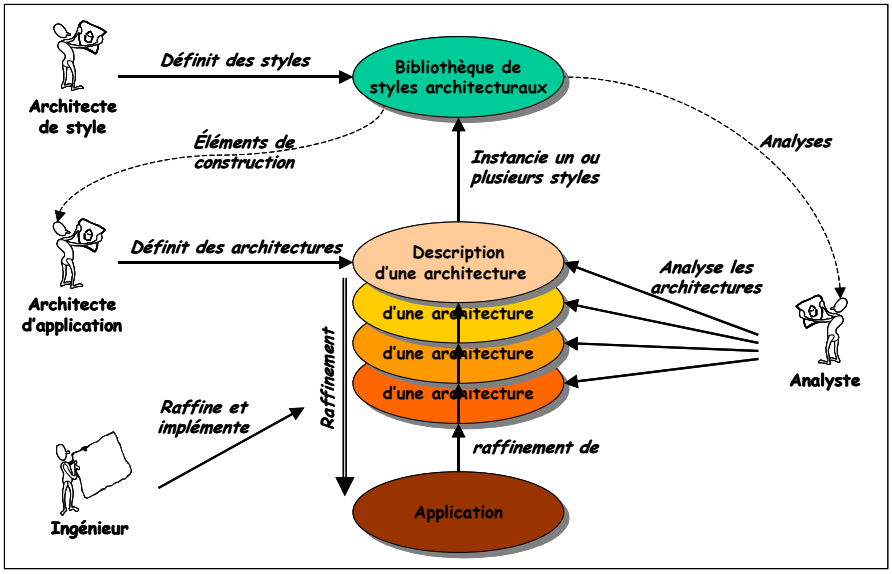



*Figure IV.1. : Processus de développement centré style*

Remarque : La notion d'architecture concrète est facilement interprétable dans le domaine du bâtiment. En effet, l'architecture concrète est le bâtiment réalisé, les architectures abstraites sont les différentes représentations réalisées avant la construction du bâtiment (esquisses, plans d'ensemble, plan d'exécution, etc.). Dans le cas du logiciel, cette notion est beaucoup

plus difficile à appréhender. En effet, le logiciel est immatériel, ce qui signifie qu'il n'est perceptible qu'à travers ses représentations (cahier des charge, diagrammes de conception, programmes en langages de haut niveau, programme exécutable, etc.). Aussi, dans le cadre de ces travaux, nous considérons que l'architecture concrète est l'application en cours d'exécution. En effet, cette perception nous semble la plus proche de la notion d'architecture concrète définie dans le domaine du bâtiment (produit utilisé).

Suite à ces remarques, nous pouvons constater qu'un style architectural offre :

- un cadre de conception à travers :
  - un support d'analyse,
  - un support de construction par la définition des éléments architecturaux du vocabulaire, des contraintes, etc. ou par le biais de « patrons » capables de traiter des problèmes récurrents,
- un cadre d'évolution.

Nous allons maintenant détailler l'utilisation du style dans ces deux cadres.

## 2.2. La notion de style dans le cadre de la conception d'une architecture

Les styles sont présents tout au long d'un cycle de développement centré architecture, depuis la spécification des besoins, jusqu'à la maintenance d'une architecture. Le schéma suivant montre que les facettes d'un style peuvent être utilisées pour :

- formaliser les spécifications d'une famille (ou d'une ligne) de produits à partir d'un cahier des charges,
- apporter un support à la description avec des notations spécifiques,
- apporter un support pour l'analyse de propriétés,
- apporter un support pour la conception de systèmes répondant à des besoins spécifiques.

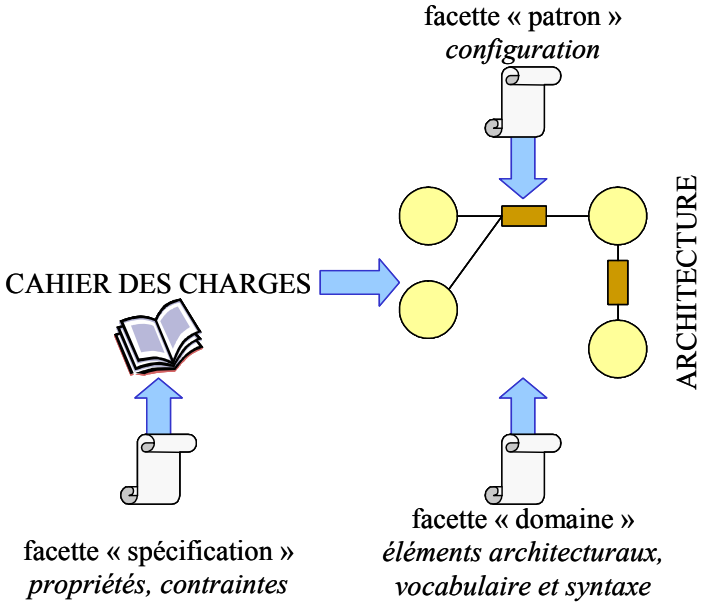



*Figure IV.2. : Les styles dans le processus de développement centré architecture*

## 2.3. La notion de style dans le cadre de l'évolution d'une architecture

La notion de style peut être utilisée pour caractériser des systèmes statiques ou dynamiques. Rappelons qu'une architecture dynamique est une architecture qui peut évoluer en cours d'exécution (Cf. Chapitre 3, § 2.2) par :

- l'ajout ou la suppression d'éléments dans l'architecture (composant ou connecteur),
- la modification d'un élément (ajout ou suppression de port),
- la modification de la topologie (par exemple, nouvelle connexion entre les composants et les connecteurs).

En définissant de telles architectures, l'architecte doit être capable d'exprimer des propriétés dynamiques telles que l'extensibilité, la personnalisation, et la modifiabilité [Med 96]. Une approche centrée style offre la possibilité de décrire les modalités d'évolution de l'ensemble des architectures d'un domaine. Ces modalités peuvent être précisées de différentes manières (en fonction des facettes) :

- la facette « spécification » peut contraindre l'évolution des architectures par la définition, à travers des contraintes, d'un espace de définition d'architecture,
- la facette « domaine » peut fournir une bibliothèque d'éléments incluant des mécanismes propres à la gestion de l'évolution dynamique (par exemple, la création dynamique de port),
- la facette « patron » peut préciser l'évolution de la configuration d'une architecture.

De plus, si la définition des facettes de style s'appuie sur une approche formelle, il est possible d'exprimer des propriétés dynamiques qui sont respectées par toutes les architectures d'un domaine. Dans le cadre de nos travaux, nous utilisons la facette « patron » pour formaliser les architectures « intra-entreprise », « inter-entreprise » et « EAI » qui sont dotées de propriétés spécifiques telles que la modifiabilité, la performance et la sûreté.

# 3. Formalisation des styles architecturaux : Le langage ASL [Ley 04]

Ce paragraphe présente le langage ASL – *Architectural Style Language* – qui permet de construire des styles formalisés. A travers ce langage, le style permet de construire des valeurs qui peuvent représenter une donnée, une connexion, une architecture, un élément architectural ou un comportement. Dans le cadre de nos travaux, nous limitons l'utilisation du style à la génération d'une architecture.

Dans un premier temps, nous présentons la définition des différents éléments composant un style. Dans un second temps, nous décrivons, de façon détaillée, l'utilisation de deux éléments particuliers du langage : le *constructeur* et la *contrainte*.

## 3.1. Concepts généraux du langage ASL

Un style est caractérisé par un nom (et éventuellement le style parent qu'il spécialise), un ensemble de types, un ensemble de styles (agrégats), un ensemble de constructeurs, un ensemble de contraintes et un ensemble d'analyse. L'encadré suivant présente la structure générale d'un style.

```
Nom_du_style is style extending Style_parent where {
    types { -- Définitions de types -- }
    styles { -- Définitions de styles -- }
    constructors { -- Définitions de constructeurs -- }
    constraints { -- Définitions de contraintes -- }
    analyses { -- Définitions d'analyses -- }
}
```

### 3.1.1. Le nom

Le ***nom*** du style permet de le référencer lors de son instanciation, lors de la vérification de la satisfaction d'une architecture par rapport à ce style et lors de son utilisation dans une relation d'héritage.

Un mécanisme d'***héritage*** (*extending*) offre la possibilité de construire un nouveau style à partir de la définition d'un style parent. Ce mécanisme permet la réutilisation d'un style existant et la définition de différents niveaux d'abstraction (généralement, du plus abstrait au plus concret)[24].

```
IC is style extending Component where { … } -- Le style IC hérite du style Component --
```

### 3.1.2. La clause *types*

La définition d'un style peut nécessiter la définition d'un ensemble de ***types***, aussi cette clause précise les types utilisés dans le style. Ces derniers sont définis à partir des types prédéfinis proposés dans le langage :

- les types de base : *Boolean*, *Natural*, *Integer*, *Real*, *String*,
- les types de construction : *Tuple*, …,
- les types collections : *Bag*, *Set*, *Sequence*,
- les types liés aux architectures : *Connection*, *Behaviour*, *Abstraction*$[T_1,...,T_n]$,
- *AnyType*, *Expression[ T ]*, *Alias*.

```
types {
    Info_div is any, -- Le type Info_div est de type "quelconque" --
    ...
    }
```

### 3.1.3. La clause *styles*

La définition d'un style peut nécessiter la définition d'un ensemble de ***styles***. Aussi, cette clause précise l'ensemble des styles agrégats qui sont utilisables au sein de ce style. Ces styles peuvent ensuite être composés car le langage propose un mécanisme de composition qui est l'agrégation. Il est à noter que les styles agrégats sont définis par une *description de style* (Cf. le style *PS* dans l'exemple suivant) ou par *référence* à un style déjà défini en dehors de la description (Cf. le style *IC* dans l'exemple suivant).

[24] Dans la mesure du possible, nous illustrons les différents concepts par des exemples issus des styles construits dans le cadre du projet ArchWare [BBD 03a] [BBD 03c].

```
Enterprise is style extending Component where {
    styles {
        IC is IC;                  -- Définition d'un style par référence (le style IC est défini
                                   en dehors du style Enterprise) --
        PS is style where{…};      -- Définition d'un style par description (ce style PS ne peut
                                   être référencé en dehors du style Enterprise) --
    }
    …
}
```

Nous pouvons noter que lors d'une extension, un style hérite des styles agrégats de son parent et qu'il peut définir ses propres styles agrégats. Dans l'exemple précédent, *Enterprise* hérite des styles agrégats du style *Component*.

### 3.1.4. La clause *constructors*

La clause ***constructors*** définit l'ensemble des constructeurs de style qui sont utilisés pour instancier une architecture. L'utilisation d'un constructeur est détaillée dans le paragraphe 3.2.

```
Enterprise is constructor (c : Integer);
    { ic is IC,
      ps is PS,
      rms is RMS { -- Constructeur du style Enterprise -- }
    }
```

Un style hérite des constructeurs du style parent, il peut aussi définir ses propres constructeurs. Ces derniers peuvent surcharger les constructeurs du style parent. Lorsqu'il y a une ambiguïté, il est possible de différencier un constructeur défini dans un style parent, d'un constructeur défini dans le style en utilisant le symbole @. Ainsi, le constructeur d'un style parent peut être appelé par *nom_parent@nom_constructeur*.

### 3.1.5. La clause *constraints*

La clause ***constraints*** définit les assertions précisant les caractéristiques que doivent respecter une architecture. L'utilisation d'une contrainte est détaillée dans le paragraphe 3.3.

```
constraints {
    -- Cette contrainte précise qu'une architecture ne peut contenir qu'un seul élément de
    style PS --
    To elements apply {
        Exists ([1]x | x in style PS)
    }
}
```

### 3.1.6. La clause *analyses*

La clause ***analyses*** définit une étude ou un traitement automatisé d'une architecture pour en déterminer ou quantifier une ou plusieurs caractéristiques[25]. Les caractéristiques identifiées peuvent être de différentes sortes. Il peut s'agir d'une vérification ou d'une mesure. Une analyse peut vérifier la satisfaction d'une propriété par l'architecture (par exemple, une analyse permet de vérifier que la structure de l'architecture est cyclique). Une analyse peut

[25] Dans le cadre de nos travaux, nous n'avons pas utilisé cette clause.

également aboutir à une mesure (par exemple, l'analyse retourne le nombre d'éléments d'une architecture).

Les analyses sont associées aux styles car les caractéristiques évaluées dépendent de la construction d'une architecture. Il existe différents moyens pour évaluer une caractéristique car celle-ci peut être évaluée différemment selon la nature de l'architecture. Par exemple, la façon de quantifier la modifiabilité d'une architecture dépend du style de l'architecture. En effet, cette caractéristique est quantifiée de deux manières différentes selon que l'architecture respecte un style « en étoile » ou « tableau noir » [KlK 99].

## 3.2. La définition et l'utilisation d'un constructeur

Rappelons qu'un constructeur permet d'instancier une architecture. Son utilisation offre plusieurs avantages parmi lesquels nous pouvons citer la définition d'une syntaxe spécifique qui, par construction, peut être proche des concepts d'un domaine particulier. Elle favorise ainsi la compréhension d'une architecture.

### 3.2.1. La définition d'un constructeur

Un constructeur est défini avec un nom, un ensemble de paramètres typés, un contexte, un corps et une notation mixfix.

```
nom_du_constructeur is constructor ( -- paramètres formels -- );
       context( -- variables de contexte -- );
       { -- corps du constructeur -- }
       as { -- notation mixfix -- }
```

L'instanciation est réalisée en fonction d'une valeur dépendant ou non d'un contexte donné et des valeurs des paramètres effectifs. Nous appelons ***espace de construction***, l'ensemble des architectures qui peuvent être générées par le constructeur.

Le langage propose des types supplémentaires pour décrire les ***paramètres*** des constructeurs :

- Type,
- Expression[ T ] permet de définir des valeurs de type T. Par exemple, un paramètre de type ***Expression[ Behaviour ]*** est la définition d'un comportement et non pas une instance d'un comportent,
- Alias dénote une référence (et non une instance).

Le ***contexte*** est défini pour simplifier l'utilisation des constructeurs en rendant transparent à l'architecte les mécanismes sous-jacents à un style. Par exemple, supposons qu'un élément est composé d'un ensemble de ports et d'un comportement. Le constructeur *include_port* permet d'inclure un nouveau port à l'élément. Logiquement, l'application de ce constructeur nécessite de préciser le port (à inclure) et l'ensemble dans lequel il doit être inclus. Ainsi, l'appel du constructeur serait décrit par :

```
…
Include_port(ensemble_de_port, nouveau_port)
…
```

Si nous considérons que les ports sont toujours inclus dans le même ensemble, la déclaration de *ensemble_de_port* dans le contexte permet de simplifier l'appel du constructeur :

```
…
Include_port(nouveau_port)
…
```

Le ***corps*** d'un constructeur décrit la manière dont une valeur est construite en fonction des paramètres du constructeur et du contexte. Dans notre cas, le corps définit comment l'architecture est instanciée. Les clauses peuvent être :

- une déclaration de valeur ou de type (***value*** *identifiant* ***is*** *valeur* ou ***type*** *identifiant* ***is*** *type*),
- une configuration,
- une action de communication,
- une action inobservable,
- une valeur d'un type défini par le langage :
  - Integer,
  - sequence[…],
  - connection[…],
  - behaviour,
  - abstraction[…],
  - etc.

L'exemple suivant présente une abstraction. Cette abstraction représente le comportement du composant PS. Il envoie une valeur sur la connexion *Objective_PS* et reçoit une valeur sur la connexion *Result_PS*.

```
PS is constructor(); {
    … -- définition des éléments de l'architecture (ports du composant) --
    abstraction ();{
        via Objective_PS send;
        via Result_PS receive;
    } …
}
```

La notation ***mixfix*** est un mécanisme de notation associée à l'utilisation d'un constructeur. Il fournit un langage spécifique à un domaine particulier pour décrire des architectures. La sémantique associée à cette syntaxe est spécifiée par le constructeur.

Cette notation introduit des termes propres à un domaine pour construire des expressions linguistes proches du domaine d'intervention. Le symbole $ est utilisé pour différencier les termes propres au domaine des paramètres du constructeur.

```
Resource named $nom is $Resource_State
```

En utilisant cette notation mixfix, nous pouvons appliquer le constructeur pour définir un nouvel élément *Resource*. Cette ressource, appelée *ERP*, envoie une valeur précisant son état.

```
Resource named "ERP" is "unable"
```

### 3.2.2. Utilisation d'un constructeur

Dans un premier temps, nous présentons deux utilisations générales d'un constructeur : application totale et application partielle. Nous décrivons ensuite, l'utilisation d'un constructeur pour configurer une architecture. Enfin, nous présentons l'utilisation du constructeur pour composer des architectures.

#### *3.2.2.1 Application totale ou partielle d'un constructeur*

Il existe deux types d'utilisation d'un constructeur :

- l'application totale,
- l'application partielle.

L'***application totale*** d'un constructeur est la génération d'une valeur en fonction d'une liste de paramètres et d'un contexte spécifique. L'application est dite 'totale' car il est nécessaire de fournir une valeur pour l'ensemble des paramètres du constructeur.

Dans l'exemple suivant, l'identifiant *entreprise* spécifie le contexte minimal requis pour l'application.

```
value entreprise is "Thésame";
value horloge is client("horloge","heure");
…
```

Cette description est équivalente à la description suivante :

```
value entreprise is "Thésame";
value horloge is {
    value nom is "horloge";
    value requete is "heure";
    value envoi is tuple(nom, entreprise, requete);
    abstraction ();{
        via call send envoi;
        via wait receive reponse:String;
        unobservable
    }
};
…
```

Donc, la valeur évaluée pour *horloge* (dans son contexte) lors de l'application totale est la suivante :

```
abstraction ();{
        via call send tuple("horloge", "Thésame", "heure");
        via wait receive reponse:String;
        unobservable
    }
```

Aussi, comme nous l'avons mentionné auparavant, nous pouvons utiliser la notation mixfix pour appliquer un constructeur. Nous aurions pu écrire :

```
value entreprise is "Thésame";
value horloge is client named "horloge" requiring "heure";
```

L'***application partielle*** d'un constructeur génère un constructeur en fonction d'un constructeur plus générique et d'un ensemble de paramètres. Une application partielle est exprimée en faisant référence à un constructeur et en donnant une liste de paramètres. Syntaxiquement, à la différence d'une application totale, il est nécessaire d'associer l'identifiant du paramètre à la valeur donnée.

Supposons qu'il existe un constructeur appelé *constructeur_generique* défini par :

- les trois paramètres de types suivants ($p_1$:***Integer***, $p_2$:***Real***, $p_3$:***set[Integer]***),
- un constructeur $c_1$.

```
constructeur_generique is constructor (p1:Integer, p2:Real, p3:set[Integer]);
       ( c1:Type_c1 );
       { -- corps_du_constructeur -- }
       as { -- notation_mixfix -- }
```

Supposons l'application partielle suivante qui ne précise les valeurs que pour $p_1$ et $p_3$.

```
constructeur_generique (1 as p1, set(1,2,3) as p3);
```

Le constructeur équivalent à cette application est le suivant : le paramètre $p_1$ n'est pas paramètre du constructeur généré, sa valeur est directement fixée à 1 ; le paramètre $p_3$ est paramètre du constructeur généré, mais une partie des éléments de l'ensemble est aussi fixée directement.

```
constructor (p2:Real, p3:set[Integer]);
       ( c1:Type_c1 );
       {   p1 is 1;
           p3 includes 1;
           p3 includes 2;
           p3 includes 3;
           -- corps du constructeur --
       }
```

Le mécanisme d'application partielle des constructeurs est utile lors de l'héritage car il permet de spécialiser des constructeurs du style parent pour qu'ils soient plus adaptés au style. Par exemple, supposons un style EAI proposant un constructeur pour générer une architecture à partir de la définition des progiciels du marché (COTS – *Components Of The Shelf*) et de leur nombre.

```
EAI is style where {
   constructors{
      EAI is constructor(COTS:Behaviour, nb_COTS:Integer);
      {   ……….   }
   }
   …
}
```

Un style, *Deux_COTS*, propose un constructeur pour générer une architecture, ce constructeur est l'application partielle de celui du constructeur EAI.

```
Deux_COTS is style where {
    constructors{
        Deux_COTS is EAI (2 as nb_COTS)
    }
… }
```

### *3.2.2.2 Construction d'une architecture par un constructeur*

Lorsqu'un ensemble de styles agrégats est défini, le constructeur doit préciser la configuration des éléments instanciés à partir de ces styles. Une clause configuration est alors utilisée dans le corps du constructeur pour décrire les connexions à construire entre les différents éléments de l'architecture.

L'exemple suivant décrit le style *Inter_enterprises* qui s'appuient sur trois styles agrégats :

- Contractor,
- Supplier,
- Mediator.

Le constructeur précise dans un premier temps les éléments architecturaux instanciés à partir de ces styles agrégats (contractor, mediator, supplier1, supplier2). Il définit ensuite les connexions entre ces éléments.

```
Inter_Enterprises is style extending Component where {-- Le style Inter_Enterprises étend le
    styles {                                            style composant --
        Contractor is style extending Enterprise where { …… },
        Supplier is style extending Enterprise where { …… },
        Mediator is style extending Connector where { …… },
    }
    constructors {
        Inter_Enterprises is constructor(); { -- Constructeur du style Inter_Enterprises --
            component with {
                constituents { -- Identification des instances des éléments architecturaux --
                    contractor is Contractor,
                    mediator is Mediator,
                    supplier1 is Supplier,
                    supplier2 is Supplier
                }
                configuration { -- Création instances des éléments architecturaux --
                    new contractor;
                    new mediator;
                    new supplier1;
                    new supplier2;
                    -- Définition de la configuration de l'architecture (connexions entre les
                    ports des composants et des connecteurs) --
                    attach contractor::Contractor_port::contractor_tender to
                              mediator::Mediator_Contractor_port::contractor_tender;
                    ……
                }
            }
        }
    }
}
```

Suite à cet exemple, nous pouvons constater que le constructeur d'un style est similaire au constructeur d'une classe définie dans le cadre d'une approche orientée objet. Il peut exister

plusieurs constructeurs au sein d'un même style, chacun d'eux permettant de créer des architectures différentes.

Un constructeur peut également être utilisé pour composer des architectures. Nous illustrons nos propos par la définition d'un constructeur *étoile* qui décrit une architecture centralisée. Ce constructeur précise l'agencement des différents éléments de l'architecture.

```
etoile is constructor(centre:Behaviour, elements:sequence[Behaviour]);
{   iterate elements by e: Behaviour
    do compose {
        centre
    and
        e
    }
}
as
{ disposer $elements autour de $centre }
```

L'exemple suivant illustre l'instanciation d'une architecture à partir de ce constructeur.

```
abstraction();
{   value c is behaviour{…};
    value p is abstraction();{…};
    value p1 is p();
    value p2 is p();
    value p3 is p();
    disposer {p1, p2, p3} autour de c
}
```

L'avantage d'une telle description est de pouvoir composer des architectures et de pouvoir ainsi créer de nouvelles architectures à partir d'architectures existantes. L'exemple suivant illustre graphiquement la construction d'une architecture composée d'une architecture étoile et d'une architecture ligne.

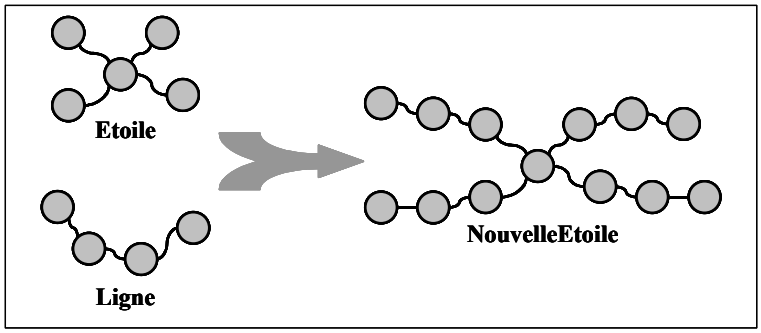



*Figure IV.3. : Composition de patrons*

## 3.3. La définition d'une contrainte

Ce paragraphe décrit de manière détaillée la définition des contraintes au sein d'un style. Rappelons qu'une contrainte est une assertion définissant des caractéristiques architecturales pouvant être évaluées sur une architecture, elles définissent les limites de l'espace de définition des architectures d'un domaine.

Nous rappelons qu'il existe différents types de contraintes (Cf. Chapitre 1, § 5) :

- les contraintes *topologiques* (par exemple, une architecture ne contient pas de cycle),
- les contraintes *comportementales* (par exemple, il n'y a pas d'interblocage),
- les contraintes d'*attribut* (par exemple, la durée de traitement d'un composant ne doit pas excéder 0.5 secondes).

La formalisation des contraintes est nécessaire pour vérifier qu'une architecture satisfait un style. Cette vérification peut être réalisée lorsqu'un architecte veut vérifier qu'une architecture particulière respecte les contraintes du style.

Une contrainte est définie avec un nom et un corps.

```
nom_de_la_contrainte is constraint {
     -- corps de la contrainte --
},
```

Une contrainte peut également être définie par référence à une contrainte d'un autre style.

```
constraints {
    nom_de_la_contrainte is référence_contrainte
}
```

Par exemple, nous pouvons écrire l'expression qui suit. La contrainte *contrainte1* du style *autre_style* est référencée par *autre_style@contrainte1* pour définir la contrainte *contrainte*.

```
constraints {
    contrainte1 is autre_style@contrainte1
}
```

Cette notion de contrainte peut être illustrée par l'exemple suivant qui spécifie qu'à n'importe quel moment il est possible de réaliser une action. De manière littérale, cette contrainte signifie que toute action aboutit à une action, cette propriété signifie donc qu'une action peut toujours être effectuée, ce qui induit qu'il ne peut exister de situation de blocage.

```
DeadLockFreedom is constraint {
    to styleInstance.behaviour apply{
        every sequence {true*} leads to state
        {some sequence {true} leads to state {true}}
    }
}
```

Le corps d'une contrainte est une propriété qui doit être décrite par un langage spécifique. La contrainte concerne l'instance d'un style. Ainsi dans l'exemple défini précédemment, la propriété de *deadLockFreedom* s'applique au comportement d'une instance.

Afin de faciliter la description des propriétés, un certain nombre de mécanismes ont été défini :

- un prédicat ***in style*** a été défini. Ce prédicat est vrai si la valeur à laquelle il est appliqué satisfait toutes les contraintes d'un style. La contrainte définie dans l'exemple suivant oblige la valeur de l'architecture instanciée à suivre le style *EAI*,

```
IsEAI is constraint {
    styleInstance in style(EAI)
}
```

- il est permis d'utiliser les constructeurs au sein des contraintes pour définir des valeurs. Toutefois, seuls les constructeurs sans contexte sont utilisables dans la description d'une contrainte,
- des fonctions peuvent être définies par l'utilisateur,
- les contraintes peuvent portées sur des actions différentes des actions de communication (par exemple, les attachements ou la création dynamique).

Les ***contraintes topologiques*** concernent la structure de l'architecture ou des éléments architecturaux. Ces contraintes ne sont pas limitées à l'initialisation de l'architecture, elles doivent être vérifiées lors de son évolution. Ces contraintes fournissent donc également des contraintes portant sur la dynamique d'une architecture. Elles peuvent contraindre :

- l'interface, en termes de type et de cardinalité,
- la nature des constituants,
- la quantité des constituants,
- les connexions entre les constituants,
- les connexions entre les constituants et l'interface de l'architecture.

Considérons *elements* comme l'ensemble des éléments constituants une architecture. Supposons que nous souhaitons réduire les styles de ces constituants à trois catégories. L'expression suivante précise que pour tout élément *x* de *elements*, *x* satisfait soit le style *IC*, soit le style *PS*, soit le style *RMS.*

```
to styleInstance.elements apply {
    forall (x | x in style IC xor x in style PS xor x in style RMS)
}
```

Cette contrainte est une ***contrainte forte*** dans le sens où elle ne laisse aucun choix à l'architecte. Une contrainte plus flexible pourrait seulement obliger l'existence de constituants d'un certain style, sans empêcher l'existence d'autres styles de constituants. L'expression suivante précise qu'il existe au moins un élément satisfaisant le style *IC*.

```
to styleInstance.elements apply {
    exists (x | x in style IC)
}
```

De plus, nous pouvons imposer des contraintes sur le nombre de constituants satisfaisant un style. L'expression suivante précise qu'il existe entre 1 et 5 constituants de *IC*.

```
to styleInstance.elements apply {
    exists (x | x in style IC and
               apply on x_set union set(x) and
               x_set.size>=1 and x_set.size<=5 )
}
```

Le langage fournit également une facilité d'expressions des cardinalités. Ainsi, l'expression suivante est équivalente à la précédente.

```
to styleInstance.elements apply {
   exists ([1..5] x | x in style IC)
}
```

Concernant les liens de communication entre les éléments, le langage fournit un prédicat, *rename* pour vérifier que deux connexions sont unifiées. Il fournit également un prédicat, *attached to*, basé sur le prédicat précédent pour vérifier que deux comportements partagent une connexion. L'expression suivante précise que deux éléments de *IC* ne peuvent être connectés.

```
to styleInstance.elements apply {
   forall(x,y | x in style IC and y in style IC
              implies not(x attached to y))
}
```

Les ***contraintes comportementales*** définissent des patrons sur l'ordonnancement des actions dans un comportement. Ainsi, ces contraintes permettent d'imposer la satisfaction de propriétés telles que la vivacité[26] ou la sûreté[27].

La contrainte suivante signifie qu'il existe deux connexions nommées *request* et *answer*, et qu'à chaque séquence d'actions terminée par une réception via la connexion *request*, il existe une séquence aboutissant à un envoi via la connexion *answer*. En d'autres termes, une réception par la connexion *request* entraîne un envoi par la connexion *answer***.** Cette contrainte est une contrainte de vivacité.

```
to styleInstance.connections apply{
   exists(r,a | r.name=request and a.name=answer and
       every sequence{true*.via r receive any}
       leads to state{true*.via a send any}
}
```

Les ***contraintes d'attribut*** sont des contraintes sur les données contenues par un élément architectural. Ces données peuvent, par exemple, rendre compte de la capacité de stockage, de la vitesse de traitement d'un élément. Nous pouvons définir des contraintes sur :

- les attributs que doit définir un élément,
- les valeurs permises pour un attribut,
- les relations entre les attributs.

Tout comme les contraintes topologiques, ces contraintes sont décrites par des prédicats sur les données. Elles doivent être vérifiées durant toute évolution de l'architecture. La propriété suivante signifie qu'il doit exister une valeur dont le nom est *latency*.

```
to styleInstance.values apply{
       exists(x | x.name=latency )
}
```

[26] Un système est ***vivace*** s'il a toujours la possibilité d'effectuer une action.
[27] Un système est ***sûr*** lorsqu'il ne s'exécutera jamais d'une manière explicitement interdite.

Nous pouvons contraindre la valeur de cet attribut. Par exemple, l'expression suivante signifie qu'il existe une valeur nommée *latency* et que sa valeur est strictement supérieure à 0 et strictement inférieure à 100. En d'autres termes, la valeur de l'attribut *latency* doit être strictement comprise entre 0 et 100.

```
to styleInstance.values apply{
       exists(x | x.name=latency and x.value>0 and x.value<100)
}
```

De plus, il est possible de définir des relations de contraintes entre deux ou plusieurs attributs. Par exemple, l'expression suivante signifie littéralement qu'il existe un couple de valeurs (*x*, *y*) tel que x est nommé *height*, y nommé *width* et *x* soit strictement inférieur à *y*. En d'autres termes, il est spécifié qu'il doit exister les attributs *height* et *width*, et que la valeur de *height* doit être plus petite que celle de *width*.

```
to styleInstance.values apply {
   exists (x,y | x.name=height and
              y.name=width and
              x.value<y.value)
}
```

Lors de l'héritage d'un style, toutes les contraintes du style parent sont héritées. Toutes les contraintes d'un style parent sont des contraintes de tous ses styles. Un style peut ajouter des contraintes sans modifier les contraintes héritées. Ainsi, il décrit un espace de définition strictement inclus dans celui de son parent.

L'héritage permet à l'architecte de style de bénéficier des caractéristiques d'un style pour les appliquer à celui qu'il construit.

L'expression des contraintes permet de définir un héritage multiple. Nous pouvons définir un style par la composition de plusieurs styles en composant leurs contraintes. L'héritage multiple est obtenu par une composition avec l'opérateur ***and*** comme le montre l'exemple suivant.

```
Style_Composite is style where{
   constraints {
      composition_style_1_2 is constraint {
             styleInstance in style Style_1
         and
             styleInstance in style Style_2
      }
   }
}
```

# 4. Cas d'application

Nous concluons ce paragraphe en présentant, de façon détaillée, deux styles issus de notre application pour illustrer nos propos. Ces descriptions présentent les styles décrivant le système de performance, le composant industriel et le connecteur entre ces deux composants (Cf. Chapitre 3, § 4.1.1) [BBD 03a] [BBD 03c]. L'annexe 5 fournit la description de l'ensemble des styles définis dans le cadre de notre projet.

## 4.1. Le système de performance (SP)

Ce style hérite du style *composant* et il encapsule la description du style formalisant un port disposant de deux connexions.

```
PS is style extending Component where {
    -- Le style PS étend le style composant --
    styles {
        Performance_port is style extending Port where {
            -- Définition du port pour interfacer le connecteur pour le IC --
            constructors {
                Performance_port is constructor(); {
                    -- Définition du constructeur du port --
                    port with {
                        connections {
                            -- Définition des connexions du port --
                            obj_PS is connection(any),
                            result_PS is connection(any)
                        }
                    }
                }
            }
        }
    }
```

Le constructeur de ce style définit le port et le comportement du système de performance.

```
    constructors {
        PS is constructor(); {
            -- Constructeur du style PS --
            component with {
                ports {
                    Performance_port is Performance_port
                }
                computation {
                    -- Définition du comportement du composant PS --
                    via obj_PS send objective;
                    unobservable;
                    via result_PS receive result:any;
                    recursive
                }
                configuration {
                    new Performance_port
                }
            }
        }
    }
```

Ce style définit une contrainte topologique qui précise qu'il doit exister au moins un port.

```
        constraints {
            -- Il y a au moins un port du PS pour communiquer avec un IC --
            To ports apply {
                Exists (p:1..*)([p]x | x in archetype Performance_port)
            }
        }
    },
```

## 4.2. Le composant industriel (CI)

Ce style hérite également du style *composant* et il encapsule la description de trois styles formalisant un port disposant de deux connexions :

- un port permettant la communication avec un autre composant industriel,
- un port permettant la communication avec le système de performance,
- un port permettant la communication avec le système de gestion de ressources.

```
    IC is style extending Component where {
        -- Le style IC étend le style composant --
        styles {
            IC_IC_port is style extending Port where {
                -- Définition du port pour interfacer le connecteur pour un autre IC --
                constructors {
                    IC_IC_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                input is connection(any),
                                output is connection(any)
            }}}}},
            IC_PS_port is style extending Port where {
                -- Définition du port pour interfacer le connecteur pour le PS --
                constructors {
                    IC_PS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                obj_IC is connection(any),
                                result_IC is connection(any)
                            }
                        }
                    }
                }
            },
            IC_RMS_port is style extending Port where {
                -- Définition du port pour interfacer le connecteur pour le RMS --
                constructors {
                    IC_RMS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                query_IC is connection(any),
```

```
                        response_IC is connection(any)
                    }
                }
            }
        }
    }
}
```

Le constructeur de ce style définit les trois ports nécessaires au composant industriel et le comportement de ce composant.

```
constructors {
    IC is constructor(); {
        -- Constructeur du style IC --
        component with {
            ports {
                IC_Mediator_port is IC_Mediator_port,
                IC_PS_port is IC_PS_port,
                IC_RMS_port is IC_RMS_port
            }
            computation {
                -- Définition du comportement du composant IC --
                choose {
                    {
                        -- Gestion des flots d'entrée/sortie du composant IC --
                        via input receive input:any;
                        unobservable;
                        via output send output
                    }
                Or
                    {
                        -- Gestion des flots d'information relatif au PS du --
                        -- composant IC --
                        via obj_IC receive objective:any;
                        unobservable;
                        via result_IC send result;
                        recursive
                    }
                or
                    {
                        -- Gestion des flots d'information relatif au PS du --
                        -- composant RMS --
                        via query_IC send query;
                        unobservable;
                        via response_IC receive response:any;
                        recursive
                    }
                }
            }
            configuration {
                new IC_IC_port;
                new IC_PS_port;
                new IC_RMS_port
            }
        }
    }}
```

Ce style définit une contrainte topologique qui précise qu'il doit exister au moins un port pour communiquer avec un autre composant industriel ou avec le port du composite. Une deuxième contrainte comportementale précise que la réception d'un objectif induit une requête vers le système de gestion de ressources.

```
        -- Définition de quelques contraintes du style IC --
        constraints {
            -- Il y a au moins un port au niveau du IC pour communiquer avec le composite --
            To ports apply {
                Exists ([1]x | x in archetype IC_IC_port)
            }
            -- Après la réception d'un objectif, il y a une requête vers le RMS --
            To ports apply {
                Exists (p1 : IC_PS_port, p2 : IC_RMS_port |
                    Every sequence {true*.via p1::obj_IC receive any}
                        Leads to state {true*.via p2::query_IC send any}
                )
            }
        }
    },
```

## 4.3. Le connecteur entre SP et CI

Ce style hérite également du style *connecteur* et il encapsule la description de deux styles formalisant deux ports disposant de deux connexions :

- un port permettant la connexion avec un composant industriel,
- un port permettant la connexion avec le système de performance.

```
    Coop is style extending Connector where {
        -- Le style Coop étend le style connecteur --
        styles {
            IC_PS_conPS_port is style extending Port where {
                -- Définition du port pour interfacer le IC --
                constructors {
                    IC_PS_conPS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                obj_IC is connection(any),
                                result_IC is connection(any)
                            }
                        }
                    }
                }
            },
            Performance_conPS_port is style extending Port where {
                -- Définition du port pour interfacer le PS --
                constructors {
                    Performance_conPS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                obj_PS is connection(any),
                                result_PS is connection(any)}}}}}}
```

Le constructeur de ce style définit les deux ports du connecteur et le protocole d'échange.

```
        constructors {
            Coop is constructor(); {
                -- Constructeur du style ConPS --
                connector with {
                    ports {
                        IC_PS_conPS_port is IC_PS_conPS_port,
                        Performance_conPS_port is Performance_conPS_port
                    }
                    routing {
                        -- Définition du comportement du connecteur ConPS --
                        via obj_PS receive objective:any;
                        via obj_IC send objective;
                        via result_IC receive result:any;
                        via result_PS send result
                    }
                    configuration {
                        new IC_PS_conPS_port;
                        new Performance_conPS_port
        }}}}},
```

# 5. Conclusion

Dans ce chapitre, nous avons présenté les principaux mécanismes du langage ASL – *Architectural Style Language* – qui est un langage permettant la description de styles architecturaux. Ce langage est utilisé avant les phases de développement d'une architecture pour formaliser un domaine d'application.

En effet, les styles architecturaux concentrent l'expérience et les connaissances de conception architecturale d'un point de vue général ou pour des domaines spécifiques. Les styles architecturaux offrent un cadre pour la conception architecturale en permettant la formalisation de « patrons » d'architecture.

Dans le cadre de notre projet, ce langage nous a permis de formaliser les architectures propres à notre domaine d'application (Cf. Chapitre 3, § 4).

# Chapitre 5 :
# Mise en œuvre, implémentation et prototypage

# Chapitre 5 :
# Mise en œuvre, implémentation et prototypage

## 1. Introduction

Les chapitres précédents nous ont permis de présenter une modélisation des processus par l'utilisation d'un langage formel centré architecture pour appréhender les points de vue « métier » et « donnée » à travers un ensemble d'architectures industrielles et logicielles évolutives. Rappelons que nous considérons que cette approche est nécessaire pour permettre à l'entreprise d'être agile dans un environnement évolutif.

Ces travaux se sont déroulés dans le cadre du projet européen ArchWare qui a pour objectif de développer un ensemble de langages et d'outils centrés architecture pour l'ingénierie des systèmes logiciels évolutifs. Aussi, ce chapitre présente l'utilisation de l'environnement pour valider nos propositions. Plus précisément, cette validation vise à développer une architecture industrielle capable de formaliser les processus opérationnels d'une entreprise. Le déroulement de ces processus s'appuie sur une architecture logicielle adaptée à la gestion industrielle des PMI/PME. Cette architecture s'articule autour d'un moteur de workflow qui intègre un ensemble d'applications dédiées à la gestion industrielle telles des progiciels de gestion intégrés, des logiciels de gestion de la qualité, des logiciels de gestion de maintenance, etc.

La première partie de ce chapitre présente l'environnement logiciel proposé par le projet ArchWare. Dans un second temps, nous décrivons la problématique abordée à travers notre cas d'étude. Enfin, nous présentons une synthèse des résultats obtenus pour répondre à cette problématique.

## 2. Présentation de l'environnement ArchWare

L'objectif de ce paragraphe est de décrire l'environnement proposé par le projet ArchWare [Arc 02] qui a servi de cadre applicatif à nos travaux. Dans un premier temps, nous présentons une vision globale de l'environnement ArchWare. Nous décrivons ensuite les différents langages proposés par cet environnement. Enfin, nous présentons les outils ArchWare développés dans le cadre de ce projet[28].

### 2.1. Vision globale de l'environnement ArchWare

Nous rappelons que l'objectif principal du projet ArchWare est de fournir les éléments nécessaires à l'ingénierie de systèmes logiciels évolutifs. Afin d'atteindre cet objectif, le projet ArchWare développe un ensemble intégré de langages et d'outils centrés architecture en se basant sur un cadre d'exécution persistant.

[28] Dans le cadre de ce projet, nous n'avons pas participé au développement de ces langages et ces outils mais nous les avons utilisés pour traiter une problématique spécifique.

Le projet ArchWare a donc pour objectif de concevoir, développer et disséminer :

- des langages innovateurs centrés architecture (langage de description d'architecture et langage d'analyse),
- des styles architecturaux,
- des modèles de raffinement,
- des environnements logiciels personnalisables,
- des outils dédiés à l'ingénierie de systèmes logiciels évolutifs.

ArchWare vise à construire un environnement personnalisable d'ingénierie logiciel qui peut être utilisé pour créer des environnements logiciels centrés architecture. Ces environnements doivent donc être facilement adaptables aux différentes organisations, grâce à la personnalisation des processus de développement et des domaines d'applications ciblés. Les mécanismes clés de cette personnalisation sont les modèles architecturaux qui sont formalisés par des styles architecturaux. En d'autres termes, les concepts développés dans le cadre du projet pour l'ingénierie des systèmes évolutifs sont appliqués à l'environnement ArchWare.

Ce projet considère qu'un environnement personnalisable centré architecture est structuré en deux couches distinctes, à savoir un cadre d'exécution et un ensemble d'outils centrés architecture.

Le ***cadre d'exécution ArchWare*** inclut un moteur d'exécution d'architectures basé sur des processus de développement évolutifs, un processus de raffinement de description d'architecture et des mécanismes supportant l'interopérabilité des outils de l'environnement.

Les ***outils centrés architecture ArchWare*** fournissent des supports pour :

- la définition des styles architecturaux (de nouveaux styles peuvent être définis en spécialisant des styles existants, une hiérarchie de styles peut être fournie),
- la construction, en appliquant les styles définis, d'une représentation symbolique des architectures grâce à des profils UML et des profils capables de personnaliser l'environnement,
- la validation des architectures (respect des besoins propres à un domaine),
- la vérification des propriétés fonctionnelles et extra fonctionnelles des architectures,
- le raffinement des descriptions d'architecture depuis un niveau abstrait jusqu'à un niveau concret,
- la génération de code des systèmes dans différents langages de programmation (cette génération de code utilise des règles explicites).

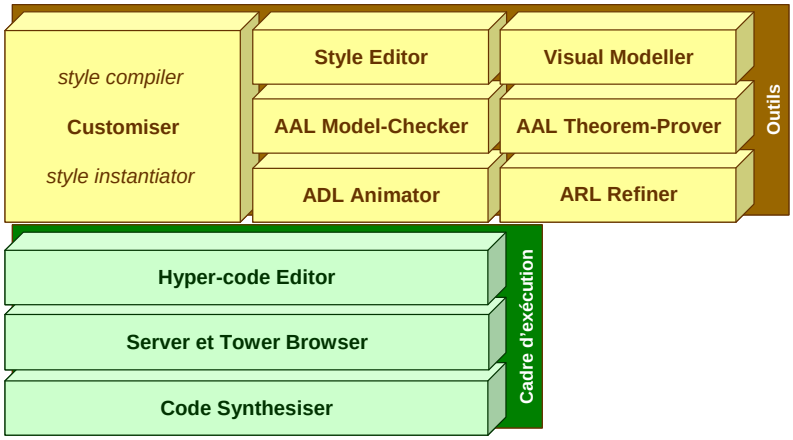



*Figure V.1. : L'environnement ArchWare*

Avant de présenter chacun de ces outils, nous présentons succinctement les différents langages développés dans le cadre de ce projet.

## 2.2. Les langages proposés par l'environnement ArchWare

Le projet européen ArchWare fournit un ensemble de langages centrés architecture :

- l'***ArchWare Architecture Description Language*** (ADL) qui est un langage pour la description d'architectures dynamiques (évolutives),
- l'***ArchWare Architecture Analysis Language*** (AAL) qui est un langage pour la description de propriétés architecturales,
- l'***ArchWare Architecture Refinement Language*** (ARL) qui est un langage pour la description de raffinements d'architectures,
- **l'*ArchWare Hyper-code Architecture Description Language*** (Hyper-code ADL) qui est un langage pour la gestion de l'évolution au sein d'une architecture en cours d'exécution,
- l'***ArchWare Architecture eXchange Language*** (AXL) [VeO 03] qui est un langage d'échange pour la communication entre les outils de l'environnement ArchWare.

Le langage ASL présenté précédemment (Cf. Chapitre 4, § 3) intègre les langages ArchWare ADL et ArchWare AAL.

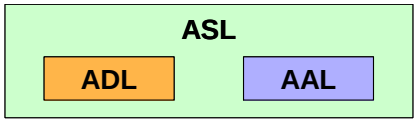



*Figure V.2. : La composition du langage ASL*

Dans notre approche, nous nous appuyons sur ces différents langages et notamment sur l'ASL (ArchWare ADL et ArchWare AAL), l'ArchWare ARL et l'ArchWare Hyper-code ADL. Au sein du prochain paragraphe, nous étudions les différents concepts sous-jacents à ces quatre langages. Nous ne présentons pas, dans le cadre de cette thèse, le langage ArchWare AXL. En effet, ce langage est un langage utilisé par l'environnement, il ne concerne donc pas le développement de systèmes logiciels.

### 2.2.1. Le langage ArchWare ADL [OAC 02] [COB 02]

L'ArchWare ADL est un langage formel conçu pour supporter la spécification exécutable d'architectures logicielles dynamiques et évolutives. Il est fondé sur le π-calcul[29] [Mil 99]. L'ArchWare ADL est défini comme une extension du π-calcul typé d'ordre supérieur[30] pour un domaine spécifique : c'est une extension « bien-formée » pour définir un calcul d'éléments architecturaux mobiles et communicants. Ce langage permet donc de formaliser la structure et le comportement au sein d'une même description.

En ArchWare ADL, une ***architecture*** est un ensemble d'éléments, appelés éléments architecturaux, qui sont reliés par des liens de communication. Ces éléments sont définis en terme de comportements (***behaviour***). Ce dernier est défini par un ensemble d'actions ordonnancées qui spécifie le traitement interne de l'élément (actions internes) et les interactions avec son environnement (actions de communication).

[29] L'annexe 3 présente les principales caractéristiques du π-calcul.
[30] Dans cette version du π-calcul, les variables sont typées et des processus peuvent transiter via des canaux de communication.

Un élément architectural communique avec les autres par une interface caractérisée par un ensemble de connexions (***connections***) qui permettent de faire transiter des données. Un mécanisme de composition et un mécanisme de liaison, appelé ***unification*** (c'est une substitution au sens du π-calcul) permettent la mise en relation des éléments architecturaux. Ces éléments peuvent interagir lorsqu'ils sont composés et liés.

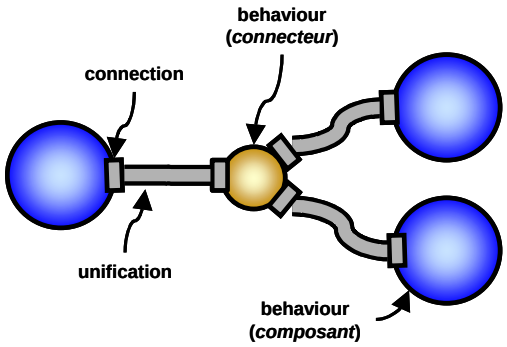



*Figure V.3. : Description informelle d'une architecture en ArchWare ADL*

Un élément architectural peut être défini comme une composition d'autres éléments (composite). En d'autres termes, un comportement peut être défini comme un ensemble d'autres comportements interconnectés.

L'ArchWare ADL permet la description ***d'architectures dynamiques***. En effet, les éléments architecturaux peuvent d'une part, être composés ou décomposés « à la volée »[31] et, d'autre part, des éléments architecturaux peuvent être créés et liés dynamiquement. Enfin, des éléments architecturaux peuvent transiter comme des données à travers les connexions.

### 2.2.2. Le langage ArchWare AAL [AGM 02]

L'ArchWare AAL est un langage formel pour exprimer les propriétés des architectures logicielles évolutives modélisées en ArchWare ADL. L'ArchWare AAL est défini comme une extension du μ-calcul [Koz 83] pour exprimer à la fois des propriétés structurelles et comportementales. En effet, le μ-calcul étant limité à l'expression de propriétés comportementales, il est nécessaire d'étendre ce calcul pour exprimer les propriétés structurelles. Aussi, l'ArchWare AAL s'appuie sur la logique des prédicats pour exprimer ces propriétés. L'ArchWare AAL fournit donc un support à la vérification de propriétés architecturales, cette vérification est implémentée par « *theorem proving* » et « *model checking* ».

En ArchWare AAL, les propriétés sont définies comme des formules prédicatives. Ce langage fournit des prédicats prédéfinis et des mécanismes (opérateurs et quantificateurs) pour construire de nouveaux prédicats.

### 2.2.3. Le langage ArchWare ARL [Oqu 03]

L'ArchWare ARL est un langage formel basé sur le π-calcul, dédié au raffinement d'architectures logicielles avec d'une part, la prise en compte du raffinement des données, des comportements, des connexions et des structures et, d'autre part, la préservation des propriétés architecturales à travers le processus de raffinement. Le noyau du langage est un ensemble d'opérations supportant des transformations architecturales. L'ArchWare ARL

[31] En cours d'exécution.

fournit donc un ensemble d'opérations appelées *actions de raffinement* pour la transformation d'architectures.

Le raffinement d'une architecture abstraite vers une architecture concrète en vue de son implémentation se fait en plusieurs étapes successives [Meg 04]. Chaque étape implique l'application d'une action de raffinement qui fournit une solution architecturale correcte. Dans l'ArchWare ARL, les actions de raffinement sont exprimées par des *pré-conditions*, des *transformations* et des *post-conditions*.

Les pré-conditions sont des conditions qui doivent être satisfaites dans une architecture avant l'application d'une action de raffinement. Les post-conditions doivent, quant à elle, être satisfaites suite à l'application d'une action de raffinement. Une transformation exprimée par une action de raffinement montre comment une architecture satisfaisant les pré-conditions peut être transformée en une architecture satisfaisant les post-conditions.

## 2.3. Le cadre d'exécution ArchWare

Le cadre d'exécution fourni par le projet ArchWare s'appuie sur un outil « *Server* », un navigateur « *Tower Browser* » et le langage « *ArchWare Hyper-code ADL* ».

### 2.3.1. Le Server et le Tower Browser [GRS 03]

L'outil *Server* et le navigateur *Tower Browser* sont les éléments du cadre d'exécution qui permettent d'intégrer les différents outils logiciels définis dans le cadre de ce projet.

Le *Server* est une machine virtuelle évoluée au sein de laquelle toutes les descriptions d'architectures, de styles, de propriétés sont stockées. Ce serveur dispose également de fonctionnalités telles que la capacité de compiler et d'exécuter des programmes écrits en ArchWare ADL.

Le *Tower Browser* permet de gérer l'organisation du *Server*. Au sein de cet outil logiciel, l'organisation de l'espace de stockage est définie de façon arborescente, chaque nœud référence un espace particulier. Cet outil permet d'une part, de définir les arborescences et, d'autre part, de naviguer entre les différents nœuds. De plus, il permet, grâce à des mécanismes d'interaction, d'appeler les différents outils pour les appliquer sur un espace de stockage particulier.

### 2.3.2. Le langage ArchWare Hyper-code ADL [BMK 03a] [BMK 03b]

Tout comme l'ArchWare ADL, l'ArchWare Hyper-code ADL est un langage fondé sur le π-calcul typé d'ordre supérieur. Ce langage est conçu pour supporter l'évolution d'une architecture en cours d'exécution en s'appuyant sur la notion de « réflexion structurelle ».

La ***réflexion structurelle*** est définie comme la capacité d'une spécification en cours d'exécution à générer de nouveaux fragments de spécification et à les intégrer au sein de sa propre exécution.

Le cadre d'exécution propose un éditeur spécifique pour réaliser les modifications d'une architecture (*Hyper-code Editor*).

## 2.4. Les outils de l'environnement ArchWare

Nous présentons les outils de l'environnement en nous appuyant sur la logique de l'ingénierie logicielle centrée architecture (Cf. Chapitre 2, § 5.3 et Chapitre 4, § 2.3) qui peut être représentée par le schéma suivant :

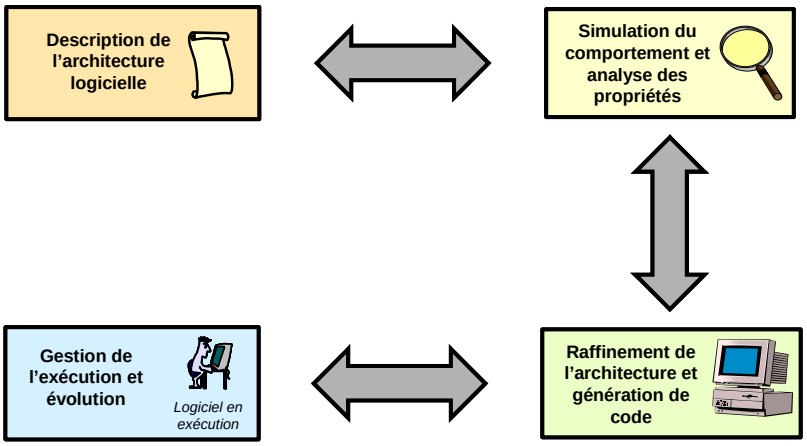



*Figure V.4. : Logique de l'ingénierie logicielle centrée architecture*

### 2.4.1. Les outils Visual modeller et Style editor

Le *visual modeller* permet la définition des styles et des architectures sous forme graphique à partir de profils UML, il s'appuie sur Objecteering. Le *style editor* est un simple éditeur de texte permettant la définition des styles sous forme textuelle.

### 2.4.2. L'outil Customiser [LBL 04]

Cet outil permet la personnalisation d'un environnement en s'appuyant sur la définition de styles architecturaux, il est implémenté en Java et en XSB Prolog. Cet outil s'appuie sur deux modules : le module « *style compiler* » et le module « *style instantiator* ». Le premier module compile des définitions de styles écrites en ASL, tandis que le second génère des instances d'abstractions (en ArchWare ADL). Cet outil réalise une analyse lexicale et syntaxique des styles.

Dans le cadre de notre cas d'étude, nous utilisons cet outil pour construire un environnement spécifique à notre domaine d'étude (Cf. Chapitre 3, § 4).

### 2.4.3. L'outil ADL Animator [APV 03]

Cet outil anime des descriptions d'architectures écrites en ArchWare ADL, il est implémenté en Java et en XSB Prolog. Cet outil permet de valider le comportement d'une architecture et, par conséquent, les styles sur lesquels sont basés cette architecture. Cet outil permet de vérifier certaines propriétés (propriétés comportementales). Toutefois, cet outil ne permet pas de réaliser des vérifications formelles.

Dans le cadre de notre cas d'étude, nous utilisons cet outil pour valider le comportement de nos architectures. Si, suite à l'animation, une erreur est détectée, une modification est réalisée soit en modifiant le style (il sera alors nécessaire de créer une nouvelle instance de l'architecture), soit en modifiant l'architecture.

### 2.4.4. L'outil AAL Model-Checker [BCD 04]

Cet outil se base sur des traces d'exécution des descriptions ArchWare ADL pour vérifier les propriétés comportementales décrites en ArchWare AAL, il est implémenté en SYNTAX, LOTOS NT, et en C. Le résultat de la vérification correspondant à la validité d'une propriété par rapport à la trace d'exécution utilisée, il est exprimé sous forme booléenne.

Dans le cadre de notre cas d'étude, nous utilisons cet outil pour vérifier les propriétés comportementales de nos architectures (Cf. Chapitre 4, § 3.3).

Cet outil n'est pas suffisant pour réaliser toutes les vérifications formelles. En effet, cet outil ne vérifie que les comportements des architectures.

### 2.4.5. L'outil AAL Theorem-Prover [LAO 03]

Cet outil vérifie des propriétés d'architectures exprimées en ArchWare AAL, il est implémenté en Java et XSB Prolog. Il s'appuie sur la logique d'ordre supérieur pour réaliser des vérifications de règles de typage, de construction des expressions, etc. En d'autres termes, cet outil vérifie que les expressions définies sont consistantes, correctes et complètes.

Dans le cadre de notre cas d'étude, nous utilisons cet outil pour vérifier les propriétés structurelles de nos architectures (Cf. Chapitre 4, § 3.3).

### 2.4.6. L'outil ARL Refiner [MeO 04]

Cet outil se base sur le langage ArchWare ARL pour construire (par raffinement) une architecture, il est implémenté sous MAUDE. Il s'appuie sur les règles de raffinement pour générer, de façon formelle, une nouvelle architecture.

Dans le cadre de notre cas d'étude, nous utilisons cet outil pour implémenter les mécanismes de raffinement par partition et par réécriture (Cf. Chapitre 1, § 3.2 et Chapitre 3, § 5).

### 2.4.7. L'outil Code Synthesiser [BMK 04]

Cet outil s'appuie sur des règles de transformation pour générer la description d'une architecture dans un langage cible (exécutable).

Dans le cadre de notre cas d'étude, nous utilisons cet outil pour générer les programmes qui sont interprétés par le moteur de workflow (Cf. Chapitre 3, § 5.2).

# 3. Le cas d'étude « Agile Integrated Process Systems » [BBD 02]

Ayant présenté l'environnement ArchWare, c'est-à-dire les langages et les outils développés dans le cadre de ce projet, nous allons aborder la problématique relative à notre cas d'étude.

## 3.1. Description de la problématique

Les entreprises de type PMI/PME sont de plus en plus conscientes que la maîtrise de leur système d'information est un élément clé en terme de performance industrielle et un élément différentiateur en terme de compétitivité. Cette maîtrise passe par la synchronisation des processus de réalisation s'appuyant sur un ensemble d'applications logicielles hétérogènes. L'objectif de ce cas d'étude est de montrer que les différents concepts, développés dans le cadre des chapitres précédents, apporte une réponse à cette problématique.

Dans le chapitre 1, nous avons montré que cette réalité opérationnelle doit être appréhendée à travers un point de vue « métier » et un point de vue « donnée ». De plus, dans le cadre du chapitre 2, nous avons identifié des solutions technologiques (les plates-formes EAI) capables de prendre en compte la synchronisation des processus de réalisation au sein d'une entreprise. Toutefois, ces solutions ne permettent pas de modéliser l'entreprise dans sa globalité.

C'est pourquoi nous avons proposé un cadre de modélisation dans le chapitre 3 qui représente le pilotage des processus, à travers les points de vue « métier » et « donnée » par des architectures industrielles et logicielles. Dans le chapitre 4, nous nous appuyons sur le concept de style architectural pour formaliser ces architectures. Nous allons donc maintenant illustrer ces différents concepts à travers un cas d'application développé grâce à l'environnement proposé par le projet ArchWare.

## 3.2. Le cas d'application

Ce cas d'application vise à piloter la fabrication d'un article spécifique, en l'occurrence un axe, en s'appuyant sur un ensemble d'applications logicielles dédiées à des entreprises de type PMI/PME. Dans un premier temps, nous décrivons le processus lié à la réalisation de cet axe. Nous présentons ensuite les applications logicielles utilisées pour supporter ces opérations de production.

### 3.2.1. Description du processus de réalisation

Dans notre cas d'application, nous considérons une entreprise qui a pour objectif de fournir sur le marché des produits spécifiques (axes). Pour répondre aux besoins de ses clients, l'entreprise décide d'intégrer une opération de traitement thermique dans la gamme de fabrication de ses produits pour augmenter leur qualité. Cependant, cette entreprise ne dispose pas des compétences nécessaires pour réaliser cette opération de traitement thermique. Il est donc envisagé de sous-traiter cette opération auprès de fournisseurs spécialisés.

La gamme de fabrication du produit est composée de cinq opérations :

- une opération de tournage qui réalise le diamètre extérieur du produit final,
- une opération de chanfreinage qui prépare le diamètre pour effectuer l'opération d'assemblage,
- une opération de traitement thermique qui modifie les propriétés physiques de l'axe,
- une opération de tournage qui finalise l'usinage du produit,
- une opération d'assemblage qui ajoute une bague plastique à l'axe.

Ces différentes opérations sont représentées au sein de la figure suivante :

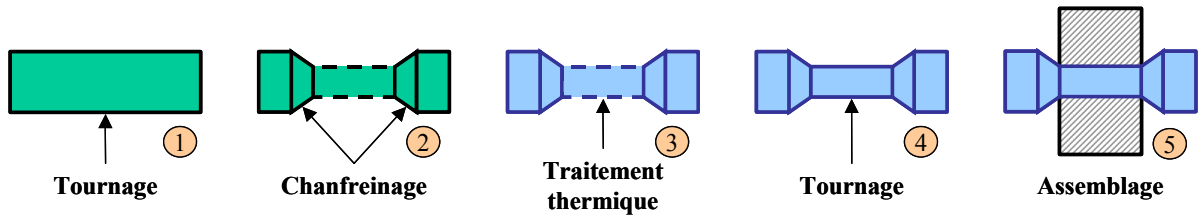



*Figure V.5. : La gamme de fabrication*

Le traitement thermique étant effectué par un sous-traitant, il est nécessaire de synchroniser sa réalisation avec les autres opérations de la gamme de fabrication.

### 3.2.2. Présentation des applications logicielles

L'entreprise et son sous-traitant s'appuient sur un ensemble d'applications logicielles hétérogènes (COTS – *Component Of The Shelf*) pour réaliser les opérations de production définies dans le paragraphe précédent. Chaque COTS dispose de fonctionnalités spécifiques permettant de répondre à un besoin particulier d'une entreprise. Les différents COTS étudiés dans cette application sont décrits par le tableau suivant :

| Nom du COTS | Type | Système d'exploitation | Base de données |
|---|---|---|---|
| ADC | Progiciel de devis | Windows 2000 | Access |
| Alpha3i | Progiciel de contrôle de production | Windows NT | Terminaux |
| Agilium | Moteur de workflow | Windows 2000 | XML |
| Infodev | Progiciel de gestion intégré | Linux | Propriétaire en mode ASP |
| Quasar CTRL réception | Progiciel de contrôle qualité | UNIX | Oracle |
| Quasar SPC | Progiciel de contrôle qualité | Windows 2000 | Propriétaire |
| Silverprod | Progiciel de gestion intégré | Windows 98 | Progress |

*Figure V.6. : Les COTS du cas d'application*

Comme le montre ce tableau, un moteur de workflow (Agilium) coordonne le fonctionnement de ces différentes applications. Ce logiciel intègre l'ensemble des concepts du BPM – *Business Process Management* – (Cf. Chapitre 2, § 4) et il permet de modifier l'enchaînement des activités en cours d'exécution.

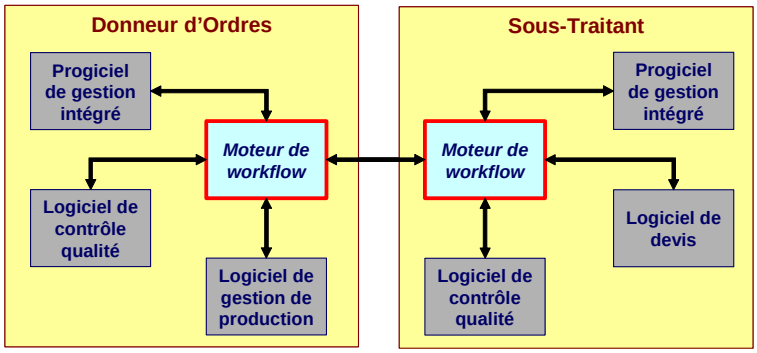



*Figure V.7. : L'architecture logicielle de notre cas d'étude*

Nous montrons donc, dans ce chapitre, comment intégrer le pilotage des processus au sein de ce type de logiciel.

## 4. Présentation des résultats [BBD 03c]

Ayant présenté l'étude de cas sur laquelle nous nous appuyons pour illustrer les concepts présentés dans les chapitres précédents, nous allons maintenant fournir une synthèse des résultats au sein de l'environnement ArchWare.

Dans un premier temps, nous décrivons les environnements qui ont été personnalisés à partir d'un environnement de base fournit par le projet. Nous présentons ensuite les styles de notre cas d'étude. Enfin, nous montrons comment les architectures évoluent au sein de l'environnement ArchWare.

## 4.1. Définition d'un environnement

Le projet ArchWare propose un environnement de base (*ArchWare Core*) permettant de construire des environnements adaptés à un domaine spécifique. Par conséquent, il est nécessaire de distinguer le processus d'ingénierie qui permet de construire l'environnement dédié à un domaine, du processus d'exploitation.

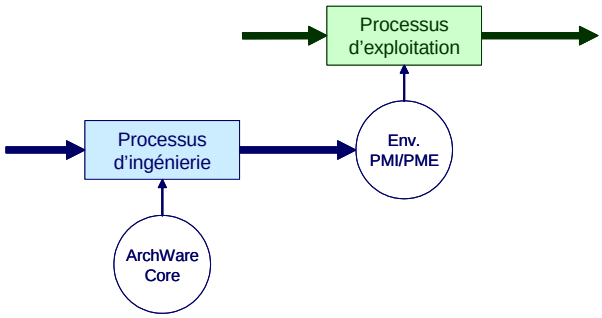



*Figure V.8. : Les environnements du projet ArchWare*

Le processus d'ingénierie permet de développer différents environnements d'exploitation.

Dans notre cas d'étude, nous avons identifié deux environnements :

- environnement dédié aux plates-formes BPM – *Business Process Management*, cet environnement formalise l'ensemble des éléments architecturaux nécessaires à la construction des architectures industrielles et logicielles,
- environnement dédié aux entreprises de type PMI/PME, cet environnement permet la construction des architectures industrielles et logicielles spécifiques à une entreprise.

Le processus d'exploitation intègre les processus de réalisation d'une entreprise. Dans notre cas d'étude, nous nous intéressons au fonctionnement d'une entreprise faisant appel à des activités de sous-traitance. Pour ce faire, il est nécessaire de construire un environnement adapté à chacune des entreprises. En conséquence, il est nécessaire de construire un environnement adapté pour chacune des entreprises.

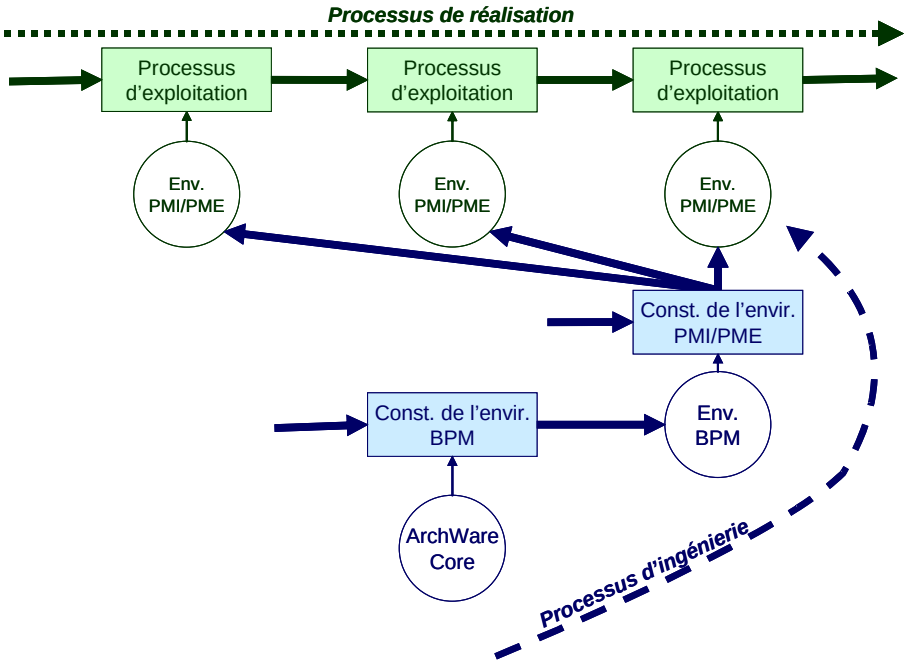



*Figure V.9. : Les environnements du cas d'application*

## 4.2. Organisation des environnements

Ce paragraphe présente l'arborescence des environnements au sein du navigateur *Tower Browser*. La construction d'un environnement (à partir d'un environnement donné) est réalisée grâce à des mécanismes de raffinement.


*Figure V.10. : Organisation des environnements*

Nous allons maintenant décrire de façon plus détaillée l'arborescence propre aux environnements BPM et PMI/PME.

### 4.2.1. L'environnement BPM

L'environnement BPM permet de construire les éléments architecturaux spécifiques aux plates-formes BPM, c'est-à-dire les éléments permettant de construire les architectures industrielles et logicielles. Rappelons que les éléments d'une architecture industrielle sont (Cf. Chapitre 3, § 4.1.1 et § 4.1.2) :

- le composant « *activité* »,
- le composant « *chorégraphe* »,
- le composant « *composant industriel* »,
- le composant « *système de performance* »,
- le composant « *système de gestion de ressources* »,
- le connecteur « *assemblage par ports* »,
- le connecteur « *relation d'échange* »,
- le connecteur « *coopération* »,
- le connecteur « *délégation* ».

De même, les éléments d'une architecture logicielle sont (Cf. Chapitre 3, § 4.2.1 et § 4.2.2) :

- le composant « *application* »,
- le composant « *workflow* »,
- le connecteur « *communication* ».

Cet environnement peut être partitionné de la façon suivante :

- *Visual modeller* : définition des besoins spécifiques aux plates-formes BPM par des diagrammes UML,
- *Style editor* : définition des styles propres à cet environnement,
- *Analysis tools* : vérification des propriétés définies par les styles,
- *Customiser* : personnalisation de l'environnement ArchWare Core pour construire l'environnement BPM,
- *Animator* : validation des comportements définis dans les styles.

L'organisation de cet environnement est représentée par le schéma suivant :

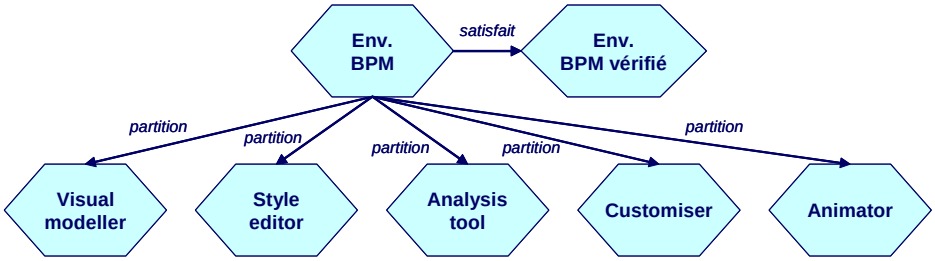



*Figure V.11. : L'environnement BPM*

### 4.2.2. L'environnement PMI/PME

L'environnement PMI/PME spécialise l'environnement BPM pour intégrer l'ensemble des caractéristiques propres aux entreprises manufacturières. Le processus d'exploitation défini au sein de cet environnement doit permettre de piloter les processus de réalisation d'une entreprise. En conséquence, cet environnement permet de construire les architectures industrielles en s'appuyant sur les styles suivants (Cf. Chapitre 3, § 4.1.3.2 et § 4.1.3.3) :

- le style « *inter-entreprise* »,
- le style « *intra-entreprise* ».

De même, cet environnement doit permettre la conception d'architectures logicielles (Cf. Chapitre 4, § 2.3) en s'appuyant sur le style « *architecture EAI* » (Cf. Chapitre 3, § 4.2.3.1).

En conséquence, cet environnement doit permettre la définition, l'instanciation et le raffinement des architectures, il peut être partitionné de la façon suivante :

- *Visual modeller* : définition des besoins spécifiques aux architectures industrielles et logicielles,
- *Style editor* : définition des styles « *inter-entreprise* », « *intra-entreprise* » et « *architecture EAI* »,
- *Analysis tools* : vérification des propriétés définies par les styles,
- *Customiser* : personnalisation de l'environnement BPM pour construire l'environnement PMI/PME,
- *Animator* : validation des comportements définis dans les styles,
- *Refiner* : raffinement des architectures,
- *Code synthesiser* : génération du code des applications.

L'organisation de cet environnement est représentée par le schéma suivant :

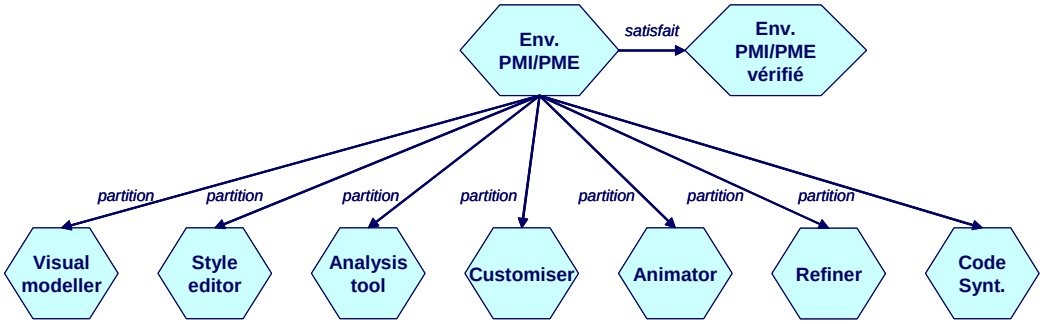



*Figure V.12. : L'environnement de conception PMI/PME*

De plus, cet environnement doit intégrer les applications logicielles et le moteur de workflow pour exécuter et faire évoluer les architectures industrielles et logicielles :

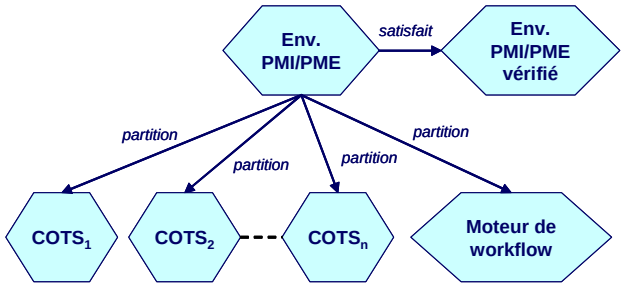



*Figure V.13. : L'environnement d'exécution PMI/PME*

## 4.3. Les styles de notre cas d'étude

Ce paragraphe présente les styles « patrons » d'architectures qui sont utilisés pour construire les architectures industrielles et logicielles représentant respectivement les points de vue « métier » et « donnée ». Dans un premier temps, nous décrivons les principaux styles développés pour notre cas d'étude et nous montrons ensuite l'organisation de ces styles.

### 4.3.1. Description des styles

Le schéma suivant rappelle les interactions entre les différentes architectures de notre cas d'étude (Cf. Chapitre 3, § 5).

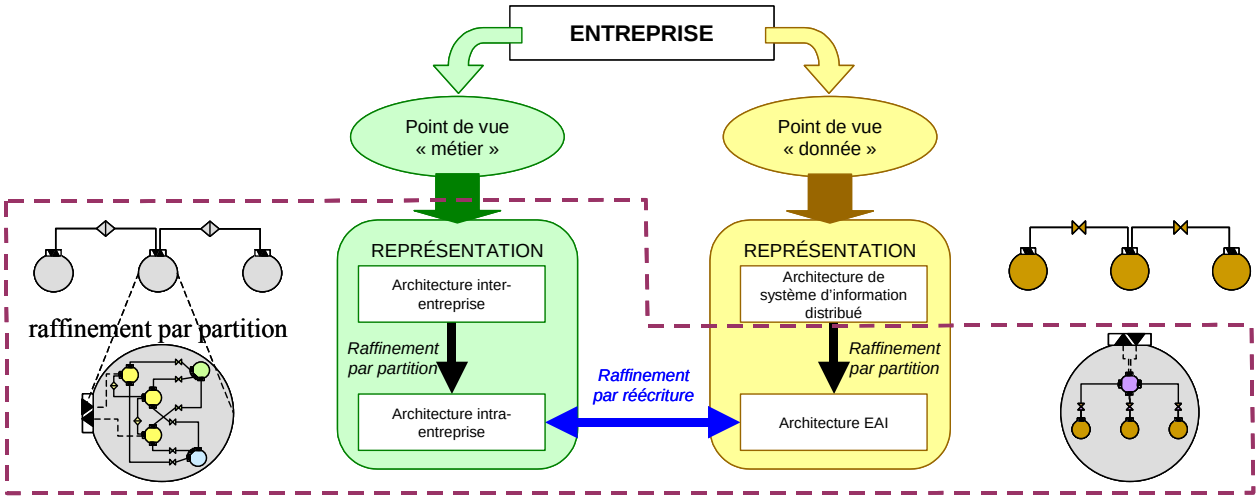



*Figure V.14. : Les interactions entre les architectures de notre cas d'étude*

Nous rappelons que, dans le cadre de nos travaux, nous ne nous sommes pas intéressés aux interactions qui existent entre l'architecture de système d'information distribué et les autres architectures.

L'utilisation de styles architecturaux nous permet de construire des « patrons » d'architectures formalisant les concepts relatifs à nos deux points de vue. Ces « patrons » sont utilisés pour construire les architectures de notre cas d'étude.

Nous montrons dans les paragraphes suivants les principaux « patrons » définis pour notre cas d'étude.

#### *4.3.1.1 Style « patron » inter-entreprise*

Ce style formalise les interactions d'une entreprise avec son environnement. Rappelons que ces interactions sont limitées aux flux de production, ce qui réduit l'environnement de l'entreprise à ses clients et fournisseurs (Cf. Chapitre 3, § 4.1.3.3). Ce style est composé de composants « composant industriel » représentant le fonctionnement d'une entreprise par un processus global. Les connecteurs entre les composants industriels (les entreprises) sont des connecteurs supportant des mécanismes mettant en œuvre une relation d'échange entre deux entreprises.

Dans notre cas d'étude, nous identifions deux types de composants industriels et un connecteur :

- un composant « donneur d'ordres » (*contractor*),
- un composant « sous-traitant » (*supplier*),
- un connecteur « négociateur » (*mediator*).

Ainsi, le style architectural inter-entreprise est une composition des styles formalisant les trois éléments architecturaux précédents.

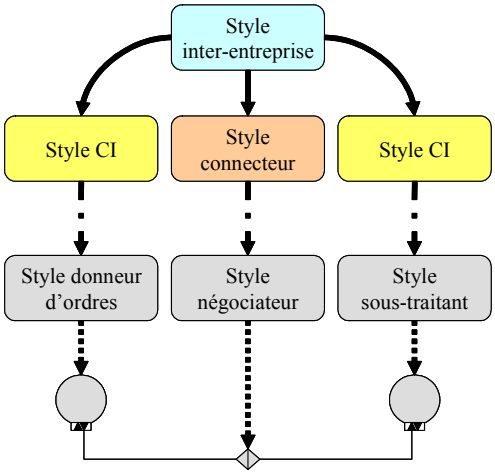



*Figure V.15. : Les styles pour la formalisation de l'architecture inter-entreprise*

La suite de ce paragraphe présente de manière synthétique ces différents styles, elle décrit le style donneur d'ordres, le style sous-traitant, le style connecteur et le style inter-entreprise.

```
Contractor is style extending IC where {
    styles { -- Définition des ports -- }
    }
    constructors {
        Contractor is constructor(contractor_name:String); {
            -- Définition du constructeur --
                computation {
                    -- Définition du comportement du composant --
                }
                configuration {
                    -- Définition de la configuration du composant --
                }
            }
        }
    }
},
```

```
        Supplier is style extending IC where {
            styles { -- Définition des ports --       }
            }
            constructors {
                Supplier is constructor(supplier_name:String); {
                    -- Définition du constructeur --
                        computation {
                            -- Définition du comportement du composant --
                        }
                        configuration {
                            -- Définition de la configuration du composant --
                        }
                    }
                }
            }
        },
```

```
        Mediator is style extending Connector where {
            styles { -- Définition des ports --      }
            }
            constructors {
                Mediator is constructor(); {
                    -- Définition du constructeur --
                        }
                        routing {
                            -- Définition du comportement du connecteur --
                        }
                        configuration {
                            -- Définition de la configuration du connecteur --
                        }
                    }
                }
            }
        }
```

```
Inter_Enterprises is style extending Component where {
    styles {
        -- CONTRACTOR STYLE --
        -- SUPPLIER STYLE --
        -- MEDIATOR STYLE --
    }
    constructors {
        Inter_Enterprises is constructor(); {
            -- Définition du constructeur --
                }
                configuration {
                    -- Définition de la configuration du composant --
                }
            }
        }
    }
}
```

Ces styles permettent d'instancier une architecture inter-entreprise spécifique. Le diagramme suivant présente une architecture composée d'un donneur d'ordres (*C1*) et de deux sous-

traitants (*S1*, *S2*), ce diagramme est réalisé à partir du *Visual modeller* qui intégre des profils symbolisant les composants et les connecteurs.

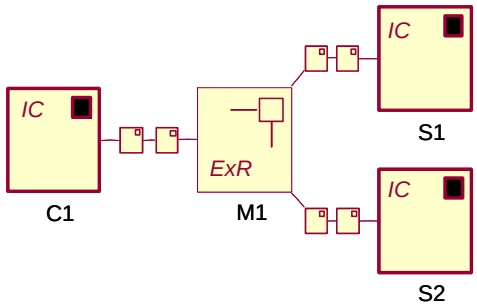



*Figure V.16. : Une architecture inter-entreprise*

#### *4.3.1.2 Style « patron » intra-entreprise*

Ce style formalise la coordination des processus de réalisation d'une entreprise (Cf. Chapitre 3, § 4.1.3.2). Ce style est composé de composants « composant industriel », d'un composant « système de performance » et d'un composant « système de gestion de ressources ». De plus, ce style comporte des connecteurs permettant de lier deux composants industriels, des connecteurs permettant de lier un composant industriel et le système de gestion de performance et des connecteurs permettant de lier un composant industriel et le système de gestion de ressources.

Dans notre cas d'étude, nous identifions les styles suivants :

- un composant « composant industriel » (*IC*),
- un composant « système de performance » (*PS*),
- un composant « système de gestion de ressources » (*RMS*),
- un connecteur « relation d'échange » (*ExR*),
- un connecteur « coopération » (*Coop*).

Le style architectural intra-entreprise est une composition des styles précédents.

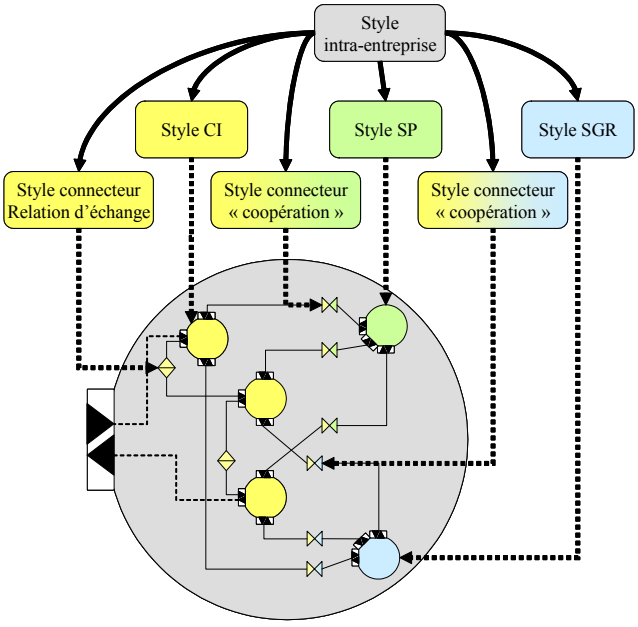



*Figure V.17. : Les styles pour la formalisation de l'architecture intra-entreprise*

La suite de ce paragraphe présente de manière synthétique le style entreprise.

```
Enterprise is style extending Component where {
    styles {
        -- IC STYLE --
        -- PS STYLE --
        -- RMS STYLE --
        -- ExR STYLE --
        -- Coop STYLE --
    }
constructors {
        Enterprise is constructor( numIC:Natural ); {
            -- Définition du constructeur --
                }
                configuration {
                    -- Définition de la configuration du composant --
                }
            }
        }
    }
    constraints {
        -- Définition des contraintes du style --
        }
    }
    analyses {
        -- Définition des analyses du style --
    }
}
```

Ces styles permettent d'instancier une architecture intra-entreprise spécifique. Le diagramme suivant présente une architecture composée de deux composants industriels (*IC1*, *IC2*) et d'un système de performance (*PS*) et d'un système de gestion de ressources (*RMS*).

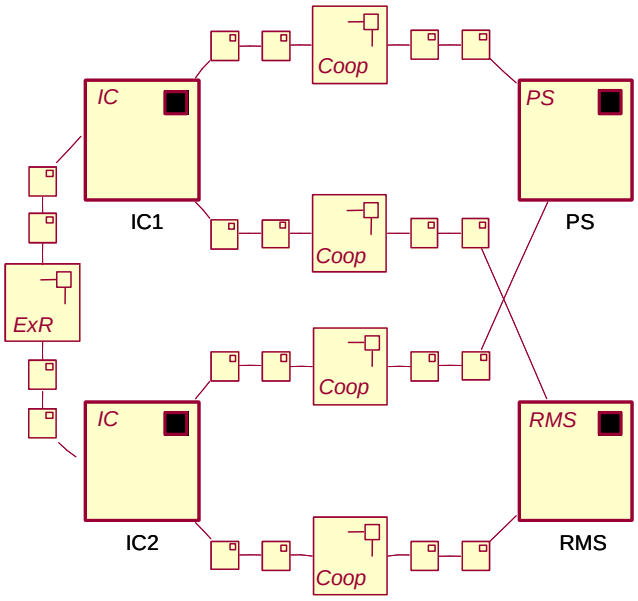



*Figure V.18. : Une architecture intra-entreprise*

#### *4.3.1.3 Style « patron » EAI*

Ce style formalise la coordination d'un moteur de workflow et d'un ensemble d'applications logicielles (Cf. Chapitre 3, § 4.2.3.1). Ce style est composé de composants « application » et d'un composant « workflow ». Les connecteurs entre les composants « application » et le composant « workflow » sont des connecteurs supportant des mécanismes de communication.

Dans notre cas d'étude, nous identifions les styles suivants :

- un composant « application » (*COTS*),
- un composant « workflow » (*workflow*),
- un connecteur « relation d'échange » (*Com*).

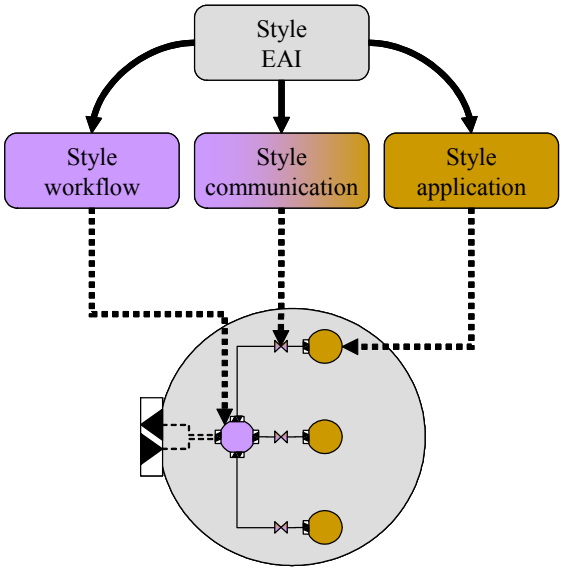



*Figure V.19. : Les styles pour la formalisation de l'architecture EAI*

Dans le paragraphe suivant, nous ne présentons de manière synthétique que le style EAI.

```
EAI is style extending Component where {
    styles {
        -- WORKFLOW STYLE --
        -- COTS STYLE --
        -- COM STYLE --
    }
    constructors {
        EAI is constructor (); {
            -- Définition du constructeur --
                }
                configuration {
                    -- Définition de la configuration du composant --
                }
            }
        }
    }
}
```

Ces styles permettent d'instancier une architecture EAI spécifique. Le diagramme suivant présente une architecture composée de trois composants applications (*C1*, *C2*, *C3*) et d'un moteur de workflow (*WF*).

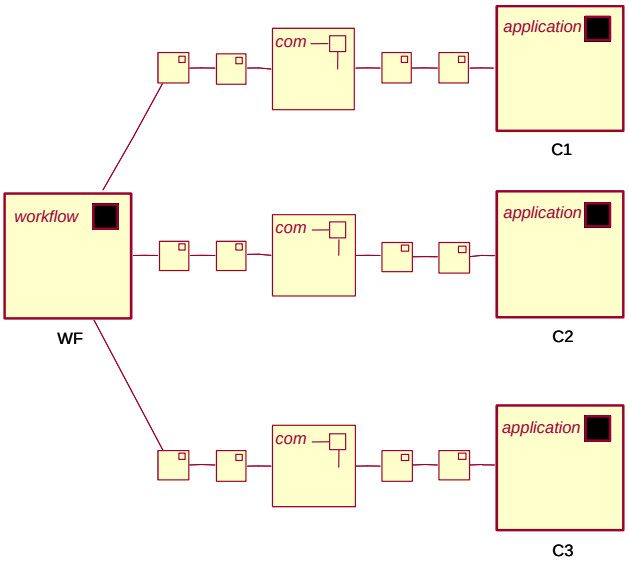



*Figure V.20. : Une architecture EAI*

### 4.3.2. L'organisation des styles

Ce paragraphe présente l'organisation des styles de notre cas d'étude. Rappelons que nous nous appuyons sur le modèle « composant / connecteur » pour formaliser nos architectures (Cf. Chapitre 3, § 2). Dans un premier temps, nous présentons les styles propres à l'environnement BPM.

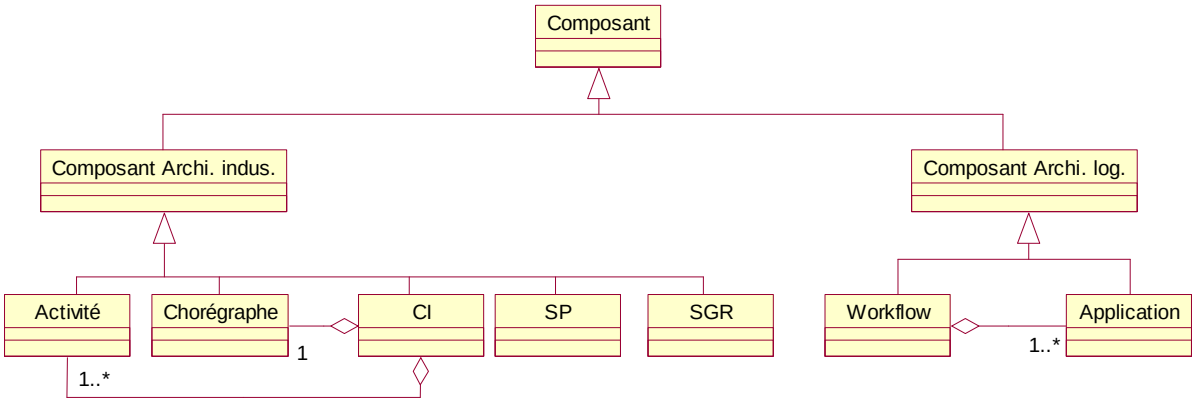



*Figure V.21. : Organisation des styles décrivant les composants*

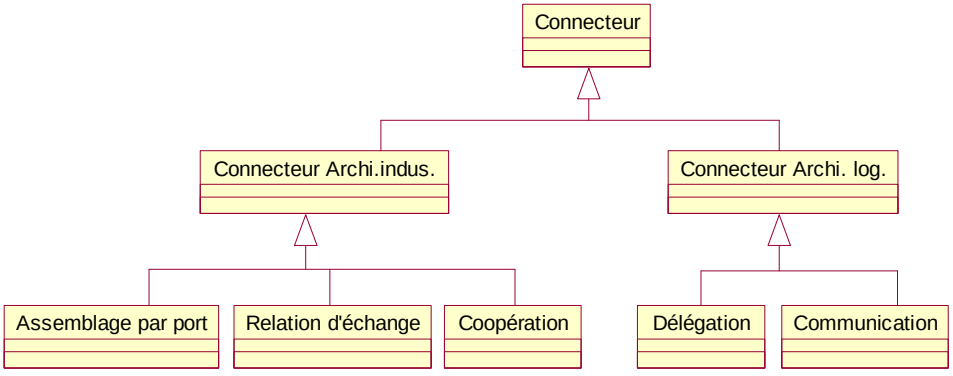



*Figure V.22. : Organisation des styles décrivant les connecteurs*

Le schéma suivant présente l'organisation des styles définis dans l'environnement PMI/PME.

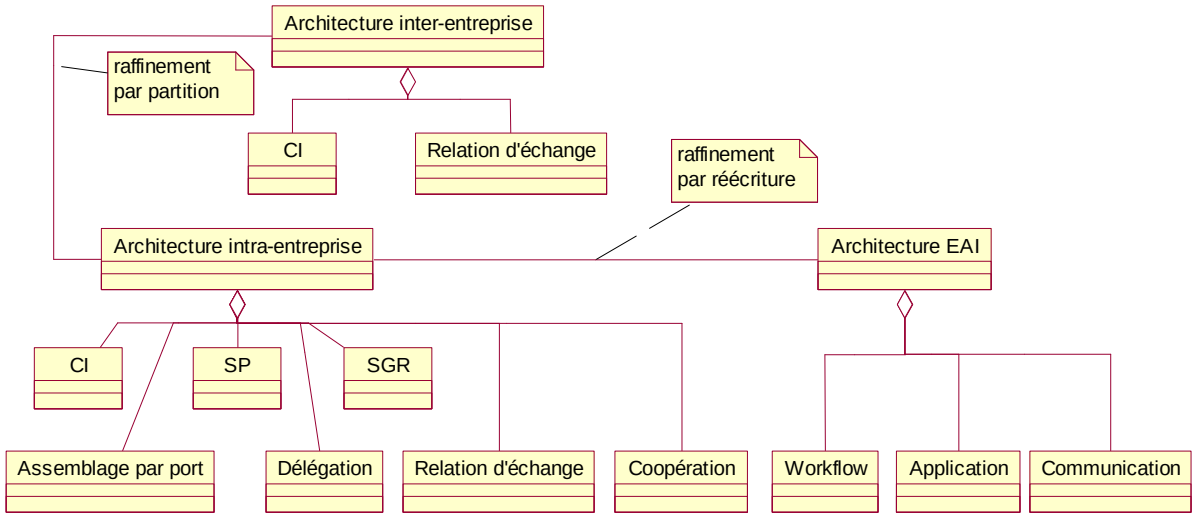



*Figure V.23. : Organisation de l'environnement PMI/PME*

Il est à noter que des styles « de base » (*foundation styles*), autres que les styles « *composant* » et « *connecteur* » sont fournis par l'environnement ArchWare mais ils ne sont pas représentés sur le schéma précédent. Ils permettent de construire les styles spécifiques à un domaine. Par exemple, dans notre cas d'application, nous nous appuyons sur le style de base « *Data Indirection* » (Cf. Annexe 4) dans le style « *Inter-enterprises* ».

## 4.4. L'utilisation des outils dans notre cas d'étude

Ce paragraphe montre la manière dont nous utilisons les différents outils de l'environnement ArchWare dans notre cas d'étude.

### 4.4.1. Animation de l'architecture

Cette phase d'animation est importante dans notre cas d'étude car les entreprises de type PMI/PME ne disposent généralement pas de « savoir-faire » en matière de représentations architecturales, de représentations et de vérifications formelles. Aussi, l'animation du fonctionnement des architectures permet de valider les interactions entre les différents éléments.

### 4.4.2. Raffinement de l'architecture

Cet outil implémente les mécanismes de raffinement nécessaires à notre cas d'étude.

Le raffinement par partition permet de construire une *architecture intra-entreprise* à partir d'un composant *entreprise* (CI). Le raffinement par réécriture transforme l'*architecture intra-entreprise* en une *architecture EAI*.

### 4.4.3. Génération de code

Cet outil permet de générer un fichier XML interprétable par le moteur de workflow qui décrit les architectures logicielles. Pour ce faire, le générateur de code s'appuie sur des règles de transformation « Langage ArchWare ADL – Langage cible » pour générer le fichier XML.

## 4.5. Evolution des architectures

Un des principaux objectifs du projet ArchWare et de ces travaux de thèse est de permettre l'évolution d'une architecture en cours d'exécution. Aussi, au sein de notre cas d'étude, nous pouvons identifier deux types d'évolution :

- une *évolution prévue* par la phase de conception,
- une *évolution non planifiée* lors de la phase de conception.

Dans ces deux cas, nous nous appuyons sur l'environnement ArchWare et sur les styles architecturaux construits pour réaliser ces évolutions.

### 4.5.1. Evolution prévue dans la phase de conception

Dans notre cas d'application, cette évolution offre la possibilité de créer un nouveau fournisseur pour absorber une éventuelle « surcharge » de production. Dans ce cas, lorsqu'une surcharge est détectée par une application logicielle (Silverprod ou Infodev), un événement est transmis au moteur de workflow (Agilium). Suite à cette réception, le moteur de workflow suspend l'exécution et répercute cet évènement à l'environnement ArchWare (cadre d'exécution). Cet événement est interprété et l'architecture inter-entreprise est modifiée suivant une logique définie par le style inter-entreprise. Suite à cette évolution, les mécanismes de raffinement permettent de construire dans un premier temps, une nouvelle architecture intra-entreprise (raffinement par partition) et, dans un second temps, une nouvelle architecture EAI (raffinement par réécriture). Enfin, un nouveau fichier XML est généré et est transmis au moteur de workflow qui l'intègre à la volée et reprend l'exécution.

### 4.5.2. Evolution non planifiée dans la phase de conception

Dans notre cas d'application, cette évolution se caractérise par le remplacement d'une application logicielle par une autre. En effet, lorsque l'entreprise décide de changer une application logicielle supportant une activité du processus de réalisation, l'architecte s'appuie sur l'environnement ArchWare pour modifier les mécanismes de raffinement par réécriture de façon à intégrer la nouvelle application lors du raffinement de l'architecture intra-entreprise. Lorsque cette modification est réalisée, une nouvelle architecture EAI est construite (application de nouvelles règles de raffinement par réécriture) et le fichier XML correspondant à cette architecture est généré. Ce fichier est transmis au moteur de workflow qui l'intègre à la volée.

# 5. Conclusion

Le cas d'étude présenté dans le cadre de ce chapitre nous a permis d'illustrer la définition d'architectures industrielles et logicielles pour appréhender la réalité opérationnelle à travers le point de vue « métier » et le point de vue « donnée ». Ce cas d'étude a été développé à partir des langages et des outils fournis par l'environnement ArchWare.

Notre cas d'étude a pour domaine d'application la formalisation du système d'information d'une entreprise manufacturière de type PMI/PME. Dans nos travaux, nous nous sommes surtout intéressés à l'intégration des applications logicielles supportant les activités des processus opérationnels d'une entreprise. Nous avons montré, par l'utilisation de langages formels centrés architecture, qu'il était possible de construire des représentations adaptées à la problématique des entreprises de type PMI/PME, capables de représenter les architectures industrielles et logicielles et dont les correspondances peuvent être exprimées grâce à des

mécanismes de raffinement (par partition ou par réécriture). Ces langages nous ont donc permis de modéliser des architectures selon nos deux points de vue.

De plus, nous avons montré que l'utilisation de ces différents langages nécessite la mise à disposition d'un ensemble d'outils capables de :

- définir de manière textuelle ou graphique des styles et des architectures,
- vérifier des propriétés structurelles et comportementales définies au sein de ces architectures,
- animer le comportement d'une architecture pour valider son adéquation aux besoins d'un domaine spécifique,
- raffiner les architectures pour passer d'une vision globale à une vision détaillée ou d'un point de vue à un autre,
- personnaliser un environnement pour un domaine d'applications particulières.

Enfin, nous avons présenté l'évolution de ces architectures, d'une manière prévue ou non planifiée. Cette évolution s'appuie sur le cadre d'exécution développé dans le cadre du projet ArchWare.

# Conclusion et perspectives

# Conclusion

Le thème abordé dans ce mémoire relève de la modélisation des architectures industrielles et logicielles évolutives pour formaliser les points de vue « métier » et « donnée » nécessaires à l'appréhension d'une réalité opérationnelle.

Dans le cadre de ces travaux, nous avons montré que le système d'information d'une entreprise manufacturière devait être structuré en :

- un système de gestion de connaissance,
- un système de gestion de processus,
- un système de gestion de ressources.

Nous nous appuyons ensuite sur le paradigme d'ambivalence pour appréhender la réalité opérationnelle d'une entreprise manufacturière à travers deux points de vue. De plus, nous considérons que la complexité des systèmes étudiés, la variété des représentations, la coordination de métiers différents (génie industriel et génie informatique) nécessitent une représentation du système à travers son architecture.

Dans un second temps, nous montrons que l'agilité d'une entreprise, c'est-à-dire sa capacité à s'adapter, voire anticiper les évolutions de son environnement, nécessite une organisation basée sur une approche processus. Le nombre de plus en plus important d'applications logicielles utilisées dans les entreprises pour supporter les activités d'un processus, nécessite des solutions technologiques capables de masquer l'hétérogénéité de ces applications aux différents acteurs de l'entreprise. Les plates-formes EAI offrent une solution à cette problématique, toutefois, nous avons montré qu'il n'existait pas à l'heure actuelle de solutions capables de fournir un cadre de modélisation permettant de représenter l'entreprise dans sa globalité. Un des objectifs de ces travaux a donc été de proposer un tel cadre de modélisation intégré dans un environnement logiciel centré architecture développé dans le cadre d'un projet européen.

Ce cadre de modélisation s'appuie d'une part, sur un modèle d'architecture « composant / connecteur » et, d'autre part, sur le modèle de pilotage de processus proposé par le modèle OLYMPIOS. Ce cadre propose une formalisation des architectures industrielles et logicielles à travers une vision globale et détaillée. De plus, il définit les mécanismes de raffinement par partition pour transformer une représentation globale en une représentation détaillée et les mécanismes de raffinement par réécriture pour transformer les représentations entre les deux points de vue. Enfin, nous avons montré que l'intégration du modèle de pilotage dans une modélisation d'architecture nous permet de construire des architectures évolutives.

Nous avons montré que la définition des architectures, qu'elles soient industrielles ou logicielles, est facilitée par l'utilisation de styles architecturaux. En effet, ces styles permettent de formaliser d'une part, des « patrons » d'architectures et, d'autre part, les propriétés topologiques, comportementales et d'attributs spécifiques à un domaine. La définition de ces styles est réalisée à partir d'un langage formel développé dans le cadre du projet européen ArchWare.

Enfin, nous avons illustré l'application de l'ensemble de ces concepts en nous appuyant sur un cadre d'exécution, des langages formels et des outils spécifiques développés au sein du projet

ArchWare. Cette application concerne plus particulièrement les entreprises de type PMI/PME, car nous considérons que ce type d'entreprise dispose d'un ensemble de solutions logicielles qu'il est nécessaire d'intégrer.

De plus, nous avons montré que cet environnement nous permet de faire évoluer les architectures (industrielles et logicielles). Ces évolutions concernent des modifications prévues lors de la phase de conception ou autorise des changements non planifiés par un architecte.

Ce travail s'est appuyé sur le modèle OLYMPIOS et sur différents résultats fournis par le projet ArchWare pour proposer un modèle des processus « métier » d'une entreprise mis en œuvre sur une plate-forme EAI en vue de leur pilotage. Il a été concrétisé par le développement du cas d'étude « *Agile Integrated Process Systems* » du projet ArchWare (*ArchWare WorkPackage 7*) [BBD 02].

# Perspectives

Le cadre de modélisation à la base de nos travaux est le modèle OLYMPIOS. Ce modèle appréhende le système d'information à travers le paradigme développé dans le cadre de cette thèse. De plus, il fournit l'ensemble des concepts nécessaires à la formalisation des processus et à leur pilotage. Dans le cadre de ces travaux, nous utilisons différents langages formels centrés architecture pour appréhender ces concepts. Nous définissons également une démarche, basée sur la démarche d'ingénierie centrée architecture, pour appréhender une problématique donnée. Enfin, nous utilisons un ensemble d'outils logiciels développés dans le cadre du projet ArchWare pour supporter les modèles, les langages et la démarche. Nous pouvons donc considérer que ce travail définit la base d'une transformation du modèle OLYMPIOS en une méthode. Pour ce faire, il est nécessaire d'enrichir l'ensemble des styles « patrons » d'architectures pour formaliser de façon plus détaillée le système de performance et le système de gestion de ressources.

Des travaux sont actuellement en cours pour formaliser le système d'indicateurs de performance [Cli 04]. Il serait donc intéressant de nous appuyer sur les résultats obtenus pour formaliser des styles « patrons » d'architectures capables d'appréhender le système de performance. D'autre part, d'autres travaux concernent la définition d'un cadre de référence pour la conception et l'exploitation d'un logiciel de gestion intégré de type ERP [Dar 04], nous pensons que les résultats proposés dans le cadre de cette étude peuvent servir de base à la définition des styles « patrons » d'architectures du système de gestion de ressources.

Dans le cadre de ces travaux, nous avons limité notre champ d'étude à l'échange du flux de production entre entreprises. Or, nous avons montré que le fait de considérer l'entreprise comme un système nécessite de prendre en compte des facteurs de production tels que les flux de travail, d'énergie, de capitaux et de savoir. Cette étude doit donc être complétée pour formaliser les connecteurs capables de lier l'entreprise aux différents marchés au sein desquels elle évolue. Cette extension doit permettre de formaliser de façon complète les réseaux d'entreprises.

Une fois cette extension réalisée, il sera nécessaire de formaliser les mécanismes de raffinement par réécriture permettant de transformer une architecture d'entreprise en une architecture de systèmes d'information distribués. Il faudra, de ce fait, également préciser les

mécanismes de raffinement par partition entre l'architecture de systèmes d'information distribués et l'architecture EAI.

Dans le cadre de nos travaux, nous nous sommes limités à l'utilisation des connecteurs supportant une relation d'échange pour formaliser les interactions entre les composants industriels. Lors de la présentation des mécanismes d'assemblage de ces composants, nous avons montré qu'il existait également des mécanismes d'assemblage par les ports et des mécanismes de délégation. Une étude de ces différents mécanismes est donc nécessaire pour compléter la formalisation des architectures intra-entreprise. Il est à noter que des travaux, portant sur la mobilité des comportements au sein d'une architecture, sont actuellement en cours au sein du laboratoire. Ces travaux devraient servir de base à la définition des connecteurs supportant les mécanismes de délégation.

Enfin, nous avons validé nos travaux sur un cas d'étude au sein d'un projet de recherche européen. Il nous reste donc à étudier le transfert de ces résultats dans le monde de l'édition des logiciels. C'est pourquoi, nous avons commencé un travail avec un éditeur de logiciels (Agilium), un centre européen de ressources en technologies (THESAME – Mécatronique et Management) et l'association des éditeurs rhône-alpin (Club ERA) pour mettre en œuvre une démarche de transfert de technologie.

# Références bibliographiques

# Références bibliographiques


[AAG 93] G. Abowd, R. Allen et D. Garlan, « *Using Style to Give Meaning to Software Architecture* ». SIGSOFT'93:Foundation Software Eng., ACM, New York, 1993.

[AAG 95] G. Abowd, R. Allen et D. Garlan, « *Formalizing style to understand descriptions of software architecture* ». ACM transaction on Software Engineering and Methodology, vol. 4, pages 319-364, octobre 1995.

[AAR 03] ArchWare Annual Report, « *Annual Report n°1 – Project Achievements in Year 1 (2002)* ». ArchWare European RTD Project IST-2001-32360, Deliverable D0.4.1, février 2003.

[Abd 96] A. Abd-Allah, « *Composing Heterogeneous Software Architecture* ». Thèse. Center for Software Engineering, University of Southern California, 1996.

[AGM 02] I. Alloui, H. Garavel, R. Mateescu et F. Oquendo, « *The ArchWare Architecture Analysis Language* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D3.1b, décembre 2002.

[AJP 00] L. Avignon, D. Joguet et P. Pezziardi, « *Intégration d'applications – l'EAI au cœur du e-business* », Eyrolles, 2000.

[AlG 94] R. Allen et D. Garlan, « *Formalizing Architectural Connection* ». 16th International Conference Software Engineering, IEEE Computer Soc. Press, Los Alamitos, Californie, pages 71-80, 1994.

[AlG 97] R. Allen et D. Garlan, « *A Formal Basis for Architectural Connection* ». ACM Transactions on Software Engineering and Methodology, 6(3), juillet 97.

[AnC 98] T. Andro et J.-M. Chauvet, « *Objets métiers* », édition Eyrolles, 1998.

[Ant 66] R. Anthony, « *Planning and Control Systems: A framework for analysis* », Harvard University Press, 1966.

[APV 03] S. Azzaiez, F. Pourraz, H. Verjus et F. Oquendo, « *Final ArchWare Architecture Animator – Release 1* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D2.2b, décembre 2003.

[Arc 02] Projet « *ArchWare – Architecting Evolvable Software* ». ARCHWARE European RTD Project IST-2001-32360. www.arch-ware.org. 2002-2004.

[Arc 91] B. Archiméde, « *Conception d'une architecture réactive distribuée et hiérarchisée pour le pilotage des systèmes de production* », Thèse de doctorat de l'Université de Bordeaux I, 1991.

[Ark 02] A. Arkin, « *Business Process Modeling Language* », Version 1.0, http://bpmi.org, novembre 2002.

[BBD 02] L. Blanc dit Jolicoeur, C. Braesch, R. Dindeleux, S. Gaspard, D. Le Berre, F. Leymonerie, A. Montaud, C. Chaudet, A. Haurat et F. Théroude, « *Final Specification of Business Case 1, Scenario and Initial Requirements* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D7.1b, décembre 2002.

[BBD 03a] L. Blanc dit Jolicoeur, C. Braesch, R. Dindeleux, S. Gaspard, D. Le Berre, F. Leymonerie et A. Montaud, « *Definition of Architectural Styles and Process Models for Business Case 1* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D7.2, juin 2003.

[BBD 03b] L. Blanc dit Jolicoeur, C. Braesch, R. Dindeleux et F. Théroude, « *An EAI system based on an Industrial Component model* », In proceedings of CESA 2003, IMACS Multiconference, Lille, 9-11 juillet 2003.

[BBD 03c] L. Blanc dit Jolicoeur, C. Braesch, R. Dindeleux, D. Le Berre, F. Leymonerie, A. Montaud et F. Pourraz, « *Definition of Architectural Models for Business Case 1* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D7.6, décembre 2003.

[BBD 03d] L. Blanc dit Jolicoeur, C. Braesch, R. Dindeleux et F. Theroude, « *Un modèle de processus pour la mise en œuvre d'une démarche EAI* », MOSIM'03, Toulouse, France, pages 134-139, 23-25 avril 2003.

[BBH 94] B. Boehm, P. Bose, E. Horowitz et M. J. Lee, « *Software requirements negotiation and renegociation aids: A theory-W based spiral approach* ». Proceedings of the 17th International Conference on Software Engineering, 1994.

[BCD 04] D. Bergamini, D. Champelovier, N. Descoubes, H. Garavel, R. Mateescu et W. Serwe, « *ArchWare Architecture Analysis Tool by Model-Checking* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D3.6b, juin 2004.

[BCK 99] L. Bass, P. Clements et R. Kazman, « *Software Architecture in Practice* ». Addison Wesley, ISBN 0-201-19930-0, 1999.

[BCO 00] T. Bolusset, C. Chaudet et F. Oquendo, « *Description d'architectures logicielles dynamiques : langage formel et applications industrielles* », Génie Logiciel, juin 2000.

[BCO 99] T. Bolusset, C. Chaudet et F. Oquendo, « *Description d'architectures logicielles dynamiques : langage formel et applications industrielles* », Proceedings of the 12th International Conference on Software and Systems Engineering and their Applications (ICSSEA'99), Paris, France, 8-10 décembre 1999.

[BEJ 96] P. Binns, M. Engelhart, M. Jackson et S. Vestal, « *Domain-Specific Software Architectures for Guidance, Navigation, and Control* ». International Journal of

Software Engineering and Knowledge Engineering, 1996.

[Ber 01] J.C. Bernadac, « *Etat de l'art de l'intégration* », Acte de conférence de 01 Séminaires, Réussir son projet EAI, 19 décembre 2001.

[Ber 97] L. Berrah., « *Une approche de l'évaluation de la performance industrielle : modèle d'indicateur et techniques floues pour un pilotage réactif* », Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 1997.

[Bev 95] J.-M. Beving, « *Généralités sur les systèmes* », La modélisation systémique en entreprise, Hermès, 1995.

[BFW 00] M. Beck, J. Finout et B. Westlake, « *Change Management and EAI* ». eAI Journal, pages 77-79, septembre 2000.

[BGL 02] M. Boullier, L. Geoffray, E. Le Gouzouguec et B. Masson, « *Le livre blanc du BPM – L'orchestration des processus métier* », Vistali SA, 54 pages, septembre 2002.

[BHB 95] C. Braesch, A. Haurat et J.-M. Beving, « *L'entreprise-système* », La modélisation systémique en entreprise, Hermès, 1995.

[BHM 92] C. Braesch, A. Haurat, Y. Meslans et J.-C. Moureau, « *Une approche de la génération de plans en robotique* », APII, vol. 26, n°1, 1992.

[Bin 91] Z. Binder, « *Conduite décentralisée coordonnée* », Modélisation, Simulation, Conduite des systèmes de production, Ecole d'été d'automatique de Grenoble, 9-13 septembre 1991.

[BiV 93] P. Binns et S. Vestal, « *Formal real-time architecture specification and analysis* ». 10th IEEE Workshop on Real-Time Operating Systems and Software, mai 1993.

[Bla 01] L. Blanc Dit Jolicoeur, « *Modélisation de processus par approche workflow* », rapport de DEA informatique de l'Université de Savoie, 2001.

[Bla 02] X. Blanc, « *Du profil UML au composant MDA* », Développeur Référence, Version 2.22, décembre 2002.

[BlB 01] L. Blanc Dit Jolicoeur et C. Braesch, « *Proposition de modélisation de processus pour la mise en œuvre de workflow* », Congrès international sur l'Ingénierie des systèmes et Nouvelles Technologies de l'Information et de la Communication (Nîmes'TIC 2001), Nîmes, pages 47-52, décembre 2001.

[BMK 03a] D. Balasubramaniam, R. Morrison, G. Kirby, S. Cîmpan et F. Oquendo, « *ArchWare ADL Reference Model* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D1.6, février 2003.

[BMK 03b] D. Balasubramaniam, R. Morrison, G. Kirby et K. Mickan, « *Integration of Hyprt-code and Structural Reflection in ArchWare ADL* ». ARCHWARE

European RTD Project IST-2001-32360. Deliverable D1.5, février 2003.

[BMK 04] D. Balasubramaniam, R. Morrison, G. Kirby, K. Mickan et S. Norcross, « *ArchWare Code Synthesiser* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D6.4a, mars 2004.

[BMR 96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad et M. Stal, « *Pattern Oriented Software Architecture: A System of Patterns* ». John Wiley & Sons, 1996.

[Boa 95] M. Boasson, « *The Artistry of Software Architecture* ». Guest editor's introduction, IEEE Software, 1995.

[Bol 04] T. Bolusset, « *β-SPACE : Raffinement de descriptions architecturales en machines abstraites de la méthode formelle B* », Thèse de Doctorat de l'Université de Savoie, 2004.

[Bon 98] L. Bongaerts, « *Intégration of scheduling and control in holonic manufacturing systems* », doctoraat in de toegepaste wetenschapen, Katholieke Universiteit Leuven, décembre 1998.

[Bor 02] F. Bordage, « *L'intégration d'applications* » dossier de Décision micro, 06 mai 2002.

[Bou 02] M. Boullier, L. Geoffray, E. Le Gouzouguec et B. Masson, « *Le livre blanc du BPM* ». http://www.vistali.com, 2002.

[BPE 03] BPEL4WS, « *Business Process Execution Language for Web Services* », Version 1.1, mai 2003.

[BPM 02] BPMI, « *Business Process Modeling Notation* », Version 0.9, http://bpmi.org, novembre 2002.

[Bra 02] C. Braesch, « *Le modèle OLYMPIOS* », Ecole de modélisation d'entreprise, Groupe « modélisation d'entreprise » du Groupement pour la Recherche en Productique, Albi-Carmaux, 2002.

[Bra 89] C. Braesch, « *Approche de modélisation du système de production d'une entreprise manufacturière* », Thèse de Doctorat de l'Université de Franche-Comté, 1989.

[BrB 04] C. Braesch et L. Blanc dit Jolicoeur, « *Vers une orchestration des applications informatiques* », EAI/BPM comment orchestrer vos processus dans votre système d'information, dossier n°74 du Pôle Productique Rhône-Alpes, pages 5-10, mars 2004.

[BrH 95] C. Braesch et A. Haurat, « *De la nécessité de maîtriser l'information* », La modélisation systémique en entreprise, Hermès, 1995.

[BTH 00] C. Braesch, F. Théroude et A. Haurat, « *OLYMPIOS : towards a component-*

*based enterprise modelling* », CEN Workshop, Berlin, 2000.

[CBK 95] P. Clements, L. Bass, R. Kazman et G. Abowd, « *Predicting software quality by architecture-level evaluation* ». Proceedings of the Fifth International Conference on Software Quality, Austin, Texas, octobre 1995.

[Cha 02] C. Chaudet, « *π-Space : langage et outils pour la description d'architectures évolutives à composants dynamiques. Formalisation d'architectures logicielles et industrielles* ». Thèse de Doctorat de l'Université de Savoie, 2002.

[Che 03] H. Chelli, « *Urbaniser l'entreprise et son système d'information* », Vuibert, 2003.

[CHL 89] A. Collongues, J. Hugues et B. Larache, « *MERISE – 1. Méthode de conception* », éditions Dunod informatique, 240 pages, 1989.

[CHL 99] H. Crespel, F. Horn, L. Leboucher, P. Potteau, J.-M. Sfez et G. Thiébault, « *Mémento technique n°14 – Bases de données et système d'information* ». Partie relative à la transformation du Système d'Information, 26 octobre 1999. http://www.rd.francetelecom.com/fr/conseil/mento14/intro.html

[ChO 00] C. Chaudet et F. Oquendo, « *A Formal Architecture Description Language Based on Process Algebra For Evolving Software Systems* », Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00), Grenoble, France, 11-15 septembre 2000.

[ChO 01] C. Chaudet et F. Oquendo, « *π-SPACE: Modeling Evolvable Distributed Software Architectures* ». Proceedings of International Conference PDPTA'01, Las Vegas, juin 2001.

[ChT 02] A. Chardonnet et D. Thibaudon, « *Le guide du PDCA de Deming* », éditions d'organisation, 368 pages, novembre 2002.

[CiM 96] P. Ciancarini, C. Mascolo, « *Analysing and Refining an Architectural Style* », 1996.

[Cli 04] V. Clivillé, « *Approche systémique et méthode multicritère pour la définition d'un système d'indicateurs de performance* », Thèse de Doctorat de l'Université de Savoie, 2004.

[CLL 99] V. Chapurlat, M. Larnac, E. Lamine. et J. Magnier, « *Definition of a formal analysis framework for existing enterprise modelling approaches* ». Actes de Annual conf. of ICIMS-NOE, Life cycle approaches to production systems: management, control, supervision (ASI), Louvain, Belgique, 1999.

[CNM 96] P. Coad, D. North et M. Mayfield, « *Object Models – Strategies, Patterns and Applications* », Yourdon Press Computing Series, 1996.

[COB 02] S. Cimpan, F. Oquendo, D. Balasubramaniam, G. Kirby et R. Morrison, « *The ArchWare ADL: Definition of the Textual Concrete Syntax* ». ArchWare

European RTD Project IST-2001-32360, Deliverable D1.2b, décembre 2002.

[CoS 93] L. Coglianese et R. Szymanski, « *DSSA-ADAGE: An Environment for Architecture-based Avionics Development* ». Proceedings of AGARD'93, mai 1993.

[Dar 04] F. Darras, « *Proposition d'un cadre de référence pour la conception et l'exploitation d'un progiciel de gestion intégré* », Thèse de Doctorat de l'Institut Polytechnique de Toulouse, 2004.

[Dav 74] D.-B. Davis, « *MIS: conceptual, foundation, structure and development* », Mc Graw Hill, 1974.

[DBW 91] D.M. Dilts, N.P. Boyd, et H.H. Whorms, « *The evolution of control architectures for automated manufacturing systems* », Journal of Manufacturing Systems 10, 1, 79-93, 1991.

[DeL 96] R. DeLine, « *Toward User-Defined Element Types and Architectural Styles* ». Proceedings of the Second International Software Architecture Workshop, pages 47-49. San Francisco, 1996.

[Dem 82] E.W. Deming, « *Quality, Productivity and Competitive Position* », The MIT Press, 1982.

[Den 03] A. Denguir, « *Etude des modèles d'entreprise pour la contribution à la définition d'un style architectural 'Génie Industriel' adapté à l'EAI* », rapport de DEA informatique de l'Université de Savoie, 2003

[Din 98] R. Dindeleux, « *PROPILOT, une contribution à la modélisation des processus industriels* ». Thèse de Doctorat de l'Université de Savoie. 1998.

[Evg 02] T. Evgeniou, « *Information Integration and Information Strategies for Adaptive Enterprises* », European Management Journal, volume 20, issue 5, pages 486-494, 2002.

[EVO 01] Groupe EVOLUTION, « *Entrepôt de données pour l'aide à la décision* », Ingénierie des Systèmes d'Information, Traité IC2, Hermès Science, 2001.

[EYR 03] N. Erasala, D. C. Yen et T. M. Rajkumar, « *Enterprise Application Integration in the electronic commerce world* », Computer Standards & Interfaces, volume 25, pages 69-82, 2003.

[FaL 99] R. Fannader, H. Leroux, « *UML Principes de modélisation* », Éditions DUNOD 286 pages, 1999.

[GAO 94] D. Garlan, R. Allen et J. Ockerbloom, « *Exploiting Style in Architectural Design Environments* ». SIGSOFT'94, ACM Press, New York, pages 179-185, décembre 1994.

[Gar 00] D. Garlan, « *Software Architecture: a Road Map* ». Proceedings of the

conference on The future of Software engineering, mai 2000.

[Gar 95] D. Garlan, « *What is Style ?* », Proceedings of Dagshtul Workshop on Software Architecture, 1995.

[GaS 93] D. Garlan et M. Shaw, « *Introduction to Software Architecture* ». Advances in Software Engineering and Knowledge Engineering. World Scientific Publishing Company, 1993.

[GHJ 95] E. Gamma, R. Helm, R. Johnson et J. Vlissides, « *Design Patterns, Elements of reusable Object-Oriented Software* », Addison-Wesley Publishing Company, 1995.

[Gia 91] V. Giard, « *La Gestion de Projet* », Paris, Economica, 1991.

[GMW 97] D. Garlan, R. Monroe et D. Wile, « *ACME: an Architectural Description Interchange Language* ». Proceedings of CASCON'97, pages 169-183, Toronto, novembre 1997.

[GRS 03] M Greenwood, I. Robertson, W. Seet, B. Snowdon, B. Warboys et B. Senior, « *Final ArchWare Repository* ». ArchWare European RTD Project IST-2001-32360, Deliverable D2.3b, décembre 2003.

[Gru 93] T.R. Gruber, « *Toward principles for the design of ontologies used for knowledge sharing, in Formal Ontology in Conceptual Analysis and Knowledge Representation* », Edité par N. Guarino, Kluwer Academic Publishers, 1993.

[GSO 92] D. Garlan, M. Shaw, C. Okasaki, C. Scott et R. Swonger, « *Experiences with a Course on Architectures for Software Systems* ». Proceedings of the 6th SEI Conference on Software Engineering Education, 1992.

[Hoa 85] C.A.R. Hoare, « *Communicating Sequential Processes* », Prentice Hall International Series in Computer Science, 1985.

[Hos 00] E. Hostachy, « *Le système d'information doit être centré sur l'EAI* », Informatiques Magazine, pages 40-44, mai 2000.

[ISO 00] Norme européenne NF EN ISO 9001 version 2000, « *Systèmes de management de la qualité – Exigences* », AFNOR, 2000.

[ITL 03] Z. Irani, M. Themistocleous et P.E.D. Love, « *The impact of enterprise application integration on information system lifecycles* », Information & Management, volume 41, pages 177-187, 2003.

[Joh 92] R.E. Johnson, « *Documenting frameworks using patterns* », Object-oriented programming systems, languages, and applications OOPSLA'92, ACM Press, Vancouver, 18-22 octobre 1992.

[JoP 01] P. Johannesson et E. Perjons, « *Design principles for process modelling in*

*enterprise application integration* », Information Systems, volume 26, pages 165-184, 2001.

[Kal 85] G. Kallel, « *Proposition d'une conduite hiérarchisée coordonnée (CODECO) pour un atelier de fabrication* », Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 1985.

[KlK 99] M. Klein et R. Kazman, « *Attribute-Based Architectural Styles* ». Technical Report CMU/SEI-99-TR-022. ESC-TR-99-022, octobre 1999.

[Koz 83] D. Kozen, « *Results on the propositional Mu-Calculus* », dans Theorotical Computer Science 27, pages 333-354, 1983.

[KTK 03] T. Kobayashi, M. Tamaki et N. Komoda, « *Business process integration as a solution to the implementation of supply chain management systems* », Information & Management, volume 40, pages 769-780, 2003.

[LAO 03] D. Le Berre, I. Alloui et F. Oquendo, « *Preliminary ArchWare Architecture Analysis Tool by Theorem Proving* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D3.5a, décembre 2003.

[LCO 01] F. Leymonerie, S. Cîmpan et F. Oquendo, « *Extension d'un langage de description architecturale pour la prise en compte des styles architecturaux: Application à J2EE* ». Proceedings of ICSSEA 14th International Conference on Software and Software Engineering and their Applications, Paris, décembre 2001.

[LCO 02] F. Leymonerie, S. Cîmpan et F. Oquendo, « *Etat de l'art sur les styles architecturaux : Classification et Comparaison des Langages de Description d'Architectures Logicielle* », Revue Génie Logiciel, N°62, septembre 2002.

[LeG 92] F. Le Gallou, « *Activités des systèmes* », Systémique, Théorie & Applications, Lavoisier, 1992.

[LeM 77] J.-L. Le Moigne, « *La théorie du système général, théorie de la modélisation* », Presses Universitaires de France, 1977 (3ème édition, 1990).

[LeM 90] J.-L. Le Moigne, « *La modélisation des systèmes complexes* », Afcet-systèmes, Dunod, 1990.

[Ley 04] F. Leymonerie, « *ASL : un langage et des outils pour les styles architecturaux. Contribution à la description d'architectures dynamiques* », Thèse de Doctorat de l'Université de Savoie, 2004.

[Lin 00] D. S. Linthincum, « *Application Servers and EAI* », eAI Journal, pages 74-80, juillet-août 2000.

[LKA 95] D.C. Luckham, J.J. Kenney, L.M. Augustin, J. Vera, D. Bryan et W. Mann, « *Specification and analysis of system architecture using Rapide* ». IEEE Transaction on Software Engineering, vol. 21, n°4, pages 336-355, April 1995.

[Lor 95] P. Lorino, « *Comptes et récits de la performance : essai sur le pilotage de l'entreprise* », les éditions d'Organisation, 288 pages, juin 1995.

[Man 99] J.E. Mann, « *Workflow and EAI* », eAI Journal, pages 49-53, septembre-octobre 1999.

[MaS 92] N. Maiden et A. Sutcliffe, « *Exploiting Reusable Specifications Through Analogy* », dans Communications of the ACM, volume 35, issue 4, pages 55-64, avril 1992.

[MaS 93] N. Maiden et A. Sutcliffe, « *The Domain Theory: Object System Definition* », Nature Report CU-93-OOA, 1993.

[MDE 95] J. Magee, N. Dulay, S. Eisenbach et J. Kramer, « *Specifying Distributed Software Architectures* ». Proceedings of the Fifth European Engineering Conference (ESEC'95), Barcelona, septembre 1995.

[Med 96] N. Medvidovic, « *ADLs and Dynamic Architecture Changes* », Proceedings of the Second International Software Architecture Workshop (ISAW-2), San Francisco, CA, pages 24-27, 14-15 octobre 1996.

[Meg 04] K. Megzari, « *REFINER : Environnement logiciel pour le raffinement d'architectures logicielles fondé sur une logique de réécriture*», Thèse de Doctorat de l'Université de Savoie, 2004.

[MeG 92] E. Mettala et M. H. Graham, « *The domain-specific software architecture program* ». Technical report CMU/SEI-92-SR-9, Carnegie Mellon Software Engineering Institute, juin 1992.

[Mei 00] J.P. Meinadier, « *Ingénierie et intégration des systèmes* », Édition Hermes, 543 pages, octobre 2000.

[Mel 72] J. Mélèse, « *L'analyse modulaire des systèmes* », Les Hommes et Techniques, 1972.

[Mel 79] J. Melèse, « *Approches systémiques des organisations, vers l'entreprise à complexité humaine* », Hommes et techniques, 1979 (Editions d'organisation, 1990).

[MeM 98] C.P. Menzel et R.J. Mayer, « *The IDEF Family of Languages* ». Handbook on architectures of information systems, Bernus P., Mertins K. et Schmidt G. ed., Berlin, Springer, 1998.

[MeO 04] K. Megzari et F. Oquendo « *ArchWare Architecture Refinement Tools* », ArchWare European RTD Project IST-2001-32360, Deliverable D6.3b, avril 2004.

[MES 97a] MESA International, « *Execution-Driven Manufacturing Management for Competitive Advantage* ». White Paper numéro 5, http://www.mesa.org, 1997.

[MES 97b] MESA International, « *MES Functionalities & MRP to MES Data Flow Possibilities* ». White Paper numéro 2, http://www.mesa.org, 1997.

[MeT 00] N. Medvidovic et R. Taylor, « *A Framework for Classifying and Comparing Architecture Description Languages* », *IEEE Transactions on Software Engineering*, 2000.

[MeT 97] N. Medvidovic et R. Taylor, « *A Classification and Comparison Framework for Software Architecture Description Languages* ». Technical Report UCI-ICS-97-02, Department of Information and Computer Science, University of California, Irvine, février 1997.

[Mil 89] R. Milner, « *Communication and Concurrency* », Prentice-Hall, 1989.

[Mil 99] R. Milner, « *Communicating and mobile systems: the π-calculus* », Cambridge university press, 1999.

[MKM 97] R.T. Monroe, A. Kompanek, R. Melton et D. Garlan, « *Architectural Styles, Design Patterns, and Objects* ». IEEE software volume 14 Number 1, pages 43-52, janvier/février 1997.

[MoG 96] R. Monroe, D. Garlan, « *Style-Based Reuse for Software Architectures* ». Proceedings of the 1996 International Conference on Software Reuse, avril 1996.

[Mon 01] R.T. Monroe, « *Capturing Software Architecture Design Expertise With Armani* ». Technical Report CMU-CS-98-163, 2001.

[Mor 77] E. Morin, « *La Méthode 1* » La nature de la Nature, Paris, Seuil, 1977.

[Mor 87] E. Morin, « *Contribution du MRT à l'approche systémique de l'entreprise* », Revue internationale de systémique, Vol 1, N°4, pages 377-394, 1987.

[MOR 96] N.Medvidovic, P. Oreizy, D.S Rosemblum et R. Taylor, « *Using Object-Oriented Typing to Support Architectural Design in the C2 style* ». Proceedings of ACM SIGSOFT'96: Fourth symposium on the foundations of software engineering (FSE4), pages 24-32, San Francisco, CA, octobre 1996.

[Mou 90] J.-C. Moureau, « *Génération de plans d'actions en robotique : complexité, spécificités et définition d'une approche propre à la robotique* », Université de Franche-Comté, 1990.

[MQR 95] M. Moriconi, X. Qian et R. Riemenschneider, « *Correct Architecture Refinement* ». IEEE Transactions on Software Engineering, Volume 21, Number 4, pages 356-372, 1995.

[Mul 97] P.A. Muller, « *Modélisation objet avec UML* », Edition Eyrolles, 421 pages, 1997.

[Mur 02] B. Mury, « *Comment formaliser ses processus* », Conférence Progection organisée par THESAME, Annecy, 7 février 2002.

[MyB 98] K.L. Myers et P.M. Berry. « *Workflow management systems: An AI perspective* », Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1998.

[NEC 92] D. Nanci, B. Espinasse, B. Cohen et H. Heckenroth, « *Ingénierie des systèmes d'information avec Merise, vers une deuxième génération* », éditions Sybex, 650 pages, 1992.

[OAC 02] F. Oquendo, I. Alloui, S. Cîmpan et H. Verjus, « *The ArchWare ADL: Definition of the Abstract Syntax and Formal Semantics* ». ARCHWARE European RTD Project IST-2001-32360. Deliverable D1.1b, décembre 2002.

[OMG 02a] OMG, « *UML Profile for Enterprise Distributed Object Computing Specification* », février 2002.

[OMG 02b] OMG, « *UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification* », septembre 2002.

[Oqu 03] F. Oquendo, « *Final Definition of the ArchWare. Architecture Refinement Language* ». ArchWare European RTD Project IST-2001-32360, Deliverable D6.1b, décembre 2003.

[Orl 75] J. Orlicky, « *Materiel Requirements Planning* », Mc Graw Hill, 1975.

[Par 01] S. Parpinelli, « *Concept, Standards et Enjeux* », Acte de conférence de 01 Séminaires, Réussir son projet EAI, 19 décembre 2001.

[Pau 87] B. Paulré, « *L'entreprise-système* », Revue internationale de systémique, Vol 1, N°4, pages 377-394, 1987.

[Pau 92] B. Paulré, « *L'entreprise est-elle vraiment un système ?* » dans Systémique, Théorie et applications, Techniques et documentation, Lavoisier, pages 259-275, 1992.

[Pel 85] X. Pellet, « *Sur la hiérarchisation des décisions, Application à la conduite d'atelier* », Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 1985.

[Per 87] M. Périgord, « *Réussir la qualité totale* », éditions d'organisations, Paris, 1987.

[PeW 92] D. Perry et A. Wolf, « *Foundations for the Study of Software Architecture* ». ACM SIGSOFT, Software Engineering Notes, Vol. 17, n°4, pages 40-52, octobre 1992.

[Por 80] M.E. Porter, « *Competitive strategy: Technique for Analyzing Industries and Competitors* », Free Press, 1980.

[PPR 03] Pôle Productique Rhône-Alpes, « *Maîtriser les processus, pour rendre l'entreprise plus agile* », dossier n°73 du Pôle Productique Rhône-Alpes, 28 pages, décembre 2003.

[RDT 97] Rapide Design Team, « *Rapide 1.0. Pattern Language* ». Reference Manual, juillet 1997.

[Rei 00] R. Reix, « *Systèmes d'informations et management des organisations* », Vuibert, 3ème edition, 2000.

[Rev 96] N. Revault, « *Principes de méta-modélisation pour l'utilisation de canevas d'applications à objets (MetaGen et les frameworks)* », Thèse de Doctorat de l'Université Paris VI, 1996.

[Rob 90] P. Robbins, « *Organization theory* », Englewood Cliffs, 1990.

[Rob 93] M. Roboam, « *La Méthode GRAI : Principes, Outils, Démarche et Pratique* », Teknéa, Toulouse, 1993.

[Ros 75] J. de Rosnay, « *Le Macroscope, vers une vision globale* », Points, Seuil, 1975.

[SaS 01] G. Samtani et D. Sadhwani, « *EAI and Web Services - Easier Enterprise Application Integration ?* », Web Services Architect, 17 octobre 2001.

[ScG 96] M.Shaw et D.Garlan, « *Software Architecture: Perspectives on an Emerging Discipline* ». Prentice-Hall, Englewood Cliffs, N.J., 1996.

[Sch 00] J. Schmidt, « *Enabling Next-Generation Enterprises* », eAI Journal, pages 74-80, juillet-août 2000.

[Sch 96] B. Schweyer, « *Un modèle de gestion de projet appliqué à l'entreprise manufacturière* », Thèse de Doctorat de l'Université de Savoie, 1996.

[Sch 98] A.W. Scheer. « *ARIS* », Handbook on architectures of information systems, Bernus P., Mertins K. et Schmidt G. ed., Berlin, Springer, 1998.

[SDK 95] M.Shaw, R.DeLine, D.Klein, T.Ross, D.Young et G.Zelesnik, « *Abstractions for Software Architecture and Tools to Support Them* ». IEEE Transactions on Software Engineering: Special Issue on Software Architecture, avril 1995.

[Sim 83] H.A. Simon, « *Administration et processus de décision* », Economica, 1983.

[ShC 96] M. Shaw et P. Clements, « *Toward Boxology: Preliminary Classification of Architectural Styles* ». Second International Software Architecture Workshop, pages 50-54, 1996.

[ShC 97] M. Shaw et P. Clements, « *A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems* », Proceedings of the 21st International Computer Software and Applications Conference (COMPSAC'97), 1997.

[Sof 99] SoftTeam, « *Contrôlez totalement le développement d'applications avec UML* », White Paper, 1999.

[Soh 96] C. Sohier, « *Pilotage de cellules adaptatives de production : Apport des systèmes multi-agents* », Thèse de Doctorat de l'Ecole Normale Supérieure de Cachan, janvier 1996.

[SRW 93] J. A. Stafford, D. J. Richardson et A. L. Wolf, « *Aladdin: A Tool for Architecture-Level Dependence Analysis of Software* ». University of Colorado at Boulder, Technical Report CU-CS-858-98, avril 1993.

[Sto 99] M. Stonebraker, « *Integrating Islands of Information* », eAI Journal, pages 1-5, septembre/octobre 1999.

[TaT 87] H. Tardieu et M. Theys, « *Système d'information et pilotage de l'entreprise : 'Quelques boucles étranges'* », Revue internationale de systémique, Vol 1, N°4, pages 471-497, 1987.

[Teb 93] M. Tebaa-Gable, « *Proposition d'un modèle cognitif e raisonnement expert en situation de résolution de problèmes, basé sur une approche systémique des activités cognitives de raisonnement* », Thèse de Doctorat, Université de Franche-Comté, 1993.

[Thé 02] F. Théroude, « *Formalisme et système pour la représentation et la mise en œuvre des processus de pilotage des relations entre donneurs d'ordre et fournisseurs* », Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 2002.

[Tre 96] D. Trenteseaux, « *Conception d'un système de pilotage distribué, supervisé et multicritère pour les systèmes automatisés de production* », Thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 1996.

[UEM 02] Projet UEML, « *Report on the state of the art in enterprise modelling* », UEML European RTD Project IST-2001-34229. www.ueml.org. Deliverable D1.1, 2002.

[Val 95] R. Vallée, « *Historique de la théorie des systèmes* », La modélisation systémique en entreprise, Hermès, 1995.

[VBC 03] B. Vallespir, C. Braesch, V. Chapurlat et D. Crestani, « *L'intégration en modélisation d'entreprise : les chemins d'UEML* », MOSIM'03, Toulouse, 23-25 avril 2003.

[VBW 88] T.E. Vollmann, W.L. Berry, C.C. Whybark, et I. Dow Jones, « *Manufacturing Planning an Control Systems* », 1988.

[VeO 03] H. Verjus et F. Oquendo, « *Final XML ArchWare style-based ADL* ». ArchWare European RTD Project IST-2001-32360, Deliverable D1.3b, juin 2003.

[Ver 02] F. Vernandat, « *UEML : Towards a unified enterprise modelling language* », International Journal of Production Research, volume 40, n° 17, 4309 – 4321, 2002.

[Ver 99] F. Vernadat, « T*echniques de modélisation en entreprise : Applications aux processus Opérationnels* », économica, 1999.

[Wil 83] R. Wilensky, « *Planning and understanding: a computational approach to human reasoning* », Addison Wesley, 1983.

[Wil 99] D. Wile, « *AML: An Architecture Meta Language* ». Proceedings 14th International Conference on Automated Software Engineering, pages 183-190. Cocoa Beach, FL, octobre 1999.

[Zwe 98] A. Zwegers, « *On system architecting: A study in shop floor control to determine architecting concept and principles* », Chapitre 6, thèse, 1998.

# Annexes

# TABLE DES MATIERES DES ANNEXES

# Annexe 1 :
# Caractéristiques des structures de pilotage

# Annexe 1 :
# Caractéristiques des structures de pilotage

L'objectif du pilotage est de contrôler la réalisation d'un système dans un environnement en fonction d'un but. Nous avons présenté en détail les différentes phases de pilotage permettant cette réalisation au sein du chapitre 2. Cependant, nous n'avons pas abordé la manière dont s'organise le pilotage dans l'entreprise. En effet, le pilotage est très souvent distribué entre plusieurs entités en interaction. Nous allons donc au sein de cette annexe, présenter très rapidement les principales structures de pilotage dans une entreprise [Thé 02] [Cha 02].

L'évolution des techniques, et plus particulièrement celles liées aux technologies de communication, a provoqué une évolution des différentes structures de pilotage durant ces dernières années [DBW 91]. Ces différentes structures étudiées sont présentées dans de nombreux travaux [Arc 91] [Tre 96] [Soh 96] [Zwe 98]. Elles sont représentées au sein de la figure ci-dessous.

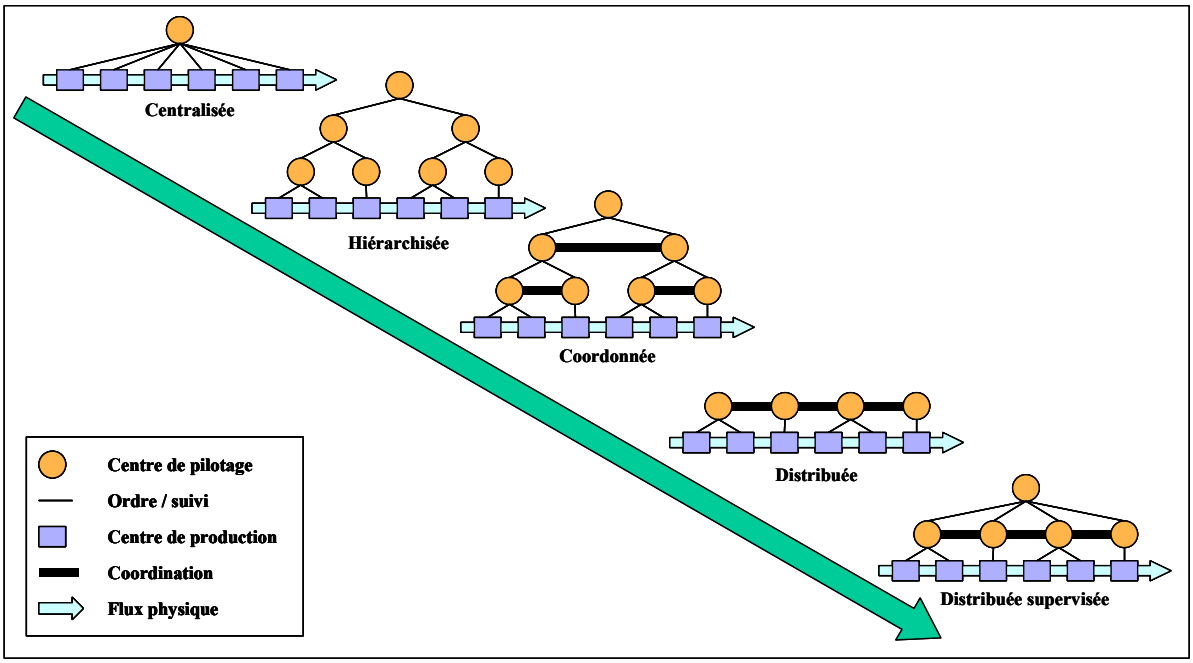



*Figure A.1. : L'évolution des structures de pilotage*

Comme le montre ce schéma, nous nous appuyons sur la notion de centre de pilotage et de centre de production pour décrire ces différentes structures. Un **centre de pilotage** est une structure capable de mettre en œuvre les trois étapes liées au pilotage. Un **centre de production** est une structure capable de réaliser les actions spécifiées par le référent. Les actions à réaliser sont définies par des flux désignés **ordres,** tandis que les résultats obtenus sont appelés **suivis**.

Nous allons maintenant recenser les caractéristiques qu'un centre de pilotage doit posséder pour pouvoir réaliser les différentes architectures d'une entreprise. Au terme de ces descriptions, nous présentons un tableau de synthèse précisant les caractéristiques, les avantages et les inconvénients de chaque structure.

# 1. Structure centralisée

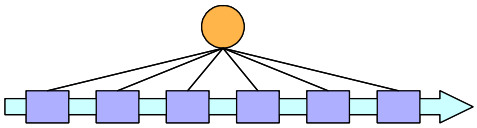


*Figure A.2. : Structure centralisée*

Une **structure centralisée** (*Centralised form*) est caractérisée par un centre de pilotage **unique** qui supervise et coordonne le fonctionnement des différents centres de production. Les différents ordres circulent du centre de pilotage vers les différents centres de production qui lui renvoient le suivi les concernant.

L'avantage de cette approche est que le centre de pilotage dispose d'un accès à toutes les informations du système piloté, ce qui lui permet de prendre des décisions concernant l'ensemble des centres de production.

D'un autre coté, cette structure est difficile à mettre en œuvre lorsqu'il existe un nombre important de centres de production car tous les cas doivent être pris en compte sur une ressource unique (problème de temps de réponse due à la charge à gérer). De plus, la flexibilité est médiocre car les modifications sont délicates du fait du regroupement au sein d'un centre unique et il existe un risque de paralysie complète en cas de défaillance du centre de pilotage.

# 2. Structure hiérarchisée

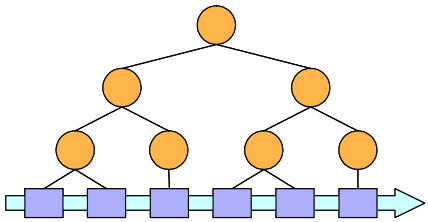


*Figure A.3. : Structure hiérarchisée*

La **structure hiérarchisée** (*Proper hierarchical form*) vise à éliminer les défauts constatés de la structure centralisée en s'appuyant sur les avancées technologiques des matériels informatiques. Elle introduit différents **niveaux** où les centres de pilotage sont organisés dans une structure pyramidale. Chacun de ces niveaux est caractérisé par la perception qu'il a des différents centres de production (degré de granularité) et par l'horizon de travail (horizon temporel lié aux différentes phases de pilotage). Les centres de pilotage subordonnés (esclaves) sont **totalement contraints** aux ordres du centre de pilotage du niveau supérieur (maître). Chaque centre de pilotage définit, d'une part les objectifs des centres de pilotage du niveau inférieur (s'ils existent) et, d'autre part, fournit ses résultats au centre de pilotage du niveau supérieur (s'il existe). En d'autres termes, les décisions descendent en étant adaptées jusqu'à l'exécution, tandis que les informations de suivi remontent des « esclaves » vers les « maîtres » en étant agrégées d'un niveau à l'autre.

Nous pouvons noter un certain nombre d'avantages liés à cette approche :

- les relations rigides (maître / esclaves) et la division de la prise de décision permettent un temps de réponse rapide,
- les centres de pilotage sont moins complexes (par rapport au centre de pilotage unique de l'approche centralisée), ce qui facilite leur conception et implémentation.

D'un autre coté, cette approche présente les inconvénients suivants :

- si un lien de subordination ou un centre de pilotage ne fonctionne pas, c'est l'ensemble des subordonnés qui est paralysé (problème de tolérance aux fautes),
- la structure hiérarchique est établie dès la conception, ce qui signifie qu'une modification de cette structure est délicate à mettre en œuvre a posteriori (flexibilité),
- la décision prise, suite à l'apparition d'un problème à un niveau donné, peut ne pas être adaptée du fait de l'agrégation des informations, ce qui pose un problème de fiabilité.

Le problème de tolérance aux fautes peut être en partie traité par l'introduction, lors de la conception, de centres de pilotage redondants au sein de la structure, de façon à remplacer un centre défaillant. Toutefois, il est à noter que cette solution augmente la complexité des centres de pilotage.

## 3. Structure coordonnée

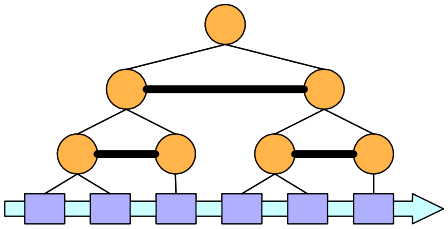


*Figure A.4. : Structure coordonnée*

Cette structure s'appuie sur la structure hiérarchisée. La **structure coordonnée** (*Modified hierarchical form*) a été développée pour fournir aux centres de pilotage un degré d'autonomie plus important. L'autonomie d'un centre peut être caractérisée par sa capacité à entreprendre des actions sans faire intervenir les centres de pilotage de plus haut niveau [Kal 85] [Pel 85] [Bin 91]. Cette autonomie induit une coordination entre les centres de pilotage d'un même niveau.

Plusieurs avantages découlent directement de cette structure :

- les interactions entre niveaux hiérarchiques sont diminuées,
- cette approche est plus tolérante aux défaillances des centres de pilotage des niveaux supérieurs du fait de l'autonomie attribuée aux subordonnés,
- la notion d'autonomie transfère des fonctions d'un centre maître aux centres subordonnés, ce qui laisse plus d'autonomie au centre maître.

Toutefois, si certains désavantages de l'approche hiérarchisée sont minimisés, cette structure n'en reste pas moins soumise aux problèmes liés à une structure hiérarchique. De plus, de nouveaux problèmes émergent :

- les interactions au sein d'un même niveau apparaissent,
- l'autonomie d'un centre augmente sa complexité,
- la tolérance aux défaillances des centres de haut niveau n'est que temporaire,

- du fait de son autonomie, un centre doit posséder plus que les informations agrégées des centres subordonnés (informations de coordination).

Cette structure atténue certains défauts de la structure hiérarchique, mais elle complexifie les aspects de conception

## 4. Structure distribuée

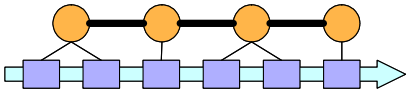


*Figure A.5. : Structure distribuée*

Pour éliminer les désavantages des structures précédentes, différents auteurs proposent une structure distribuée (*Heterarchical form*). Une **structure distribuée** considère que chaque centre de pilotage est autonome (sans relations maître / esclaves) et coopère avec l'ensemble des autres centres. Dans une telle approche, il n'y a qu'un seul niveau de pilotage. Les décisions sont prises par accord mutuel entre les différents centres de pilotage et les informations sont échangées librement.

L'avantage d'une telle approche est d'offrir une grande flexibilité et une bonne tolérance aux perturbations du fait de l'autonomie de chaque centre de pilotage.

Malheureusement, le problème qui émerge dans une telle structure est qu'il est difficile d'optimiser globalement le système piloté et que le comportement du système n'est pas toujours prévisible [Bon 98].

## 5. Structure distribuée supervisée

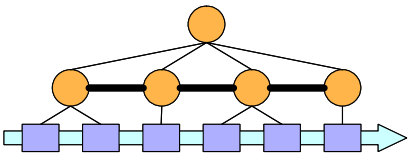


*Figure A.6. : Structure distribuée supervisée*

La **structure distribuée supervisée** (*Holonic form*) a été proposée pour combiner les avantages d'une structure hiérarchisée et d'une structure distribuée [Tre 96] [Bon 98]. Une telle structure doit permettre d'obtenir une architecture de pilotage :

- prévisible et permettant une optimisation globale,
- robuste vis à vis des perturbations,
- flexible,
- de conception aisée,
- réactive.

Chaque centre de pilotage garde son autonomie de décision mais peut demander « conseil » au centre de plus haut niveau. Ce centre de plus haut niveau a une visibilité sur l'ensemble du système alors que les centres de pilotage de plus bas niveau ont une vue plus détaillée et précise d'une situation locale. Le rôle du centre de haut niveau est donc de coordonner et

d'optimiser globalement le système tandis que le rôle des centres de bas niveau est de permettre une réaction rapide à une perturbation.

Par contre, des redondances peuvent apparaître du fait de l'autonomie attribuée à chaque centre de pilotage. Ces redondances nécessitent la mise en place de mécanismes de résolution de conflits (cohérences des décisions prises si des fonctions sont redondantes). Dans ce cas, la difficulté va être de déterminer le niveau d'autonomie de chaque centre de pilotage :

- sans autonomie, il s'agit d'une structure centralisée,
- avec autonomie complète, il s'agit d'une structure distribuée.

De fait, un compromis doit être établi entre obéissance et indépendance des centres de bas niveau.

L'avantage le plus important de cette structure est la possibilité de prévoir le comportement global du système en fonction de la progression individuelle de chaque centre de pilotage. Enfin, c'est une architecture intéressante parce qu'elle peut être appliquée à une structure centralisée en effectuant graduellement la migration sur la structure en place.

# 6. Synthèse

Le tableau suivant résume les caractéristiques, les avantages et les désavantages des différentes structures de pilotage présentées [DBW 91] :

| CARACTÉRISTIQUES | AVANTAGES | DÉSAVANTAGES |
|---|---|---|
| **Structure centralisée (*centralized*)** | | |
| ✓ Toutes les décisions sont prises dans un centre de pilotage unique.<br>✓ Toutes les informations sont centralisées à un endroit donné. | ✓ Accès à l'ensemble des informations.<br>✓ Optimisation globale possible.<br>✓ Source unique de distribution d'information. | ✓ Réponse lente et non régulière (surcharge du système).<br>✓ Confiance portée sur un centre de pilotage unique.<br>✓ Modification du pilotage difficile. |
| **Structure hiérarchisée (Proper hierarchical form)** | | |
| ✓ Relation entre niveaux de décision maître / esclaves rigide.<br>✓ Un maître coordonne toutes les activités qui lui sont subordonnées.<br>✓ Informations agrégées à chaque niveau. | ✓ Implémentation graduelle, redondance et problème de développement réduit.<br>✓ Addition incrémentale de centre de décision possible.<br>✓ Possibilité de comportement adaptatif.<br>✓ Possibilité de gestion de différentes échelles de temps.<br>✓ Réponse rapide. | ✓ Capacité de décision limitée.<br>✓ Augmentation du nombre de connexions entre les niveaux de décision.<br>✓ Difficultés à gérer le contrôle adaptatif.<br>✓ Difficulté à gérer la mise en place de modifications inattendues. |
| **Structure coordonnée (Modified hierarchical form)** | | |
| ✓ Suppression des relations de maître / esclaves entre niveaux de décision.<br>✓ Coopération entre les centres de pilotage d'un même niveau. | ✓ Tous les avantages d'une structure hiérarchique.<br>✓ Autonomie des centres de pilotage. | ✓ La plupart des désavantages d'une structure hiérarchisée.<br>✓ Augmentation du nombre de connexions entre les centres de pilotage. |

| **Structure distribuée (Heterarchical form)** | | |
|---|---|---|
| ✓ Pas de relation maître / esclaves.<br>✓ Autonomie complète de décision.<br>✓ Pilotage distribué pour la coordination des activités.<br>✓ Informations locales à chaque centre de pilotage. | ✓ Autonomie complète de décision.<br>✓ Diminution de la complexité de gestion.<br>✓ Tolérance aux fautes implicite.<br>✓ Reconfiguration et adaptation aisées.<br>✓ Diffusion rapide de l'information. | ✓ Limitations techniques des centres de pilotage.<br>✓ Optimisation locale. |
| **Structure distribuée supervisée (Holonic form)** | | |
| ✓ Autonomie partielle des centres de pilotage de bas niveau. | ✓ 0ptimisation globale possible.<br>✓ Migration ne nécessitant pas forcément de refonte totale.<br>✓ Tolérance aux fautes.<br>✓ Rapidité. | ✓ Difficulté de décider le niveau d'indépendance de bas niveau.<br>✓ Mise en place de mécanismes de gestion de conflits, cohérence des décisions. |

# Annexe 2 :
# Etude approfondie relative à l'EAI

# Annexe 2 :
# Etude approfondie relative à l'EAI

Cette annexe présente une étude générale relative à l'EAI. La première section décrit les principales technologies utilisées dans les solutions d'EAI. La seconde partie présente les modèles EAI et leur évolution. Une description de la manière dont les Services Web peuvent améliorer l'EAI est ensuite donnée au sein de la troisième partie de cette annexe. Enfin, la dernière partie présente les formats d'échange au sein des différentes solutions EAI.

## 1. Les technologies EAI

Cette partie présente respectivement les topologies d'intégration, les modes de communication, les mécanismes d'échange, les différentes couches des solutions EAI et montre l'intérêt de l'utilisation d'un format pivot au sein de telles solutions [Par 01].

### 1.1. Les topologies d'intégration

Nous décrivons ici les différentes technologies pouvant être employées pour intégrer des applications hétérogènes.

#### 1.1.1. Le « point-à-point »

La topologie « point-à-point » consiste à faire communiquer les différentes applications en les connectant directement les unes aux autres. Chaque interface est alors codée directement au sein des différentes applications. Cette topologie n'offre donc aucune flexibilité. C'est en réalité l'exemple même de ce qu'une solution EAI tente d'éviter.

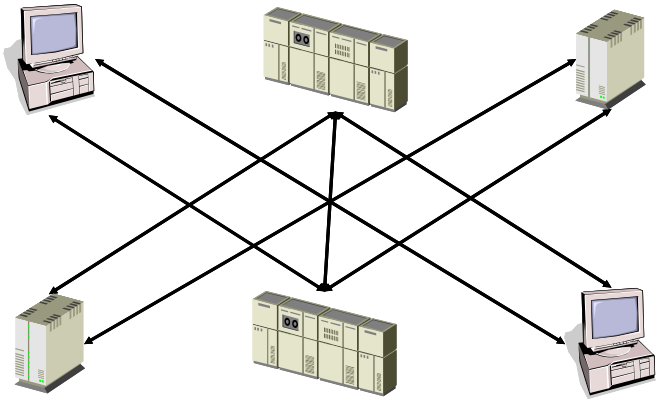


*Figure A.7. : Le « point-à-point »*

#### 1.1.2. Le bus (Publication / abonnement décentralisé)

Au sein de cette topologie, les applications envoient leurs messages à un bus qui est accessible par les autres applications. Une ou plusieurs applications peuvent s'abonner à une émission de message sur ce bus. Toutes les applications peuvent ainsi voir l'intégralité des messages mais seules les applications qui sont abonnées à un type de message donné reçoivent effectivement ces messages. De plus, il est généralement nécessaire d'installer un agent sur chaque application pour s'interfacer avec le bus d'échange. Le rôle de cet agent est en fait de contrôler les différents mouvements d'information sur ce bus.

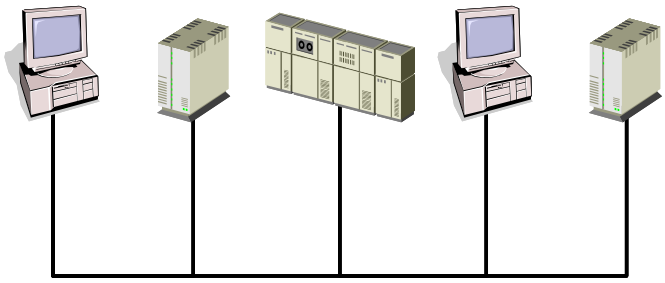


*Figure A.8. : Le bus (Publication / abonnement décentralisé)*

### 1.1.3. Le bus « Hub and Spoke » (publication / abonnement centralisé)

Au sein de cette topologie, des messages sont envoyés à un hub central qui est souvent implémenté sur un serveur dédié. L'application source envoie un message dans un format donné, le hub traduit (si besoin est) le message et le diffuse vers les diverses applications abonnées. Cette approche évite, en partie, la répétition des saisies de données et améliore ainsi la réutilisabilité.

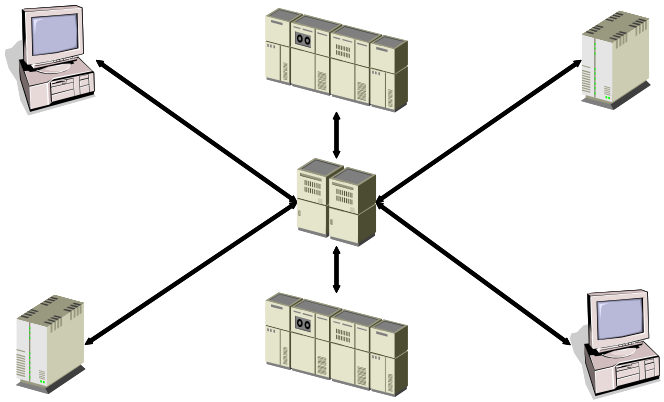


*Figure A.9. : Le bus « Hub and Spoke »*

### 1.1.4. Le bus « Snowflake » (publication / abonnement centralisé)

Il est possible d'associer et de configurer plusieurs hub au sein de son système d'intégration. Un bus « Snowflake » permet ainsi de « regrouper » autour d'un même hub, certaines applications échangeant énormément de données. Différents hubs peuvent également être reliés pour gérer tous les autres échanges de données.

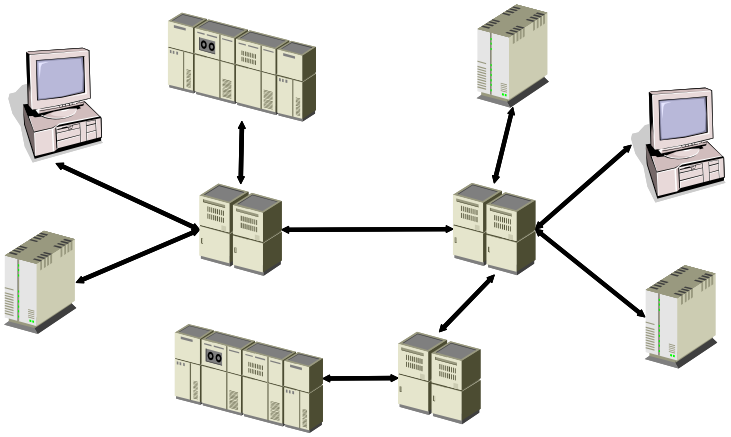


*Figure A.10. : Le bus « Snowflake »*

## 1.2. Les modes de communication

Nous pouvons énumérer cinq modes de communication différents (deux en mode synchrone et trois en mode asynchrone).

### 1.2.1. Le mode « Conversationnel »

Ce mode de communication est un mode synchrone. Nous pouvons le comparer à un appel téléphonique. Les deux applications doivent ainsi être disponibles en même temps pendant la conversation (l'échange de données). Au sein de ce mode, il peut y avoir échanges entre les applications, c'est-à-dire plusieurs requêtes/réponses successives.

### 1.2.2. Le mode « Requête / Réponse »

Il s'agit également d'un mode de communication synchrone. La différence avec le mode précédent se situe au niveau des échanges de messages. Au sein de ce mode, il n'y a seulement qu'un seul aller/retour au niveau des communications de messages. Cela se traduit donc généralement par un appel de fonctions distantes de type RPC – *Remote Procedure Call*.

### 1.2.3. Le mode « Emission de messages »

Ce mode de communication est un mode asynchrone. Ce mode peut être comparé à l'envoi d'une lettre par la poste. L'application source envoie l'information sans s'inquiéter de la disponibilité de l'application cible.

### 1.2.4. Le mode « Mise en file d'attente de messages »

Ce mode de communication est aussi un mode asynchrone. Il fonctionne de la même manière que le mode précédemment présenté mais avec un aspect « sécurité » supplémentaire. En effet, lorsqu'un message est envoyé, il est stocké au sein d'une mémoire pour s'assurer qu'il ne sera pas perdu avant sa livraison à l'application cible.

### 1.2.5. Le mode « Publication / Abonnement »

Ce dernier mode de communication est lui aussi asynchrone. Il fonctionne de la même manière que le mode « Mise en file d'attente de messages » mais ici, les émissions de messages sont de type 1 → n, alors que le type d'émission précédent est de type 1 → 1.

## 1.3. Les différents mécanismes d'échange

Cette partie décrit les différents mécanismes d'échange utilisés au sein des solutions EAI.

### 1.3.1. Le transfert de fichiers

Employer un mécanisme d'échange basé sur le transfert de fichiers implique nécessairement que les deux systèmes (ou bases de données) soient compatibles. Si ce n'est pas le cas, des problèmes peuvent apparaître. Ainsi, il pourrait être nécessaire d'adapter le contenu et les formats des messages échangés entre ces différentes applications.

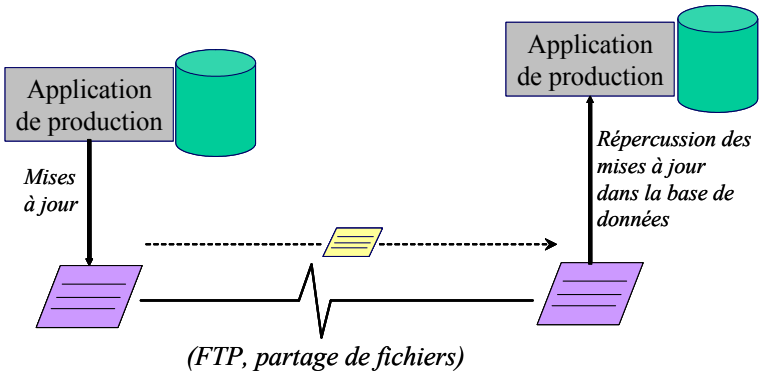



*Figure A.11. : Le transfert de fichiers*

### 1.3.2. Le mécanisme « Extraction, Transformation, Chargement » (ETL)

Ce mécanisme d'échange, basé sur la transformation du contenu de données, peut être réalisé grâce à l'utilisation d'un dictionnaire de méta-données. Ce dictionnaire de méta-données s'appuyant sur les données des applications sources (SGDT, fichiers plats, etc.), les transforme dans le format requis par l'application cible, et les envoie à cette application cible.

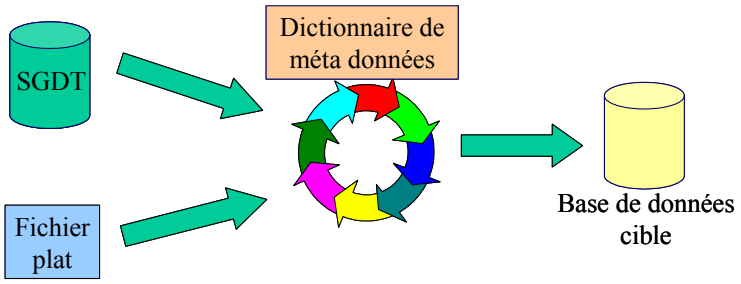



*Figure A.12. : Le mécanisme « ETL »*

### 1.3.3. La réplication de données

Une autre technologie consiste à utiliser un serveur de réplication. Cette technologie est généralement employée quand les diverses bases de données doivent être mises à jour en temps réel, contrairement aux deux solutions précédentes, qui sont généralement utilisées en mode batch.

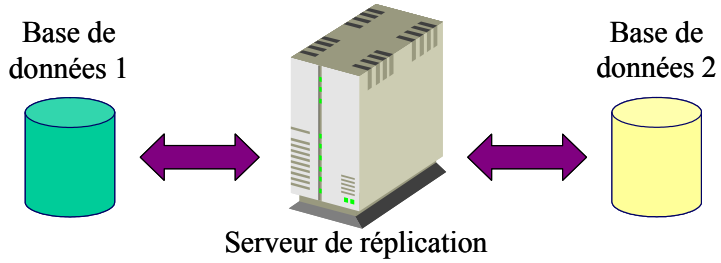



*Figure A.13. : La réplication de données*

Le but de ce serveur de réplication est de répliquer en temps réel l'information d'une base de données dans une autre base de données. Ceci implique naturellement que les deux bases de données soient compatibles. Si cela n'est pas le cas, il faut coupler ce serveur de réplication à un mécanisme permettant de gérer ce changement de format de données.

### 1.3.4. L'échange de messages

Nous avons vu, au sein de ce mémoire, que la clé d'une solution d'intégration passe par la maîtrise des échanges de messages entre différentes applications. Au sein d'une solution EAI, c'est le courtier de message qui assure cet échange. Le rôle de ce courtier de message est également de transformer les formats de données, de contrôler les files d'attente de message et d'envoyer les flots d'information aux différentes applications.

### 1.3.5. Les différentes couches d'une solution EAI

Une solution EAI fait interagir de manière optimale divers progiciels hétérogènes de l'entreprise afin de proposer une vision unifiée de l'information, en s'appuyant sur l'approche processus. Pour permettre ceci, une solution EAI s'articule autour de plusieurs couches qui sont représentées au sein de la figure ci-après :

- la couche « *Connexion et Transport* »,
- la couche « *Transformation – Interprétation des données* »,
- la couche « *Routage* »,
- la couche « *Modélisation* ».

#### *1.3.5.1. La couche « Connexion et Transport »*

Afin de faire communiquer diverses applications les unes avec les autres via une solution EAI, il faut être en mesure d'accéder physiquement à ces applications, c'est-à-dire de s'interfacer avec ces applications. Ainsi, la première couche d'une solution EAI correspond à la récupération et au transport des données de ces applications de manière physique. Cette couche « Connexion et Transport » comprend notamment les différents **connecteurs** (progiciels ou standards) qui permettent de s'interfacer avec les applications pour récupérer et/ou insérer des données, ainsi que les **protocoles de transport** qui permettent eux, d'acheminer et de distribuer les données.

#### *1.3.5.2. La couche « Transformation – Interprétation des données »*

Après la couche relative à la connexion avec les différents progiciels de l'entreprise, vient la couche de transformation. La couche « Transformation – interprétation des données » sert à structurer le format des données véhiculées au sein de la solution EAI. Le but de cette couche est donc de transformer le format des messages (données des diverses applications) dans un format pivot propre à la solution EAI. Ce format pivot est, à l'heure actuelle, généralement basé sur le format XML.

#### *1.3.5.3. La couche « Routage »*

La couche suivante, la couche « Routage », sert, elle, à aiguiller les différents flux d'information vers les bonnes applications. Cet aiguillage est rendu possible grâce à la représentation des processus métier fournie par la couche modélisation décrite au sein du paragraphe suivant. D'un point de vue technique, cet aiguillage peut notamment être rendu possible grâce à la mise en place d'outils logiciels permettant de gérer des files d'attentes de messages.

Aussi, pour couvrir les fonctionnalités de ces deux dernières couches, c'est généralement le courtier de messages propre à chaque solution EAI qui réalise ces fonctions (Transformation – Interprétation des données et Routage).

### *1.3.5.4. La couche « Modélisation »*

La dernière couche correspond à la couche « Modélisation ». Cette couche est très importante car elle permet de prendre en charge les processus des entreprises. Ces processus permettent de décrire les tâches, les organisations et les personnes impliquées, les informations nécessaires, et les applications requises pour les différentes étapes de ces processus.

Aussi, c'est à ce niveau que la plupart des solutions EAI ont un manque car elles ne possèdent pas de réelle gestion de ces processus.

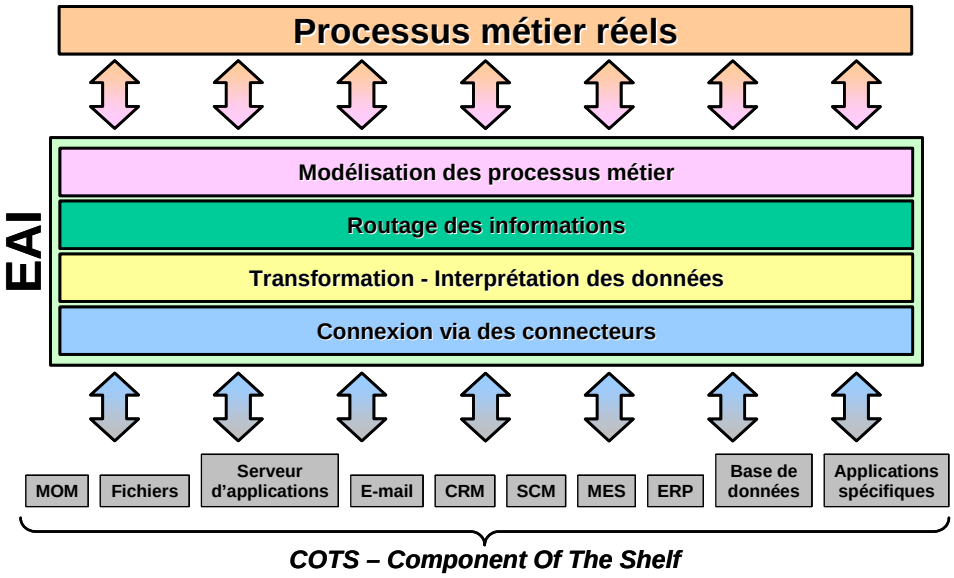



*Figure A.14. : Les différentes couches d'une solution EAI*

### *1.3.5.5. L'intérêt d'un format pivot*

Nous venons de le voir, les connecteurs jouent un rôle important au sein des solutions EAI. Le format des messages a un rôle tout aussi important. En effet, si deux applications « discutent » entre elles, il est nécessaire que les deux connecteurs de ces applications puissent se comprendre en parlant le même langage.

C'est la raison pour laquelle les messages sont généralement traduits dans un format pivot comme le langage XML. De plus, nous pouvons également remarquer que l'emploi du langage XML laisse plus de flexibilité concernant le B2B.

En effet, auparavant, si deux entreprises voulaient faire du B2B, la manière la plus commune de le mettre en application était d'employer le standard EDI. Cependant, l'EDI ne couvre pas tous les secteurs de l'entreprise (il n'y a aucune considération des processus de chaque entreprise ; il n'y a aucun moyen de synchroniser une application de contrôle qualité avec une application de gestion de production par exemple). Grâce à une plate-forme d'intégration, en utilisant XML, deux entreprises peuvent facilement faire du B2B en comblant les manques relatifs à l'utilisation de l'EDI.

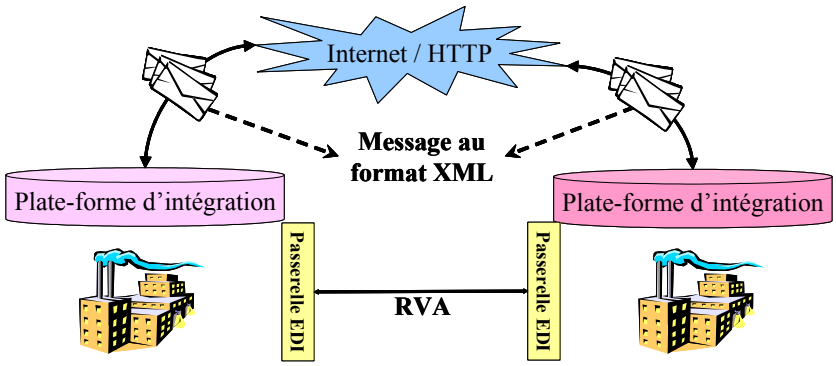



*Figure A.15. : L'utilité d'un format pivot*

# 2. Le modèle EAI et son évolution

Cette partie décrit dans un premier temps le modèle EAI et sa topologie, puis dans un second temps, le modèle eAI [Ber 01].

## 2.1. Le modèle EAI « A to A » et sa topologie

Le premier modèle EAI, appelé modèle « A to A » (pour application to application) a été créé approximativement en 1999. Il est exclusivement consacré aux communications entre applications d'une même entreprise.

**Moteur d'intégration**

**Composants**

**Adaptateurs**

**Communication**

**La couche moteur d'intégration :** Elle transforme et diffuse les messages (courtier de message).
**La couche composants :** Apport métier à la logique d'intégration et/ou serveur d'applications,
**La couche adaptateurs :** Les adaptateurs correspondent aux connecteurs d'extraction et d'insertion de données,
**La couche communication :** Ceci correspond à un « protocole » de communication donnée (MOM, ORB, HTTP).

### 2.1.1. La couche communication

*Rôle :*
La couche de communication assure le transport des données et/ou l'invocation de services. Il masque au programmeur la complexité du système d'exploitation et du réseau.

*Technologies :*
MOM – *Message Oriented Middleware* :
- messages placés dans des files d'attente auxquelles les applications sont abonnées (mécanismes de type « Publication/abonnement »),
- couplage faible entre les applications (communication asynchrone),
- logiciel de prédilection des plates-formes d'intégration.

ORB – *Object Request Brokers* :
- bus de composants métiers distribués fournissant des services applicatifs,
- les deux applications doivent être disponibles simultanément (et en permanence) pour pouvoir échanger l'information.

HTTP – *Hyper-Text Transport Protocol* :
- protocole complémentaire indispensable aux échanges via Internet.

### 2.1.2. La couche adaptateurs

*Rôle :*

- Les adaptateurs (ou connecteurs) assurent l'interface de la plate-forme d'intégration avec les applications ou les bases de données existantes.
- Ils permettent de conserver les données de chaque application existante dans leur format natif et sur leur support d'origine, afin que l'application continue d'assurer ses services, sans coût de migration.

L'extraction et l'insertion des données sont réalisées de façon :

- non intrusive (l'application initiale n'est pas modifiée),
- en gérant les erreurs ou exceptions pour la plate-forme d'intégration.

Les adaptateurs fournissent une interface standard d'accès à l'application et en masquent la complexité. L'adaptateur est un élément de différenciation important entre les offres EAI.

Les éditeurs proposent également :

- des SDK – *Software Development Kit* – pour développer les adaptateurs manquants ou ceux liés à des applications spécifiques,
- des prestations de services assurées par eux-mêmes ou par des partenaires.

### *2.1.3.* La couche composants

*Rôle :*

- Les composants supportent la logique fonctionnelle de traitement de l'information manipulée par la plate-forme d'intégration.
- Ils constituent le référentiel métier du moteur d'intégration.

Le moteur d'intégration s'adresse aux composants pour réaliser un traitement spécifique suite à la réception d'un message.

*Technologies :*

- Composants générés par la plate-forme d'intégration.
- Composants externes à la plate-forme d'intégration, pris en charge par un serveur d'application standard et connectés au moteur d'intégration via un adaptateur particulier.

### 2.1.4. La couche moteur d'intégration

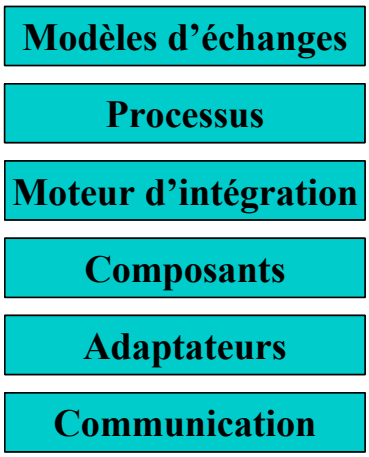
*Rôle :*

- Le moteur d'intégration (ou Message Broker) convertit les données sources dans le format de l'application destinataire, et fait parvenir ces données à l'application destinataire.
- Il facilite les échanges entre applications car il prend en charge la gestion des différences de formats et l'acheminement vers le ou les destinataires.

Des services complètent l'architecture :

- Administration centralisée des éléments du système (files d'attente, connecteurs applicatifs, composants).
- Surveillance des échanges, de leur volumétrie et des performances de la plate-forme.

## 2.2. Le modèle eAI « B to B »




Ce modèle complète le problème initial d'intégration. Il correspond au raccordement du marché de l'EAI et du marché de B2B.

Ce modèle est composé de six couches techniques et fonctionnelles complémentaires pour adresser un besoin global de l'entreprise étendue.

Le modèle eAI permet à l'entreprise de préparer son système d'information avant de l'ouvrir à ses partenaires en associant l'intégration interne et une démarche d'urbanisme du système d'information.

### 2.2.1. La couche processus

*Rôle :*

- Un processus, déclenché par un événement métier, formalise les enchaînements de traitements sur des flux d'information.
- Il prend en compte l'intervention humaine pour la fourniture de données supplémentaires, la prise de décision, ou la résolution de cas de routage trop complexe pour être gérées nativement par le moteur d'intégration.

La gestion des processus métiers se concrétise techniquement par un moteur de workflow qui pallie les limitations du moteur d'intégration :

- interruption / reprise de processus,
- envoi de notification à un utilisateur,
- prise de décision en cas de non réponses aux notifications.

Cette couche processus permet de séparer la modélisation métier de l'implémentation technique des processus :

- moteur d'intégration = gestion des processus techniques (séquencement de la traduction et du routage des messages),
- workflow = gestion des processus métier (automatisation et coordination du routage d'informations ou de documents entre utilisateurs et systèmes).

### 2.2.2. La couche modèles d'échanges

*Rôle :*

- Les modèles d'échanges établissent des relations entre partenaires sur la base de définition commune de modèles de données pour les échanges.
- C'est la couche de formalisation des messages reçus et émis des partenaires lors d'une relation commerciale faisant intervenir le système d'information des différentes parties.

La couche modèle d'échange permet l'interconnexion des systèmes d'information entre entreprises partenaires : c'est l'intégration étendue.

*Fonctionnalités :*

- Il constitue le référentiel pour la gestion des partenaires :
  - rôle des partenaires dans l'échange,
  - échanges ou transactions auxquels ils peuvent participer.
- Flexibilité et facilité d'intégration.

XML joue un rôle prépondérant :

- Echange de données normalisées en XML (cXML, xCBL, Biztalk, RosettaNet, ebXML…), invocation de services distants (SOAP).
- Modélisation / pilotage XML du Workflow, interopérabilité des processus métier.

# 3. EAI et Services Web

Cette partie a pour objectif de présenter les Services Web et de voir la manière dont ils peuvent améliorer l'EAI.

## 3.1. Architecture de Services Web versus architecture EAI

### 3.1.1. Empaquetage/Transport

Ce niveau permet à l'information d'être empaquetée dans des messages spécifiques et d'être transportée entre différents participants.

### 3.1.2. Service

Cette couche décrit les interfaces opérationnelles d'un Service Web.

### 3.1.3. Qualité de services

Cette couche décrit des aspects non-opérationnels de services, y compris des dispositifs de fiabilité et de sécurité.

### 3.1.4. Orchestration

Cette couche décrit comment les services agissent les uns sur les autres au sein de processus utilisant des descriptions de workflows (déroulement d'opération).

### 3.1.5. Contrat

Cette couche décrit la manière dont des partenaires métier spécifiques collaboreront en vue d'exécuter un certain processus métier partagé.

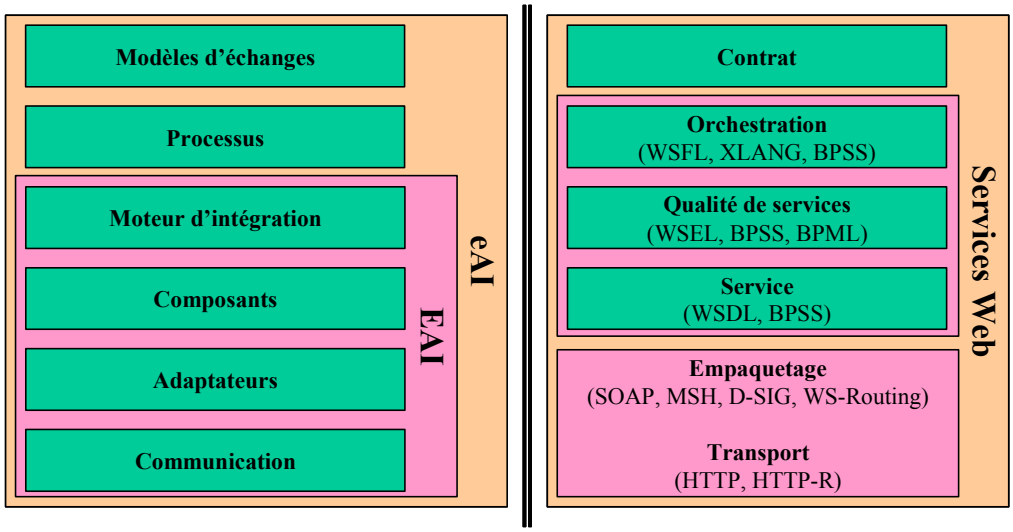



*Figure A.16. : EAI et Services Web*

## 3.2. Description des Services Web

Les Services Web fournissent une technologie informatique répartie pour indiquer les services d'applications sur Internet ou intranet en utilisant des protocoles et des formats standard XML. L'utilisation de protocoles standard XML rend la plate-forme de Services Web, le langage et le fournisseur indépendant. De plus, l'utilisation de XML le rend un candidat idéal pour son utilisation au sein de solutions EAI.

Les Services Web éliminent les questions d'interopérabilité de solutions existantes, telles que CORBA et DCOM, par l'utilisation de normes Internet ouvertes – *Web Services Description Language* (WSDL – pour décrire), *Universal Description Discovery and Integration* (UDDI – pour annoncer), *Simple Object Access Protocol* (SAOP – pour communiquer) et *Web Services Flow Language* (WSFL – pour définir les workflows, ce n'est cependant pas encore un standard W3C).

## 3.3. EAI et Services Web

Les Services Web sont justes une autre technologie qui permet de faire de l'EAI, et il peut changer de manière significative l'approche « point-à-point » traditionnelle [SaS 01].

En utilisant les Services Web pour intégrer lâchement des applications, une entreprise réalise juste un sous-ensemble des fonctionnalités offertes par l'EAI. D'autre part, l'EAI adopte une approche holistique complète afin d'intégrer et de relier toutes les applications et systèmes qui soutiennent les processus métier d'une entreprise. L'EAI constitue des années d'engagement et d'effort continus à partir des différentes unités techniques de l'entreprise, un investissement élevé, et des ressources substantielles.

Les Services Web, sous leur forme courante de collections liées de services, sont plutôt une solution ad-hoc qui peut être développée rapidement et facilement, éditée, découverte, et évolutive. Dans la génération actuelle de Services Web, il est seulement possible de réaliser l'intégration fonctionnelle entre les applications. Par nature, elles ne sont pas transactionnelles et ne fournissent pas la fonctionnalité de base de type « requête/réponse ». La prochaine génération des Services Web, sera fonctionnellement et technologiquement améliorée, offrant l'encapsulation d'interface utilisateur et un certain niveau de sécurité. Elles pourront empaqueter une application et l'inclure dans une autre application.

Les solutions actuelles d'EAI qui se concentrent sur l'intégration d'applications prédominantes devront être changées sensiblement, car les applications empaquetées à l'avenir exposeront leurs fonctions comme des services en utilisant des technologies telles que XML, SAOP et UDDI. Ainsi, les solutions EAI devront fournir un large support pour l'intégration de services plutôt que l'intégration d'applications.

## 3.4. Les différences majeures entre des solutions EAI traditionnelles et les Services Web

*(Note: Certaines de ces différences tiennent compte des futurs perfectionnements proposés pour les Services Web).*

Quelques différences essentielles entre les solutions d'EAI traditionnelles et les Services Web sont listées ci-dessous :

- **Simplicité :** Il n'y a aucun doute que les Services Web sont beaucoup plus simples à concevoir, développer, maintenir et employer, comparés à une solution typique d'EAI, qui peut impliquer la technologie distribuée telle que DCOM ou CORBA. Une fois que le cadre pour développer et utiliser des Services Web sera prêt, il sera relativement facile d'automatiser de nouveaux processus métier utilisant de multiples applications.
- **Standards ouverts :** À la différence des solutions EAI propriétaires, les Services Web sont basés sur des normes ouvertes telles que UDDI, SOAP, HTTP et c'est probablement le facteur le plus important qui mènerait à une large adoption des Services Web. Le fait qu'ils soient construits sur des protocoles existants et omniprésents élimine le besoin pour les entreprises d'investir en supportant de nouveaux protocoles réseau.
- **Flexible :** Puisque les solutions d'EAI peuvent exiger une intégration « point-à-point » (des changements réalisés à une extrémité doivent être propagés à l'autre extrémité), ceci rend les systèmes construits très rigides et longs à développer. L'intégration basée sur des Services Web est tout à fait flexible, car elle est établie sur le couplage entre l'application éditant les services et l'application utilisant ces services.
- **Bon marché :** Les solutions EAI, basées sur des courtiers de message, sont très chères à implémenter. Les Services Web, à l'avenir, pourront accomplir plusieurs des mêmes fonctionnalités, de manière plus rapide et de meilleur marché.
- **Portée :** Les solutions d'EAI, basée sur des courtiers de message, intègrent des applications en les traitant en tant qu'entités simples, tandis que les Services Web permettent à des entreprises de décomposer de grosses applications en plus petites unités logiques indépendantes et construisent des emballages autour de ces dernières.
- **Efficace :** Comme mentionné dans le point précédent, les Services Web permettent à des applications d'être décomposées en plus petits composants logiques, ce qui facilite l'intégration des applications. Ceci rend des solutions de Services Web pour l'EAI beaucoup plus efficaces que les solutions EAI traditionnelles.
- **Dynamique :** Les Services Web fournissent une approche dynamique à l'intégration en offrant des interfaces dynamiques, tandis que les solutions EAI traditionnelles sont, par nature, beaucoup plus statiques.

# 4. Les formats d'échanges au sein des solutions EAI

Beaucoup de normes, basées sur XML et Java, frappent à la porte des solutions EAI propriétaires. Ces normes devraient autoriser une intégration du système d'information à un coût inférieur. Poussées par les géants du logiciel, elles devront s'avérer fiables avant d'être impliquées au sein des solutions spécialisées d'éditeurs.

Les solutions d'EAI ont longtemps été basées sur des infrastructures propriétaire. Leur but ? Assurer la cohérence des données entre les applications de l'entreprise (ERP, applications propriétaires, etc.), chacune d'elles ayant des formats de données spécifiques. La fiabilité et la robustesse des échanges, la pérennité de la solution et de son éditeur, et finalement, de la simplification de l'administration du système d'information ont constitué les critères principaux d'une solution. D'autre part, l'utilisation de normes n'était pas un facteur déterminant.

Les analystes ont annoncé un futur brillant pour XML. « *Cependant, mis à part le secteur de l'industrie des composants électroniques, dans laquelle triomphe le standard RosettaNet, XML n'a pas le succès annoncé. Les erreurs de l'EDI se reproduisent pour XML. Chacun a créé son standard. Ainsi, les entreprises ont adopté une position d'attente au sujet du respect de langage XML, et l'EDI reste utilisé pour les échanges « BtoB » dans les principales branches de l'industrie (automobile ou grande distribution). L'apparition d'une norme générique, un tel ebXML, a pu changer ce fait établi* », affirme Arnaud Vantichelen, consultant manager de Sterling Trades en France. « *Bien que nous installions une solution d'intégration prête à dialoguer selon les standards XML, comme ceux de l'EDI, nos partenaires utilisent l'EDI comme support des échanges entre les sociétés, parce qu'ils ne sont pas encore prêts pour XML* », indique Scott Annan, personne en charge de E-trade pour Lexmark en Europe.

## 4.1. Le couple XML-XSLT

Les solutions courantes d'intégration intra et inter-entreprises, se regroupant sous la bannière de la gestion des processus métier, pourraient laisser penser que le langage XML devrait jouer un plus grand rôle. Certains éditeurs de solutions EAI l'emploient comme un format pivot, dans lequel toutes les données venant des applications sont converties avant d'être prises en considération par le courtier de message.

De la même manière, des données XML traitées par l'EAI sont reconverties dans le format propriétaire à partir du format pivot. La transformation de données, convertie à l'avance dans le format pivot, est effectuée par des feuilles de style XSLT – *Extensible stylesheet language*. Ces feuilles de style sont complétées, si nécessaire, par les fonctions spécifiques de transformation (EDI vers XML ou XML vers EDI). Cependant, cette manière de procéder ne rassemble pas un consensus. De nouveaux entrants, tels que BEA Systems ou Microsoft sont d'accord pour utiliser XML dans les solutions EAI, parce que le couple XML-XSLT autorise la création des fonctions de transformation des formats de données à un coût inférieur. Les éditeurs, spécialistes dans l'EAI, sont eux, plus hésitant quant à l'utilisation de ce couple.

« *Aussi longtemps que les applications d'entreprise n'auront pas une interface XML, le recours à des connecteurs embarquant les règles de transformation sera justifié. Ainsi, convertir les données en laissant une application sans interface XML vers une application existante, tout en passant par un format pivot, n'apporte rien fonctionnellement, et dégrade fortement les exécutions en raison du traitement interprété des feuilles de style XSLT. Mais*

*rien n'empêche d'employer XML pour la définition de ce format pivot* », affirme Régis Mauger, directeur technique des webMethods en France, éditeur qui était un des pionniers en utilisant le langage XML dans les échanges « BtoB ».

## 4.2. Les apports intrinsèques de XML

Pour les intégrateurs, le langage XML apporte beaucoup d'avantages. « *L'utilisation d'un format pivot, basé en particulier sur XML, est essentielle pour parvenir à gérer des relations n → m entre les applications, et non seulement pour les relations « point-à-point ». Ceci permet de ne pas avoir un trop fort couplage entre les applications* », explique Franck Lechef, directeur technique de Valtech.

## 4.3. Vers la standardisation des solutions EAI

« *L'utilisation de diagrammes XML dans les solutions d'EAI devrait se développer bientôt car que les éditeurs de logiciel adaptent leurs produits à XML. La version 12 d'Oracle et la version 9 de l'ERP de PeopleSoft devraient être basées sur XML. De plus, cette utilisation des schémas XML devrait entraîner une chute des prix des solutions EAI, toujours en grande partie inaccessible pour les PME. D'autre part, les feuilles de style XSLT sont malheureusement insuffisantes à l'usage, ce qui implique souvent de recourir à d'autres langages de programmation et, par conséquent, avec des solutions plus ou moins spécifiques. Pour changer la solution EAI, il est alors nécessaire d'inviter des outils d'importation entre les cadres de référence* », souligne Henry Peyret, analyste sénior chez GigaGroup.

Un second axe de standardisation est en train de naître avec l'inclusion, dans la plate-forme J2EE, des normes facilitant la communication entre les applications J2EE et les applications d'entreprise. Il agit, d'une part, de JMS, qui définit une interface standard d'accès dans Java pour les MOM et, d'autre part, de JCA, pour le raccordement informatique aux applications d'entreprise.

JMS est la première norme ouverte définie dans le domaine du MOM. Il détermine l'interface d'accès à un MOM, depuis la plate-forme J2EE, pour la mise en file d'attente de messages (popularisé par le MOM MQSeries d'IBM) et le mécanisme de publication/abonnement (popularisé par le MOM Rendez-vous de Tibco). Cependant, JMS ne fixe pas les caractéristiques intrinsèques du MOM utilisé (robustesse, exécution, traitement de pannes, nombre de plates-formes soutenues, ou outils d'administration). Il a, néanmoins, un impact significatif sur le marché du MOM, et la majorité des éditeurs proposent une variation de celui-ci, comme IBM, iPlanet, Tibco ou webMethods. D'autres éditeurs (beaucoup plus petit) ont également placé le MOM sur le marché conformément à JMS, comme Sonic MQ, Progress Software ; ou SmartSockets pour JMS de Talarian.

Conçu initialement pour normaliser des connecteurs reliant les serveurs d'applications aux applications d'entreprise, JCA semble faire de grands pas inattendus dans le domaine de l'EAI. En effet, c'est la seule norme qui définit l'architecture des connecteurs reliant les applications à intégrer et la solution d'EAI. « *JCA est une des normes qui compteront, aussi bien que XML et les Services Web* », affirme Joanne Correia, analyste du cabinet Gartner Dataquest dans une étude de juillet 2001.

Même si JCA est encore immature, les éditeurs de solutions EAI donnent une opinion en sa faveur. Ainsi, IBM a annoncé WebSphere Adapters, son offre des connecteurs en conformité avec JCA.

Pour sa part, Sybase-Néon adoptera JCA en tant que connecteurs standards pour son offre eBiz Integrator. « Nous préparons la structure de réception des connecteurs JCA dans notre solution d'intégration. Cela permettra à nos clients de profiter d'un ensemble de connecteurs bien plus significatifs. Nous adapterons nos connecteurs pour les conformer au standard JCA. Cependant, nous considérons que la valeur d'une solution EAI n'est pas mesurée seulement avec le catalogue de connecteurs, mais également, et en particulier, aux capacités du serveur d'intégration et de la gestion des processus complexes », indique Isabelle Genestoux.

## 4.4. Un engagement sur la durée et l'évolution

De son côté, Tibco Software a créé TIB/Adapter pour EJB, conçu selon l'architecture JCA, et propose un ensemble de connecteurs JCA. D'autres éditeurs, de tels webMethods ou Attunity, voient dans JCA les moyens de relier les serveurs d'applications à leurs solutions EAI. L'apparition de JCA et la maturité des serveurs d'applications J2EE, en ce qui concerne la montée en charge et la tolérance aux fautes, mènent un nombre de plus en plus important d'acteurs à migrer leur solution d'EAI au-dessus d'un serveur d'applications J2EE. Ainsi, BEA Systems établit WebLogic Integration, sa nouvelle solution d'intégration d'applications, au-dessus de son serveur d'applications J2EE, équipé d'une structure de réception basée sur des connecteurs JCA. IBM fait converger son offre d'EAI avec son serveur d'applications. Sterling Commerce a utilisé le serveur d'applications de BEA Systems pour construire Sterling Integrator, sa nouvelle solution EAI. Iona Technologies a migré les solutions acquises de Software AG (EAI) et de Net Fish (B2Bi) au-dessus de son serveur d'applications J2EE, et adopteront très bientôt JCA. Finalement, Software AG, a son propre serveur d'applications J2EE pour l'infrastructure de sa solution EAI.

Le développement de solutions EAI au-dessus d'un serveur d'applications n'est cependant pas la prérogative du monde J2EE. Ainsi, Microsoft a créé BizTalk Server au-dessus de son serveur d'applications COM+. « *La migration des offres d'EAI sur des serveurs d'applications n'offre aucune nouvelle fonction pour les entreprises. D'autre part, elle apporte un engagement sur la durée et la capacité d'évolution de la solution EAI pour ces entreprises. L'émergence de JCA accélère cette migration, en offrant un ensemble de connecteurs beaucoup plus vastes aux utilisateurs de l'EAI des entreprises. De plus, la migration des offres EAI au-dessus d'un serveur d'applications a l'avantage, pour les éditeurs, de fortement réduire leur recherche et leurs coûts de développement, ce qui doit leur permettre de se consacrer à la création des fonctions EAI fonctionnellement plus riches* », note Franck Lechef.

En conclusion, l'émergence de technologies de Services Web mène à la fabrication de standards dans les solutions EAI. Leur rôle possible ne fait pas encore l'unanimité, du fait de la prise en compte des problèmes de performance.

Cependant, les technologies de Services Web devraient faciliter l'interopérabilité des plates-formes telles que J2EE, Java et Microsoft.NET. Les portails devraient ainsi donc être de gros consommateurs de Services Web.

# Annexe 3 :
# Le π-Calcul

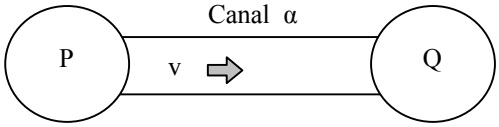
# Annexe 3 : Le π-Calcul

## 1. Introduction

Nos travaux se basent sur un langage de description d'architectures. Ce type de langage est un langage formel permettant l'analyse de l'architecture décrite et la génération d'applications logicielles sûres. C'est pourquoi nous faisons appel aux algèbres de processus. Ces algèbres constituent des méthodes formelles pour modéliser des interactions entre processus. Elles permettent, en construisant un modèle mathématique reprenant certaines descriptions du modèle d'analyse de l'application (celles relatives aux interactions), de garantir la cohérence du modèle et la conformité du programme.

Il existe différentes algèbres de processus comme CSP [Hoa 85], CCS [Mil 89], π-Calcul [Mil 99], etc. Nous avons décidé de nous baser sur le π-Calcul, car cette algèbre introduit le concept de mobilité, c'est-à-dire la possibilité de faire évoluer dynamiquement la topologie des applications. Il existe différentes versions du π-Calcul. Nous avons décidé d'utiliser le π-Calcul synchrone, polyadique. Ces nuances seront expliquées dans le paragraphe 2.5.1, 2.5.2.

Le π-Calcul permet de représenter des communications entre des processus. Par exemple, la figure ci-dessous illustre un processus *P* qui émet une valeur *v* au processus *Q* à travers un canal de transmission $\alpha$.




## 2. Les principaux opérateurs du π-Calcul

### 2.1. Convention

Les lettres majuscules définissent des processus. Un agent représente un ensemble de processus.

Les lettres minuscules définissent des variables, des valeurs ou des canaux de transmission.

### 2.2. La syntaxe des opérateurs

Dans cette partie, nous présentons les opérateurs suivants :

μ • P : préfixation d'un processus par une action. $\mu$ peut être :
- x(y) : préfixation positive. Elle dénote la réception de la variable *y* sur le canal *x*.
- $\bar{x}\,y$ : préfixation négative. Elle dénote l'émission de la valeur *y* sur le canal *x*.
- τ : préfixation silencieuse. C'est une action inobservable (action interne).

P | Q : le parallélisme, met en parallèle deux processus.

$(\nu\ x)\ P$ : la restriction. Le processus *P* ne peut pas communiquer sur le canal *x* avec l'extérieur (seule la communication interne sur ce canal est autorisée).
$P + Q$ : la somme indéterministe, permet le choix entre deux processus.
$[x = y]\ P$ : l'appariement de forme « *matching* ». Exemple : cet agent se comporte comme *P* si *x* et *y* sont identiques.
$A\ (x_1, \ldots, x_n) \overset{def}{=} P$ : définition d'un processus. Elle permet d'exprimer la récursivité.

La structure du π-Calcul peut être définie de manière inductive, c'est-à-dire qu'elle va être construite autour du processus **0**. Puis, nous allons pouvoir préfixer ce processus par des actions et le mettre en parallèle avec un autre processus, etc.

La syntaxe d'un processus *P* peut donc être résumée par :
$P ::= \mathbf{0} \mid x(y)\bullet P \mid \bar{x}\, y\bullet P \mid \tau\bullet P \mid P_1|P_2 \mid (\nu\ x)P \mid P_1+P_2 \mid [x = y]P \mid A\ (x_1, \ldots, x_n)$

L'opérateur de choix est prioritaire sur l'opérateur parallèle qui est lui-même prioritaire sur les opérateurs de restriction, de préfixation et d'appariement de forme.

Par exemple : $(\nu\ x)\ P \mid \tau \bullet Q + R$ se lit : $( ( ( (\nu\ x)\ P ) \mid ( \tau \bullet Q ) ) + R )$.

## 2.3. Sémantique

### 2.3.1. Le processus inactif

***0*** représente un processus inactif. C'est un constituant de base de tout processus, car un processus finira par ne plus rien faire (sauf dans le cas de la récursivité).

### 2.3.2. La préfixation

L'expression $P \equiv \bar{\alpha}\, v \bullet \mathbf{0}$ décrit un processus *P* qui émet la valeur *v* par le canal $\alpha$ puis s'arrête ; l'expression *puis* est traduite par l'opérateur •.

L'expression $Q \equiv \alpha\ (x) \bullet R$ décrit un processus *Q* un processus qui reçoit la variable *x* par le canal $\alpha$, puis continue en se comportant comme un processus *R* (pouvant contenir des occurrences de *x*) ; l'expression *puis* est traduite par l'opérateur •.

### 2.3.3. Le parallélisme

Pour que deux processus puissent communiquer, il faut qu'ils soient en parallèle, ceci est noté : *P* | *Q*. A un moment donné *v* va être transmis par le canal $\alpha$ :

$$\bar{\alpha}\, v \bullet \mathbf{0} \mid \alpha\ (x) \bullet R \bullet \mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid R\{v/x\} \bullet \mathbf{0}$$

Après une action inobservable $\tau$, le système a évolué en transmettant *v* du processus *P* au processus *Q*. En effet, les *x* libres qui se trouvent dans le processus *R* seront substitués par des *v*. (La notion « variables libres, variables liées » est expliquée dans le paragraphe sur la restriction)

Les propriétés suivantes sont observées :
$$P \mid \mathbf{0} \equiv P, \qquad P \mid Q \equiv Q \mid P, \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \equiv P \mid Q \mid R$$

### 2.3.4. La restriction

Considérons le système suivant : P | Q | R avec :

$$P \equiv \overline{\alpha}\, v \bullet \mathbf{0} \qquad \text{et} \qquad Q = R \equiv \alpha\,(x) \bullet \mathbf{0}$$

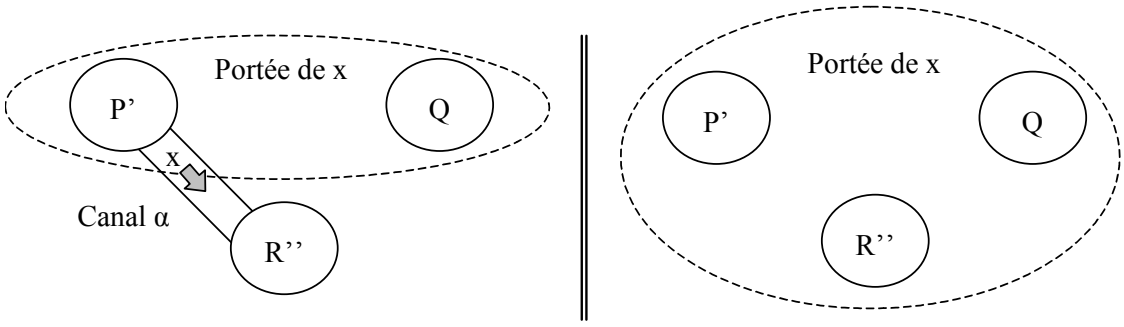
Alors *P* peut transmettre la valeur v aussi bien à *Q* qu'à *R*.

Pour restreindre la communication seulement entre *P* et *Q* il faudra faire la restriction *v* sur le canal *α*, de la manière suivante : *(v α ) (P | Q) | R*.

La communication sur le canal *α* ne pourra se faire qu'entre *P* et *Q* ; c'est-à-dire que *R* n'a pas accès au même canal *α* que celui utilisé par *P* et *Q*. Dans cet exemple, les occurrences du canal *α* dans les processus *P* et *Q* sont dites "variables liées", et dans *R α* est une "variable libre".

Une extension de portée « Scope extrusion » est possible. Par exemple :
Si *P* possède un canal privé *x* avec *Q* : *(vx) (P | Q)*,
et souhaite le transmettre à *R* : *(vx) (P | Q | R),* alors il y a une extension de portée.

Ex : $(\nu x)\,(\overline{\alpha}\, x \bullet P' \mid Q) \mid \alpha\,(y) \bullet R' \xrightarrow{\tau} (\nu x)\,(P' \mid Q \mid R'\{x/y\})$




Toutefois, la substitution de *y* par *x* : *R'{x/y}* n'est possible que si *R* ne possède pas déjà un canal *x*, sinon il faut renommer le canal privé *x* pour préserver la différence avec le canal public. Nous étendons alors la portée de *x* aux trois processus *P'*, *Q* et *R''* (avec R'' = R'{x/y} : *R''* est le processus résultant de la substitution de *y* par *x* dans *R'*).

### 2.3.5. La somme indéterministe

Un autre opérateur que nous devons introduire est l'opérateur de choix : +

$$(\alpha\,(x) \bullet R + \beta\,(y) \bullet S) \mid \overline{\alpha}\, v \bullet \mathbf{0} \mid \overline{\beta}\, w \bullet \mathbf{0}$$

Ce système peut évoluer :

- soit vers : $R\{v/x\} \mid \mathbf{0} \mid \overline{\beta}\, w \bullet \mathbf{0}$
- soit vers : $S\{w/y\} \mid \overline{\alpha}\, v \bullet \mathbf{0} \mid \mathbf{0}$

Nous pouvons observer les propriétés suivantes :

$$P + P \equiv P, \quad P + Q \equiv Q + P, \quad (P + Q) + R \equiv P + (Q + R) \equiv P + Q + R$$

### 2.3.6. L'appariement de forme

Pour spécifier le choix, nous pouvons utiliser l'appariement de forme « *matching* » :

$$[x = a]P + [x = b]Q$$

L'agent se comportera comme *P* si *x* et *a* sont identiques, comme *Q* si *x* et *b* sont identiques et dans les autres cas comme ***0***.

Il existe aussi le non appariement de forme « *mismatching* » :

$$[x \neq a]P$$

L'agent se comportera comme *P* si *x* et *a* sont différents.

Ainsi, nous pouvons facilement traduire le « si, alors, sinon » :

$$[x = a]P + [x \neq a]\,R \qquad (\text{ si } x = a \text{ alors } P \text{ sinon } R\ )$$

### 2.3.7. La notation d'un processus récursif

Un processus peut être exprimé de la manière suivante :

$$A\ (x_1, \ldots, x_n) \overset{def}{=} P$$

c'est-à-dire qu'un processus peut être remplacé par un nom avec ses canaux libres en paramètres.

L'avantage principal de cette notation est la possibilité de décrire des processus récursifs. Par exemple, un processus qui ne ferait qu'émettre indéfiniment la valeur *a* sur le canal *x* serait :

$$\text{Emission}\ (x) \overset{def}{=} \bar{x}\,a \bullet \text{Emission}\ (x)$$

## 2.4. La particularité du π-Calcul

Ce qui fait la force du π-Calcul c'est qu'un nom d'un canal peut être transmis à travers un autre canal, contrairement aux algèbres de processus tels que CSP ou CCS.

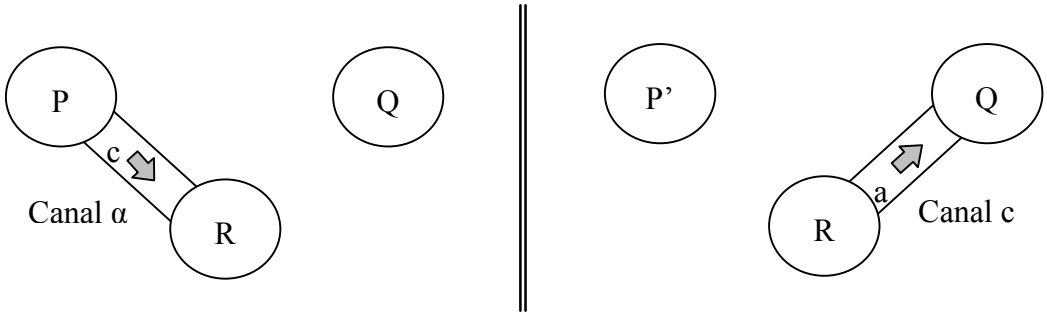
Par exemple, si nous avons les processus *P*, *Q*, et *R* en parallèle : P | Q | R

avec $P \equiv \bar{\alpha}\,c \bullet P'$ $\quad Q \equiv c\ (y) \bullet Q'$ $\quad$ et $R \equiv \alpha\ (x) \bullet \bar{x}\,a \bullet R'$

alors $P \mid Q \mid R \equiv \bar{\alpha}\,c \bullet P' \mid c\ (y) \bullet Q' \mid \alpha\ (x) \bullet \bar{x}\,a \bullet R'$

$$\xrightarrow{\tau} P' \mid c\ (y) \bullet Q' \mid \bar{c}\,a \bullet R'$$

$$\xrightarrow{\tau} P' \mid Q' \mid R'$$

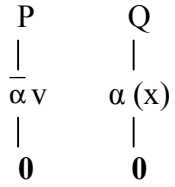



A ce moment, la topologie du système est modifiée en cours d'exécution.

## 2.5. Version du π-Calcul

### 2.5.1. π-Calcul synchrone et asynchrone

Le π-Calcul synchrone est celui décrit jusqu'à présent, c'est-à-dire que la communication se fait sous forme de rendez-vous :




Si *P* arrive le premier au rendez-vous, il va attendre que *Q* arrive à la réception de la variable *x*, pour émettre la valeur *v*. Si *Q* était arrivé le premier, il aurait attendu de la même façon *P*.

Le π-Calcul asynchrone se veut non bloquant pour l'émission. Dans l'exemple ci-dessus *P* aurait continué son chemin après avoir émis *v*. Par contre, *Q* aurait réagi de la même manière que précédemment.

### 2.5.2. π-Calcul polyadique

Le π-Calcul polyadique permet de transmettre un tuple de valeurs lors d'une interaction.

Ex : $P \mid Q$ avec $P \equiv \overline{\alpha}\langle x, y, z\rangle \bullet \mathbf{0}$ et $Q \equiv \alpha\,(u, v, w) \bullet \overline{v}\langle u, w\rangle \bullet \mathbf{0}$

$P \mid Q \equiv \overline{\alpha}\langle x, y, z\rangle \bullet \mathbf{0} \mid \alpha\,(u, v, w) \bullet \overline{v}\langle u, w\rangle \bullet \mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \overline{y}\langle x, z\rangle \bullet \mathbf{0}$

Remarquons que la notation pour l'émission en π-Calcul polyadique est différente du π-Calcul monadique : le tuple est entouré de parenthèses pointues.

# Annexe 4 :
# Les patrons d'architecture logicielle

# Annexe 4 :
# Les patrons d'architecture logicielle

## 1. Introduction

Un style architectural, comme défini par Shaw et Garlan [ScG 96], et conçu par d'autres [BMR 96] est une description de types de composant et de leur topologie, qui inclut une description de patrons de données et d'interaction de contrôle entre ces composants. Les styles architecturaux fournissent également une description informelle des avantages et inconvénients de l'utilisation de ces styles.

En 1999, la notion d'ABAS – *Attribute-Based Architectural Style* – fut introduite. Le domaine d'application de l'utilisation de ces ABAS est de faire en sorte que les styles architecturaux servent de fondation, de patrons, pour un raisonnement plus précis concernant la conception d'architectures [KlK 99]. Ceci est réalisé en associant explicitement un cadre de raisonnement (qu'il soit qualitatif ou quantitatif) à un style architectural. Ces cadres de raisonnement sont basés sur des modèles d'attributs qualité spécifiques.

Les parties ci-dessous présentent la notion d'attribut qualité, puis les différents ABAS de référence sur lesquels le concepteur d'architectures peut s'appuyer pour élaborer ses architectures.

## 2. La notion d'attribut qualité

Les ABAS montrent comment raisonner au sujet des décisions architecturales en ce qui concerne un attribut qualité spécifique tel que la performance, la sécurité ou la fiabilité. Cependant, un préalable pour construire et employer les ABAS est d'avoir une caractérisation précise de l'attribut qualité. Par exemple, la compréhension d'un style du point de vue de la modifiabilité exige une compréhension de la façon de mesurer et d'observer cette modifiabilité et une compréhension de la manière dont les décisions architecturales impactent cette mesure.

Nous divisons l'information relative à un attribut qualité en trois catégories :

- stimuli externes,
- décisions architecturales,
- réponses.

## 3. « Synchronization ABAS »

### 3.1. Description du problème

Pour le « *synchronization ABAS* », nous considérons un processeur simple sur lequel de multiples processus résident, chacun d'eux exécute des calculs sur leur propre flot de données d'entrée. Chaque sortie finalisée du système doit être produite dans un intervalle de temps spécifié après l'arrivée d'une entrée et après que tous les calculs aient été exécutés. Nous nous référerons aux données d'entrée comme des messages. La condition est que chaque message doit alors être traité complètement avant un temps de latence borné – une date limite.

Le dispositif intéressant du « *synchronization ABAS* » est qu'il doit traiter le problème de l'accès à une ressource partagée, pour laquelle des processus multiples sont en concurrence. La propriété centrale de cet ABAS pour le raisonnement est : comment cet accès à la ressource partagée est géré, et comment la gestion de cet accès affecte la performance, le temps de latence global. L'« *ABAS pipelines* » concurrent se focalise sur des processus multiples attaquant un unique processeur en utilisant une planification préemptive, tandis que cet ABAS se concentre sur des processus multiples exigeant l'accès mutuellement exclusif à une ressource partagée telle que des données partagées.

## 3.2. Critère pour le choix de cet ABAS

Cet ABAS sera approprié si votre problème a en soi des conditions de performance en temps réel et se compose des processus multiples sur un unique processeur qui partagent une ressource.

## 3.3. Mesures des stimuli et des réponses

Nous caractérisons les stimuli importants et leurs réponses mesurables et contrôlables comme suit :

- **Stimuli :** au moins deux flots d'entrée périodiques ou sporadiques,
- **Réponse :** temps de latence global, pires cas d'exécution.

## 3.4. Style architectural

Le style correspondant à cet ABAS est montré par la figure ci-dessous dans une vue concurrente illustrée sur une vue matérielle. Dans cet ABAS, il y a un unique processeur et un ensemble de processus avec des propriétés connues (ou estimées) associées, énumérées dans le tableau ci-après, qui transforment des flots d'entrée en des flots de sortie. Certains de ces processus doivent se synchroniser pour partager une ressource commandée par S, le processus « serveur ».

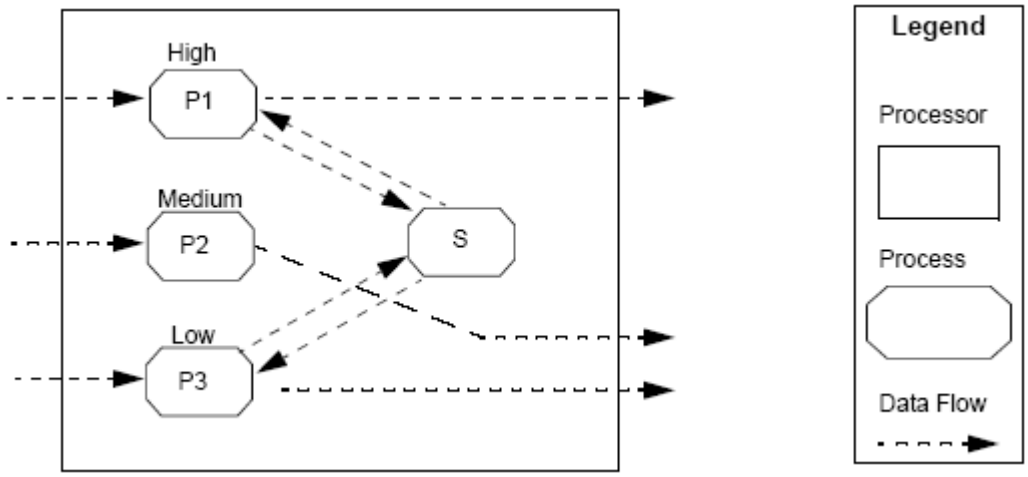



*Figure A.17. : Le « Synchronization ABAS »*

Le tableau de la page suivante décrit les décisions architecturales qui sont nécessaires pour créer un modèle analytique de latence global et des pires cas pour cet ABAS.

| **Performance Architectural Parameters** |
|---|
| topology: *star* |
| preemption policy: *priority based* |
| execution time for each process associated with processing each input: $C_i$ |
| period associated with each process: $T_i$ |
| scheduling discipline: *fixed priority* |
| synchronization protocol including:<br>• *the queuing discipline (e.g., FIFO or priority) for the server process*<br>• *how the priority is managed during the critical section (e.g., the section of code during which other processes are locked out)* |

*Figure A.18. : Décisions architecturales pour le « Synchronization ABAS »*

# 4. « Data Indirection ABAS »

## 4.1. Description du problème

Cet ABAS est caractérisé par des producteurs et des consommateurs des données partagées ayant connaissance de l'existence de chacun d'entres eux et des détails de leurs implémentations. Ceci est réalisé en interposant un intermédiaire – un composant et/ou un protocole – entre les producteurs et les consommateurs des données élémentaires partagées. Le principe général permet d'augmenter la modifiabilité en réduisant le degré de couplage des données ou des commandes entre les divers composants. Dans cet ABAS, le degré de couplage est réduit en ayant un intermédiaire – typiquement un dépôt de données partagées (*Data Repository*) – couplé aux producteurs et aux consommateurs, et par conséquent les découplant les uns des autres.

## 4.2. Critère pour le choix de cet ABAS

Cet ABAS sera approprié si vous prévoyez des changements au niveau des producteurs et des consommateurs des données, y compris l'addition de nouveaux producteurs et consommateurs. Si ces changements sont fréquents et assez importants pour justifier un souci concernant le coût de modification de l'architecture, cet ABAS est approprié.

## 4.3. Mesures des stimuli et des réponses

Nous caractérisons les stimuli importants et leurs réponses mesurables et contrôlables comme suit :

- **Stimuli :**
  - un nouveau producteur ou consommateur des données,
  - une modification au niveau d'un producteur ou consommateur des données existant,
  - une modification interne au niveau du dépôt de données,
- **Réponse :** le nombre de composants, d'interfaces, et de connexions supplémentaires, supprimés, et modifiés, avec une caractérisation de la complexité de ces changements/suppressions/modifications.

## 4.4. Style architectural

Le schéma ci-dessous montre la structure générique de cet ABAS. Sa topologie est une étoile, avec le dépôt situé au centre. Les producteurs et les consommateurs se situent à la périphérie de ce dépôt.

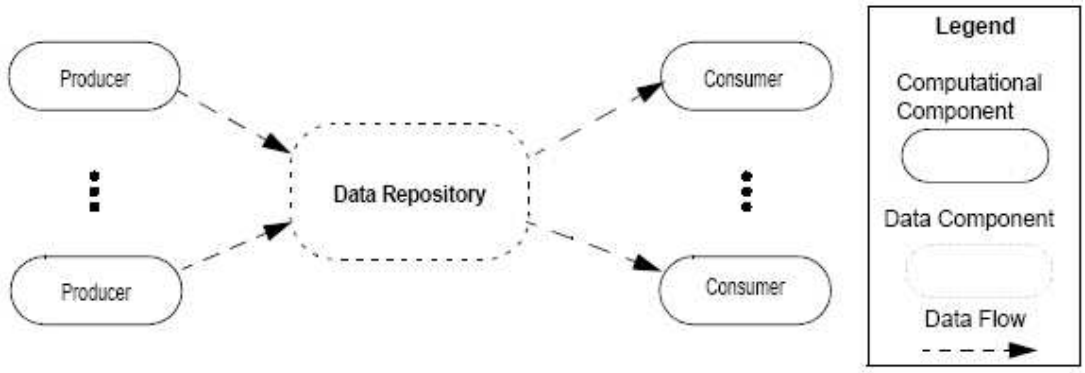



*Figure A.19. : Le « Data Indirection ABAS »*

Le dépôt de données peut être un endroit qui est connu à la fois par les producteurs et les consommateurs (par exemple, un dossier ou un secteur global de données) ou ce peut être un composant informatique séparé (un tableau noir qui est accueilli dans un processus séparé, potentiellement même sur un ordinateur séparé). La seule contrainte concernant le dépôt est qu'il doit pouvoir contenir des données. Le dépôt de données est un lieu de stockage de données persistant tel qu'un secteur de données partagé dans la mémoire, un dossier, ou une base de données.

En plus du dépôt, il y a un certain nombre de producteurs et de consommateurs de données. Un composant unique peut être soit un producteur, soit un consommateur de données. Le dépôt a une disposition des données spécifique (une structure sous forme de dossier ou schéma) et un ensemble de types de données qui sont connus par tous les producteurs et consommateurs. Cette disposition demeure conforme pendant une exécution simple du système. Les producteurs placent leurs données dans le dépôt en vertu du fait qu'ils connaissent les détails de la disposition du dépôt ; les consommateurs recherchent pareillement les données au sein du dépôt. Les façons dont la performance et la gestion des conflits d'accès sont contrôlées (par exemple, la politique et les mécanismes pour déterminer qui obtient le droit de mettre à jour le dépôt et quand ceci peut être réalisé) sont en dehors de la portée de cet ABAS.

Le tableau ci-après présente les paramètres et les décisions architecturales relatifs à cet ABAS.

| **Modifiability Architectural Parameters** |
|---|
| topology: *star* |
| persistence of data: *persistent* |
| client knowledge of data schema: *complete knowledge* |
| activeness of repository: *passive* |

*Figure A.20. : Décisions architecturales pour le « Data Indirection ABAS »*

Le « comportement » du dépôt, la persistance des données, et la connaissance du client concernant le schéma de données sont des décisions de conception qui nous permettent de différencier plusieurs sous-styles du « *Data Indirection ABAS* » comprenant le « *Abstract Data Repository ABAS* » et le « *Publisher/Subscriber ABAS* ». Ces derniers sont décrits ci-après.

## 4.5. « Abstract Data Repository Sub-ABAS »

### 4.5.1. Description du problème

Ce sous-ABAS étend le « *Data Indirection ABAS* » non seulement en ayant des producteurs et des consommateurs des données partagées qui ont la connaissance de l'existence de chacun d'entres eux et des détails de leurs implémentations, mais également en gardant les détails de l'implémentation du dépôt de données partagées comme « secret » pour ces producteurs et ces consommateurs. Ce secret est rendu possible grâce à l'implémentation d'une interface abstraite du dépôt de données. Cette interface abstraite réduit le degré de couplage entre les producteurs et les consommateurs de données au-delà de ce que le « *Data Indirection ABAS* » réalise.

### 4.5.2. Critère pour le choix de cet ABAS

Cet ABAS devrait être choisi si vous prévoyez que les producteurs de données changeront le format interne des données qu'ils produisent. Les conséquences de tels changements, si ces changements ne sont pas pris en compte, seront traduites au niveau du dépôt de données, et par conséquent au niveau de tous les consommateurs de ces données. De plus, de tels changements pourraient affecter la disposition du dépôt de données de manière non-locale, affectant de ce fait des producteurs et des consommateurs d'autres données s'il n'y avait pas une interface abstraite. Employer cet ABAS protégera donc les producteurs et les consommateurs de données contre des changements à leurs niveaux et au niveau du dépôt.

### 4.5.3. Mesures des stimuli et des réponses

Nous caractérisons les stimuli importants et leurs réponses mesurables et contrôlables comme suit :

- **Stimuli :**
  - un nouveau producteur ou consommateur des données,
  - une modification au niveau d'un producteur ou consommateur des données existant,
  - une modification interne au niveau du dépôt de données,
- **Réponse :** le nombre de composants, d'interfaces, et de connexions supplémentaires, supprimés, et modifiés, avec une caractérisation de la complexité de ces changements/suppressions/modifications.

### 4.5.4. Style architectural

Le schéma ci-après décrit la structure architecturale pour le « *Abstract Data Repository sub-ABAS* ». Dans cet ABAS, il y a un certain nombre de producteurs et de consommateurs de données. Un composant unique peut être soit un producteur, soit un consommateur des données. Ces producteurs et consommateurs ne savent rien sur l'existence des autres, communiquant uniquement par les "méthodes" que l'interface abstraite du dépôt de données fournit. Ces méthodes commandent la création, la suppression, la modification, et la récupération des données du dépôt. Les producteurs envoient leurs données à un dépôt de

données abstrait, mais n'ont aucune connaissance quant à la façon dont ces données seront stockées ou recherchées par la suite.

Le dépôt contient ces données pour les consommateurs qui peuvent les accéder par l'intermédiaire des méthodes fournies, avec encore une fois, une ignorance totale de la disposition et des algorithmes fondamentaux des données dans le dépôt. Le dépôt de données abstrait stocke les données produites jusqu'à ce qu'elles soient explicitement supprimées.

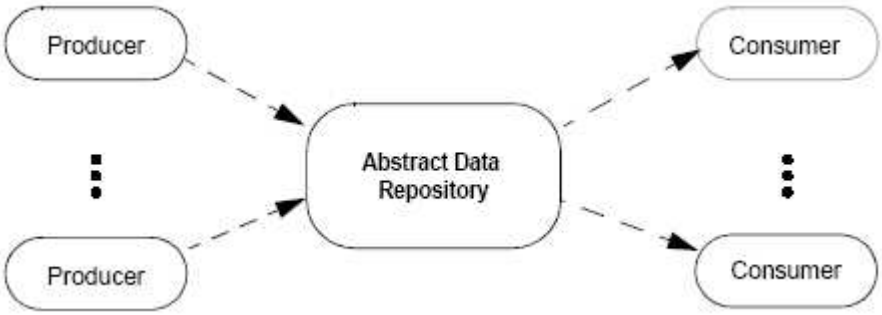



*Figure A.21. : Le « Abstract Data Repository Sub-ABAS »*

Le style architectural de ce sous-ABAS hérite des décisions architecturales du « *Data Indirection ABAS* ». Cependant, étant un sous-ABAS, certaines des décisions architecturales sont gérées différemment. Vous pouvez considérer ces relations entre des ABAS et des sous-ABAS comme des relations d'héritage en UML. Ainsi, le style est identique à moins qu'un paramètre ne soit explicitement remplacé dans le sous-style.

Le tableau ci-après présente les paramètres et les décisions architecturales relatifs à l'« *Abstract Data Repository sub-ABAS* ».

| **Modifiability Architectural Parameters** |
|---|
| topology: *star* |
| persistence of data: *persistent* |
| client knowledge of data schema: *no knowledge* |
| activeness of repository: *passive* |

*Figure A.22. : Décisions architecturales pour le « Abstract Data Repository Sub-ABAS »*

## 4.6. « Publish/Subscribe Sub-ABAS »

### 4.6.1. Description du problème

Ce sous-ABAS permet de synchroniser automatiquement l'état des producteurs et des consommateurs de données. Les producteurs de données sont appelés les « éditeurs » (*Publishers*), et les consommateurs les « abonnés » (*Subscribers*). Quand un éditeur publie un nouveau morceau de données, tous les abonnés en sont notifiés et reçoivent automatiquement la donnée. Plus concrètement, le dépôt n'accepte pas 'simplement' passivement des demandes de stockage et de recherche de données, il gère ces dernières en fonction des consommateurs de données qui sont intéressés par telles données élémentaires et informe ces consommateurs lorsque de nouvelles données appropriées ont été déposées dans le dépôt. Ce sous-ABAS

étend le « *Data Indirection ABAS* » et le « *Abstract Data Repository sub-ABAS* » en maintenant non seulement les producteurs et les consommateurs ignorants des données qu'ils partagent et des détails de l'implémentation des données partagées du dépôt, mais en rendant le « comportement » du dépôt actif et le stockage des données de manière non-persistante dans le dépôt.

### 4.6.2. Critère pour le choix de cet ABAS

Cet ABAS devrait être choisi si vous prévoyez que toutes les conditions suivantes peuvent être vérifiées :

- Les producteurs de données changeront le format des données qu'ils produisent.
- Le nombre et l'identité des producteurs et/ou des consommateurs d'une donnée élémentaire particulière sont inconnus ou sont susceptibles de changer.
- L'ordre chronologique d'« exécution » entre les producteurs et les consommateurs des données est inconnu ou est sujet à de fréquents changements.
- Il n'y a aucune dépendance temporelle complexe ou de date limite temps réel liée à la production et à la consommation des données.

Les troisième et quatrième points sont les plus cruciaux pour ce sous-ABAS. Le « *Publish/Subscribe sub-ABAS* » dissocie les producteurs et les consommateurs de données en termes de leurs identités (comme avec le « *Data Indirection ABAS* »), de leurs « endroits », de la connaissance mutuelle de leur existence, et du format interne de leurs données partagées (comme avec le « *Abstract Data Repository sub-ABAS* »).

Ce sous-ABAS permet de synchroniser les producteurs et les consommateurs en fonction de l'état de leurs données, mais cette synchronisation n'est pas étroitement contrôlée par les producteurs ou les consommateurs. Ainsi, cet ABAS est inadéquat pour des applications où des dépendances temporelles complexes existent (par exemple, cette donnée élémentaire doit être traitée par le consommateur B exactement 0,05 secondes après qu'elle est traitée par le consommateur A), ou lorsque des dates limites temps réel existent (par exemple, toutes les données de chaque dossier doivent être traitées en 0,1 secondes).

### 4.6.3. Mesures des stimuli et des réponses

Nous caractérisons les stimuli importants et leurs réponses mesurables et contrôlables comme suit :

- **Stimuli :**
  - un nouveau producteur ou consommateur des données,
  - une modification au niveau d'un producteur ou consommateur des données existant,
  - une modification interne au niveau du dépôt de données,
  - un changement au niveau du temps de production ou de consommation des données,
- **Réponse :** le nombre de composants, d'interfaces, et de connexions supplémentaires, supprimés, et modifiés, avec une caractérisation de la complexité de ces changements/suppressions/modifications.

### 4.6.4. Style architectural

Le style architectural pour cet ABAS est présenté ci-dessous. Les types de composants sont des producteurs de données (*publishers*) et des consommateurs (*subscribers*). Un composant unique peut être un producteur et un consommateur. Les connexions entre les composants

sont représentées par des flots de données, mais en réalité, ces connexions sont souvent caractérisées par des flots de données et des flots de contrôle (c'est-à-dire par l'intermédiaire d'un appel de procédure, d'un appel de procédure à distance – RPC – ou d'une invocation à distance de méthode – RMI).

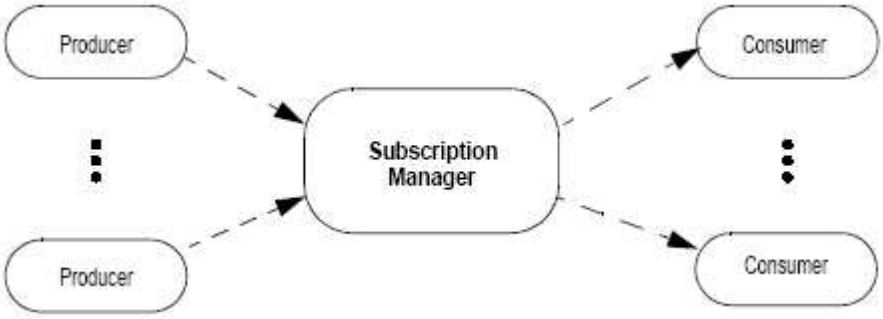



*Figure A.23. : Le « Publish/Subscribe Sub-ABAS »*

Le style architectural de ce sous-ABAS hérite des décisions architecturales du « *Data Indirection ABAS* ». Cependant, étant un sous-ABAS, certaines des décisions architecturales sont gérées différemment. La figure ci-dessous décrit les décisions architecturales relatives au « *Publish/Subscribe sub-ABAS* ».

| **Modifiability Architectural Parameters** |
|---|
| Topology: *star* |
| Persistence of data: *transient* |
| client knowledge of data schema: *no knowledge* |
| activeness of repository: *active* |

*Figure A.24. : Décisions architecturales pour le « Publish/Subscribe Sub-ABAS »*

# 5. « Layering ABAS »

## 5.1. Description du problème

Les architectures en couches ont été développées pour fournir des niveaux d'abstraction cumulatifs au-dessus d'une certaine fonctionnalité de base. Ces couches existent pour masquer les détails d'implémentation et pour permettre un certain niveau de modifiabilité à des niveaux d'abstractions différents. Des architectures en couches sont employées, par exemple, pour fournir des couches portables pour les systèmes logiciels qui doivent fonctionner sur différents systèmes d'exploitation et plates-formes matérielles, et pour fournir une abstraction commune au niveau des communications.

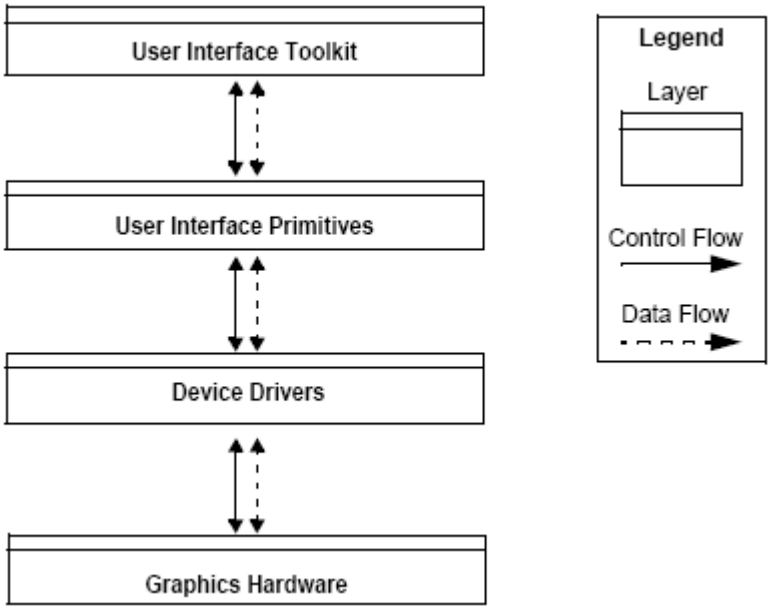



*Figure A.25. : Une représentation d'un système en couches*

## 5.2. Critère pour le choix de cet ABAS

Cet ABAS sera approprié si votre problème a de larges catégories distinguables au niveau des fonctionnalités, c'est-à-dire que ces catégories de fonctionnalités :

- sont intérieurement fortement logiques,
- sont stables en ce qui concerne les changements (c'est-à-dire, les catégories ne changent pas souvent, même si leur structure interne change),
- dépendent l'une de l'autre de manières prévisibles,
- n'ont pas de cycles de dépendance,
- ont un degré de couplage faible (en particulier, ces catégories sont en général seulement couplées tout au plus à deux autres catégories).

## 5.3. Mesures des stimuli et des réponses

Les couches permettent d'isoler des parties d'un système par rapport à d'autres. Les couches sont insérées aux endroits ou les changements sont perçus comme étant indépendants. Par exemple, des couches sont typiquement employées pour cacher les protocoles de communications ou des bases de données et pour cacher les systèmes d'exploitation ou les implémentations spécifiques d'interface utilisateur.

Nous caractérisons les stimuli importants et leurs réponses mesurables et contrôlables comme suit :

- **Stimuli :** un changement d'une couche dans le logiciel,
- **Réponses:** le nombre de couches affectées et le nombre de composants, d'interfaces, et de connexions supplémentaires, supprimés, et modifiés, avec une caractérisation de la complexité de ces changements/suppressions/modifications.

## 5.4. Style architectural

Le style architectural en couche se compose d'ensembles de composants qui implémentent de larges catégories distinguables de fonctionnalité. Chacun de ces ensembles est organisé en une couche. Les éléments de chaque couche peuvent, dans le style « *layered* », seulement interagir

avec des éléments dans leur couche ou une couche adjacente. Les éléments au sein d'une couche sont habituellement empaquetés uniformément, et sont le plus généralement des objets ou des procédures. Moins généralement une couche peut être construite avec des processus, des threads, des filtres ou d'autres types de composant. Les éléments d'une couche peuvent communiquer avec des éléments d'une couche adjacente par l'intermédiaire de connexions de données (par exemple, sockets, HTTP), mais se réalisent généralement par l'intermédiaire de connexions transférant des données et/ou des contrôles (par exemple, appel de procédure, RPC, invocation de méthode).

Le tableau ci-dessous décrit les décisions architecturales et les paramètres relatifs à cet ABAS et qui sont nécessaires pour créer un modèle analytique pour la modifiabilité.

| **Modifiability Architectural Parameters** |
|---|
| topology: *layers* |
| connectivity: *to adjacent layers only* |
| knowledge of data schema: *no knowledge* |

*Figure A.26. : Décisions architecturales pour le « Layering ABAS »*

# 6. « Simplex ABAS »

## 6.1. Description du problème

L'ABAS « *Simplex* » se concentre sur le problème de la fiabilité et de la disponibilité d'un logiciel dans des systèmes de commande [ScG 96]. En particulier, cet ABAS adresse le problème de la tolérance aux fautes au niveau logiciel présentée comme une conséquence des mises à jour des algorithmes de contrôle. Le « Simplex » adresse également le problème de la manière de tirer profit de la redondance pour augmenter la fiabilité tout en évitant des échecs logiciel en fonctionnement "standard".

Pour illustrer le problème que le « *Simplex* » adresse, considérons « le paradoxe de la mise à jour » [ScG 96]. Considérons le cas dans lequel un composant est dupliqué pour assurer sa fiabilité. Chaque duplication exécute ses calculs et envoie ses résultats à un électeur. Si les résultats ne conviennent pas (par rapport à une tolérance indiquée), l'électeur « vote pour la majorité ». Disons qu'un algorithme principal est mis à jour en fonction d'une valeur de l'ancien algorithme. Voici le paradoxe : si le nouvel algorithme est placé dans une minorité des composants dupliqués, son résultat sera voté non conforme et n'aura aucun effet ; si cet algorithme est placé dans une majorité des composants dupliqués et est défectueux, la mauvaise valeur sera utilisée.

## 6.2. Critère pour le choix de cet ABAS

Cet ABAS sera approprié si votre système :

- consiste en un fort besoin de mise à jour en ligne,
- a des besoins élevés de disponibilité, et en particulier vise à éviter des échecs en mode de fonctionnement « standard »,
- peut fonctionner dans un ou plusieurs modes dégradés, dont un peut être implémenté par l'intermédiaire d'un simple algorithme « sûr, de confiance »,
- utilise la redondance de composant pour gérer la fiabilité et la disponibilité.

## 6.3. Mesures des stimuli et des réponses

Nous caractérisons les stimuli importants et leurs réponses mesurables et contrôlables comme suit :

- **Stimuli :** un défaut arrivant au niveau du système,
- **Réponses :**
  - les niveaux du service dégradé,
  - la fiabilité et la disponibilité de chaque niveau de service.

## 6.4. Style architectural

Le « Simplex » est un style architectural qui appartient à une famille générale de styles de fiabilité qui pourraient s'appeler les styles de redondance. Le patron général pour un style de redondance est montré sur le schéma ci-dessous. Le patron, d'un point de vue de la fiabilité, est constitué de multiples composants. Les flots de données en sortie d'un ou de plusieurs composants sont envoyés à un autre composant (ou probablement à des composants) responsables de détecter des erreurs, commutant sur un composant qui lance le rétablissement du ou des composants qui ont échoué.

Le style « *Simplex* », comme représenté sur le schéma ci-après est une instance du style de redondance dans lequel les composants sont des processus. Les composants ne reçoivent pas nécessairement la même entrée ou ne produisent pas obligatoirement la même sortie.

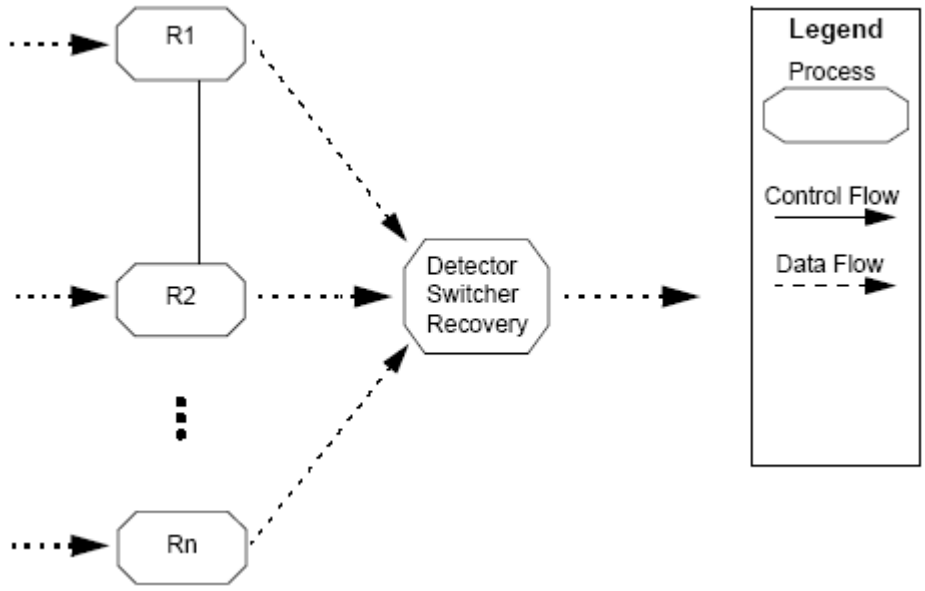



*Figure A.27. : Un style architectural de « redondance »*

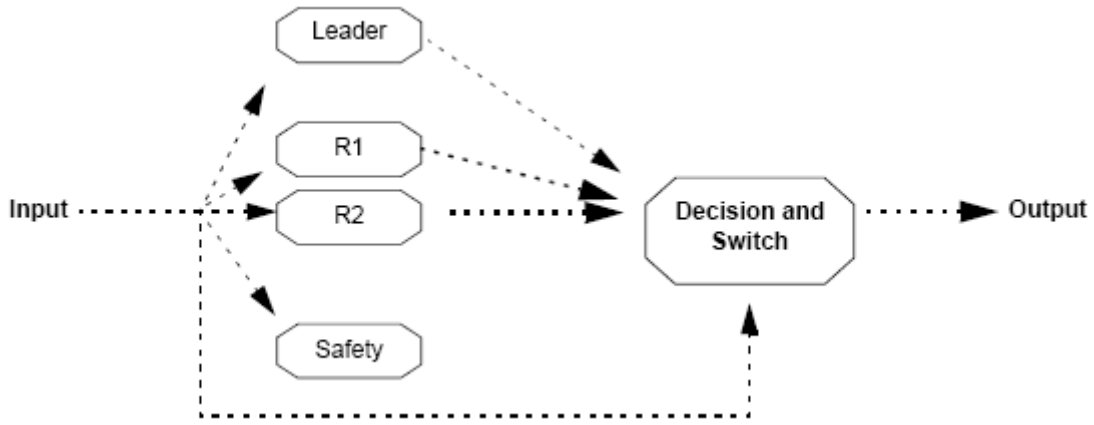



*Figure A.28. : Le style architectural « Simplex »*

La figure ci-dessous décrit les décisions architecturales relatives au « *Simplex ABAS* ».

| **Simplex Architectural Parameters** |
|---|
| Failure rates of the components (leader, safety, decision/switch, etc.): $\lambda_i$ |
| Repair rates of the components (leader, safety, decision/switch, etc.): $\mu_i$ |
| Number and type of redundant components: *m* |
| A characterization of the level of service provided by each component: *optimal/degraded* |
| Voting mechanism: *comparison each other and safety controller* |

*Figure A.29. : Décisions architecturales pour le « Simplex ABAS »*

# Annexe 5 :
# Les styles architecturaux de notre cas d'étude

# Annexe 5 :
# Les styles architecturaux de notre cas d'étude

Cette annexe présente les différents styles qui ont été définis dans le cadre de notre cas d'étude. Nous présenterons ainsi le style « intra-entreprise », le style « inter-entreprise » et le style « EAI ».

## 1. Le style intra-entreprise

Ce style intra-entreprise définit une architecture où ne figure que des composants industriels, un système de gestion de performance, un système de gestion de ressources et les différents connecteurs entre ces composants.

```
--*********************************************************--
-- LE STYLE ENTREPRISE --
--*********************************************************--
Enterprise is style extending Component where {
    -- Le style Enterprise étend le style composant --
    styles {
--*********************************************************--
-- LE STYLE COMPOSANT INDUSTRIEL --
--*********************************************************--
        IC is style extending Component where {
            -- Le style IC étend le style composant --
            styles {
                IC_IC_port is style extending Port where {
                    -- Définition du port pour interfacer le connecteur pour un autre IC --
                    constructors {
                        IC_IC_port is constructor(); {
                            -- Définition du constructeur du port --
                            port with {
                                connections {
                                    -- Définition des connexions du port --
                                    input is connection(any),
                                    output is connection(any)
                                }
                            }
                        }
                    }
                },
                IC_PS_port is style extending Port where {
                    -- Définition du port pour interfacer le connecteur pour le PS --
                    constructors {
                        IC_PS_port is constructor(); {
                            -- Définition du constructeur du port --
                            port with {
                                connections {
                                    -- Définition des connexions du port --
                                    obj_IC is connection(any),
                                    result_IC is connection(any)
                                }
                            }
                        }
```

```
            }
        },
        IC_RMS_port is style extending Port where {
            -- Définition du port pour interfacer le connecteur pour le RMS --
            constructors {
                IC_RMS_port is constructor(); {
                    -- Définition du constructeur du port --
                    port with {
                        connections {
                            -- Définition des connexions du port --
                            query_IC is connection(any),
                            response_IC is connection(any)
                        }
                    }
                }
            }
        }
    }
    constructors {
        IC is constructor(); {
            -- Constructeur du style IC --
            component with {
                ports {
                    IC_Mediator_port is IC_Mediator_port,
                    IC_PS_port is IC_PS_port,
                    IC_RMS_port is IC_RMS_port
                }
                computation {
                    -- Définition du comportement du composant IC --
                    choose  {
                        {
                            -- Gestion des flots d'entrée/sortie du composant IC --
                            via input receive input:any;
                            unobservable;
                            via output send output
                        }
                    or
                        {
                            -- Gestion des flots d'information relatif au PS du --
                            -- composant IC --
                            via obj_IC receive objective:any;
                            unobservable;
                            via result_IC send result;
                            recursive
                        }
                    or
                        {
                            -- Gestion des flots d'information relatif au PS du --
                            -- composant RMS --
                            via query_IC send query;
                            unobservable;
                            via response_IC receive response:any;
                            recursive
                        }
                    }
                }
                configuration {
                    new IC_IC_port;
```

```
                        new IC_PS_port;
                        new IC_RMS_port
                    }
                }
            }
        }
        -- Définition de quelques contraintes du style IC --
        constraints {
            -- Il y a au moins un port au niveau du IC pour communiquer avec le composite --
            To ports apply {
                Exists ([1]x | x in archetype IC_IC_port)
            }
            -- Après la réception d'un objectif, il y a une requête vers le RMS --
            To ports apply {
                Exists (p1 : IC_PS_port, p2 : IC_RMS_port |
                    Every sequence {true*.via p1::obj_IC receive any}
                        Leads to state {true*.via p2::query_IC send any}
                )
            }
        }
    },
--*********************************************************--
-- LE STYLE SYSTEME DE PERFORMANCE--
--*********************************************************--
    PS is style extending Component where {
        -- Le style PS étend le style composant --
        styles {
            Performance_port is style extending Port where {
                -- Définition du port pour interfacer le connecteur pour le IC --
                constructors {
                    Performance_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                obj_PS is connection(any),
                                result_PS is connection(any)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            PS is constructor(); {
                -- Constructeur du style PS --
                component with {
                    ports {
                        Performance_port is Performance_port
                    }
                    computation {
                        -- Définition du comportement du composant PS --
                        via obj_PS send objective;
                        unobservable;
                        via result_PS receive result:any;
                        recursive
                    }
                    configuration {
```

```
                        new Performance_port
                    }
                }
            }
        }
        -- Définition de quelques contraintes du style PS --
        constraints {
            -- Il y a au moins un port du PS pour communiquer avec un IC --
            To ports apply {
                Exists (p:1..*)([p]x | x in archetype Performance_port)
            }
        }
    },
--*********************************************************--
-- LS STYLE SYSTEME DE GESTION DE RESSOURCE --
--*********************************************************--
    RMS is style extending Component where {
        -- Le style RMS étend le style composant --
        styles {
            Resource_port is style extending Port where {
                -- Définition du port pour interfacer le connecteur pour le RMS --
                constructors {
                    Resource_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                query_RMS is connection(any),
                                response_RMS is connection(any)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            RMS is constructor(); {
                -- Constructeur du style RMS --
                component with {
                    ports {
                        Resource_port is Resource_port
                    }
                    computation {
                        -- Définition du comportement du composant RMS --
                        via query_RMS receive query:any;
                        unobservable;
                        via response_RMS send response;
                        recursive
                    }
                    configuration {
                        new Resource_port
                    }
                }
            }
        }
        -- Définition de quelques contraintes du style RMS --
        constraints {
            -- Il y a au moins un port du RMS pour communiquer avec un IC --
```

```
            To ports apply {
                Exists (p:1..*)([p]x | x in archetype Resource_port)
            }
        }
    },
--*********************************************************--
-- LE STYLE CONNECTEUR ENTRE UN IC ET UN SP --
--*********************************************************--
    ConPS is style extending Connector where {
        -- Le style ConPS étend le style connecteur --
        styles {
            IC_PS_conPS_port is style extending Port where {
                -- Définition du port pour interfacer le IC --
                constructors {
                    IC_PS_conPS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                obj_IC is connection(any),
                                result_IC is connection(any)
                            }
                        }
                    }
                }
            },
            Performance_conPS_port is style extending Port where {
                -- Définition du port pour interfacer le PS --
                constructors {
                    Performance_conPS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                obj_PS is connection(any),
                                result_PS is connection(any)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            ConPS is constructor(); {
                -- Constructeur du style ConPS --
                connector with {
                    ports {
                        IC_PS_conPS_port is IC_PS_conPS_port,
                        Performance_conPS_port is Performance_conPS_port
                    }
                    routing {
                        -- Définition du comportement du connecteur ConPS --
                        via obj_PS receive objective:any;
                        via obj_IC send objective;
                        via result_IC receive result:any;
                        via result_PS send result
                    }
                    configuration {
```

```
                        new IC_PS_conPS_port;
                        new Performance_conPS_port
                    }
                }
            }
        }
    },
--*********************************************************--
-- LE STYLE CONNECTEUR ENTRE UN IC ET UN SGR --
--*********************************************************--
    ConRMS is style extending Connector where {
        -- Le style ConPS étend le style connecteur --
        styles {
            IC_RMS_conRMS_port is style extending Port where {
                -- Définition du port pour interfacer le IC --
                constructors {
                    IC_RMS_conRMS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                query_IC is connection(any),
                                response_IC is connection(any)
                            }
                        }
                    }
                }
            },
            Resource_conRMS_port is style extending Port where {
                -- Définition du port pour interfacer le RMS --
                constructors {
                    Resource_conRMS_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                query_RMS is connection(any),
                                response_RMS is connection(any)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            ConRMS is constructor(); {
                -- Constructeur du style ConRMS --
                connector with {
                    ports {
                        IC_RMS_conRMS_port is IC_RMS_conRMS_port,
                        Resource_conRMS_port is Resource_conRMS_port
                    }
                    routing {
                        -- Définition du comportement du connecteur ConRMS --
                        via query_IC receive query:any;
                        via query_RMS send query;
                        via response_RMS receive response:any;
                        via response_IC send response
```

```
                    }
                    configuration {
                        new IC_RMS_conRMS_port;
                        new Resource_conRMS_port
                    }
                }
            }
        }
    },
--*********************************************************--
-- LE STYLE CONNECTEUR ENTRE DEUX IC --
--*********************************************************--
    ConIC is style extending Connector where {
        -- Le style ConPS étend le style connecteur --
        styles {
            IC_Input_conIC_port is style extending Port where {
                -- Définition du port pour interfacer un IC en entrée --
                constructors {
                    IC_Input_conIC_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition de la connexion du port --
                                input_IC is connection(any)
                            }
                        }
                    }
                }
            },
            IC_Output_conIC_port is style extending Port where {
                -- Définition du port pour interfacer un IC en sortie --
                constructors {
                    IC_Output_conIC_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition de la connexion du port --
                                output_IC is connection(any)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            ConIC is constructor(); {
                -- Constructeur du style ConIC --
                connector with {
                    ports {
                        IC_Input_conIC_port is IC_Input_conIC_port,
                        IC_Output_conIC_port is IC_Output_conIC_port
                    }
                    routing {
                        -- Définition du comportement du connecteur ConIC --
                        via output_IC receive output:any;
                        via input_IC send input
                    }
                    configuration {
```

```
                    new IC_Input_conIC_port;
                    new IC_Output_conIC_port
                }
            }
        }
    }
},
Enterprise_port is style extending Port where {
    -- Définition du port du composite Enterprise --
    constructors {
        Enterprise_port is constructor(); {
            -- Définition du port pour interfacer le port du composite avec les IC --
            port with {
                connections {
                    -- Définition des connexions du port --
                    input is connection(any),
                    output is connection(any)
                    }
                }
            }
        }
    }
}
constructors {
    Enterprise is constructor( numIC : Natural ); {
        -- Constructeur du style Enterprise --
        component with {
            ports {
                Enterprise_port is Enterprise_port
            }
            constituents {
                -- Définition des types des éléments du style Enterprise --
                ic is IC,
                ps is PS,
                rms is RMS,
                conps is ConPS,
                conrms is ConRMS,
                conic is ConIC
            }
            configuration {
                -- Création de l'élément PS --
                new ps;
                -- Création de l'élément RMS --
                new rms;
                iterate sequence(#1..numIC) by i:Natural do {
                    -- Création des éléments IC --
                    new ic;
                    -- Création des éléments ConPS --
                    new conps;
                    -- Création des éléments ConRMS --
                    new conrms;
                }
                iterate sequence(#1..numIC) by j:Natural do {
                    -- Création des attachements entre les IC et le ConPS --
                    attach ic::j::IC_PS_port::obj_IC to
                              conps::j::IC_PS_conPS_port::obj_IC;
                    attach ic::j::IC_PS_port::result_IC to
```

```
                        conps::j::IC_PS_conPS_port::result_IC;
                -- Création des attachements entre les PS et le ConPS --
                attach ps::Performance_port::j::obj_PS to
                        conps::j::Performance_conPS_port::obj_PS;
                attach ps::Performance_port::j::result_PS to
                        conps::j::Performance_conPS_port::result_PS;
                -- Création des attachements entre les IC et le ConRMS --
                attach ic::j::IC_RMS_port::query_IC to
                        conrms::j::IC_RMS_conRMS_port::query_IC;
                attach ic::j::IC_RMS_port::response_IC to
                        conrms::j::IC_RMS_conRMS_port::response_IC;
                -- Création des attachements entre les RMS et le ConRMS --
                attach rms::Resource_port::j::query_RMS to
                        conrms::j::Resource_conRMS_port::query_RMS;
                attach rms::Resource_port::j::response_RMS to
                        conrms::j::Resource_conRMS_port::response_RMS;
            }
            iterate sequence(#1..(numIC-#1)) by j:Natural do {
                -- Création des éléments ConIC --
                new conic;
                -- Création des attachements entre les IC et le ConIC --
                attach ic::j::IC_IC_port::output to
                        conic::j::IC_Output_conIC_port::output_IC;
                attach ic::j+#1::IC_IC_port::input to
                        conic::j::IC_Input_conIC_port::input_IC;
                -- Création des attachements entre les IC et le composite Enterprise --
                bind ic::j::IC_IC_port::input to Enterprise_port::input;
                bind ic::j+#1::IC_IC_port::output to Enterprise_port::output
            }
        }
    }
}
-- Définition de quelques contraintes du style Entreprise --
constraints {
    -- Il y a un et un seul élément qui suit le style PS --
    To elements apply {
        Exists ([1]x | x in style PS)
    }
    -- Il y a un et un seul élément qui suit le style RMS --
    To elements apply {
        Exists ([1]y | y in style RMS)
    }
    -- Il y a au moins un élément qui suit le style IC --
    To elements apply {
        Exists (p:1..*)([p]z | z in style IC)
    }
    -- Chaque IC est lié à un ConPS --
    To ports apply {
        Forall (x | x in archetype IC::IC_PS_port implies
            Exists (y | y in archetype CONPS::IC_PS_conPS_port
                and x attached to y)
        )
    }
    -- Chaque ConPS est lié au PS --
    To ports apply {
        Forall (x | x in archetype CONPS::Performance_conPS_port implies
            Exists (y | y in archetype PS::Performance_port
```

```
                and x attached to y)
            )
        }
        -- Chaque IC est lié à un ConRMS --
        To ports apply {
            Forall (x | x in archetype IC::IC_RMS_port implies
                Exists (y | y in archetype CONRMS::IC_RMS_conRMS_port
                    and x attached to y)
            )
        }
        -- Chaque ConRMS est lié au RMS --
        To ports apply {
            Forall (x | x in archetype CONRMS::Resource_conRMS_port implies
                Exists (y | y in archetype RMS::Resource_port
                    and x attached to y)
            )
        }
        -- Deux IC ne peuvent pas être liés directement --
        To elements apply {
            Forall (x | x in style IC implies
                Not Exists (y | y in style IC and x attached to y)
            )
        }
    }
    -- Description des analyses relatives au style Entreprise --
    analyses {
        ..............
    }
}
```

## 2. Le style inter-entreprise

Ce style inter-entreprise définit d'une part, une architecture où ne figure qu'un Donneur d'Ordres, un médiateur et deux sous-traitants et, d'autre part, le protocole du médiateur ne correspond qu'au protocole de négociation d'un appel d'offre entre le Donneur d'Ordres et les deux sous-traitants.

```
--*********************************************************--
-- LE STYLE INTER-ENTREPRISE --
--*********************************************************--
Inter_Enterprises is style extending Component where {
    -- Le style Inter_Enterprises étend le style composant --
    styles {
--*********************************************************--
-- LE STYLE DONNEUR D'ORDRES --
--*********************************************************--
        Contractor is style extending Enterprise where {
            -- Le style Contractor étend le style Enterprise --
            styles {
                Contractor_port is style extending Port where {
                    -- Définition du port pour interfacer le médiateur --
                    constructors {
                        Contractor_port is constructor(); {
                            -- Définition du constructeur du port --
                            port with {
                                connections {
                                    -- Définition des connexions du port --
                                    contractor_tender is connection(String),
                                    contractor_answer is connection(String),
                                    contractor_contract is connection(String)
                                }
                            }
                        }
                    }
                }
            }
            constructors {
                Contractor is constructor(contractor_name:String); {
                    -- Constructeur du style Contractor --
                    component with {
                        ports {
                            Contractor_port is Contractor_port
                        }
                        computation {
                            -- Définition du comportement du composant Contractor --
                            -- Envoi de l'appel d'offre --
                            via contractor_tender send tender;
                            -- Réception des réponses --
                            via contractor_answer receive answer:String;

                            if(answer <> "")
                            do
                            {
                                -- Initialisation --
                                value contract is "Subcontracting agreement";
```

```
                        -- Envoi du contrat --
                        via contractor_contract send contract
                    }
                }
                configuration {
                    new Contractor_port
                }
            }
        }
    },
--*********************************************************--
-- LE STYLE SOUS-TRAITANT --
--*********************************************************--
    Supplier is style extending Enterprise where {
        -- Le style Supplier étend le style composant --
        styles {
            Supplier_port is style extending Port where {
                -- Définition du port pour interfacer le médiateur --
                constructors {
                    Supplier_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                supplier_tender is connection(String),
                                supplier_answer is connection(String,Boolean),
                                supplier_contract is connection(String)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            Supplier is constructor(supplier_name:String); {
                -- Constructeur du style Supplier --
                component with {
                    ports {
                        Supplier_port is Supplier_port
                    }
                    computation {
                        -- Définition du comportement du composant Supplier --
                        -- Réception de l'appel d'offre --
                        via supplier_tender receive tender:String;
                        -- Choix au niveau de la réponse --
                        choose
                        {
                            {
                                -- Envoi d'une réponse positive --
                                via supplier_answer send supplier_name,true;
                                -- Réception contrat --
                                via supplier_contract receive contract:String
                            }
                        or
                            {
                                -- Envoi d'une réponse négative --
                                via supplier_answer send false
```

```
                        }
                    }
                }
                configuration {
                    new Supplier_port
                }
            }
        }
    },
--*********************************************************--
-- LE STYLE MEDIATEUR --
--*********************************************************--
    Mediator is style extending Connector where {
        -- Le style Mediator étend le style connecteur --
        styles {
            Mediator_Contractor_port is style extending Port where {
                -- Définition du port pour interfacer le contractor --
                constructors {
                    Mediator_Contractor_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                contractor_tender is connection(String),
                                contractor_answer is connection(String),
                                contractor_contract is connection(String)
                            }
                        }
                    }
                }
            },
            Mediator_Supplier_port is style extending Port where {
                -- Définition du port pour interfacer le Supplier --
                constructors {
                    Mediator_Supplier_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition des connexions du port --
                                supplier_tender is connection(String),
                                supplier_answer is connection(String,Boolean),
                                supplier_contract is connection(String)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            Mediator is constructor(); {
                -- Constructeur du style Mediator --
                connector with {
                    ports {
                        Mediator_Contractor_port is Mediator_Contractor_port,
                        Mediator_Supplier_port is Mediator_Supplier_port
                    }
                    routing {
```

```
-- Définition du comportement du connecteur ConPS --
-- Réception de l'appel d'offre --
via contractor_tender receive tender:String;
-- Envoi de l'appel d'offre aux sous-traitants --
via supplier_tender send tender;
via supplier_tender send tender;
-- Réception des réponses des sous-traitants --
via supplier_answer receive supplier_name1:String,answer1:Boolean;
via supplier_answer receive supplier_name2:String,answer2:Boolean;
-- Test si la réponse du sous-traitant 1 est positive --
if(answer1)
then
{
    -- Test si la réponse du sous-traitant 2 est positive --
    if(answer2)
    then
    {
        choose
        {
            {
                    -- Envoi du contrat au sous-traitant 1 --
                    via contractor_answer send supplier_name1;
                    via contractor_contract receive contract:String;
                    via supplier_contract send contract;
                    via supplier_contract send
                           "Sorry, you are not selected"
            }
        or
            {
                    -- Envoi du contrat au sous-traitant 2 --
                    via contractor_answer send supplier_name2;
                    via contractor_contract receive contract:String;
                    via supplier_contract send
                           "Sorry, you are not selected";
                    via supplier_contract send contract
            }
        or
            {
                    -- Envoi d'aucun contrat --
                    via contractor_answer send "";
                    via supplier_contract send
                           "Sorry, you are not selected";
                    via supplier_contract send
                           "Sorry, you are not selected"
            }
        }
    }
    else
    {
        choose
        {
            {
                    -- Envoi du contrat au sous-traitant 1 --
                    via contractor_answer send supplier_name1;
                    via contractor_contract receive contract:String;
                    via supplier_contract send contract
            }
        or
```

```
                                {
                                        -- Envoi d'aucun contrat --
                                        via contractor_answer send "";
                                        via supplier_contract send
                                               "Sorry, you are not selected"
                                }
                            }
                        }
                    }
                    else
                    {
                        if(answer2)
                        then
                        {
                            choose
                            {
                                {
                                        -- Envoi du contrat au sous-traitant 2 --
                                        via contractor_answer send supplier_name2;
                                        via contractor_contract receive contract:String;
                                        via supplier_contract send contract
                                }
                                or
                                {
                                        -- Envoi d'aucun contrat --
                                        via contractor_answer send "";
                                        via supplier_contract send
                                               "Sorry, you are not selected"
                                }
                            }
                        }
                        else
                        {
                            {
                                -- Envoi d'aucun contrat --
                                via contractor_answer send ""
                            }
                        }
                    }
                }
                configuration {
                    new Mediator_Contractor_port;
                    new Mediator_Supplier_port
                }
            }
        }
    }
}
constructors {
    Inter_Enterprises is constructor(); {
        -- Constructeur du style Inter_Enterprises --
        component with {
            constituents {
                -- Définition des types des éléments du style Inter_Enterprises --
                contractor is Contractor,
                mediator is Mediator,
                supplier1 is Supplier,
```

```
                supplier2 is Supplier
            }
            configuration {
                -- Création de l'élément Contractor --
                new contractor;
                -- Création de l'élément Médiator --
                new mediator;
                -- Création des deux éléments Supplier --
                new supplier1;
                new supplier2;
                -- Création des attachements entre le Contractor et le Mediator --
                attach contractor::Contractor_port::contractor_tender to
                          mediator::Mediator_Contractor_port::contractor_tender;
                attach contractor::Contractor_port::contractor_answer to
                          mediator::Mediator_Contractor_port::contractor_answer;
                attach contractor::Contractor_port::contractor_contract to
                          mediator::Mediator_Contractor_port::contractor_contract;
                -- Création des attachements entre les Suppliers et le Mediator --
                attach supplier1::Supplier_port::supplier_tender to
                          mediator::Mediator_Supplier_port::supplier_tender;
                attach supplier1::Supplier_port::supplier_answer to
                          mediator::Mediator_Supplier_port::supplier_answer;
                attach supplier1::Supplier_port::supplier_contract to
                          mediator::Mediator_Supplier_port::supplier_contract;
                attach supplier2::Supplier_port::supplier_tender to
                          mediator::Mediator_Supplier_port::supplier_tender;
                attach supplier2::Supplier_port::supplier_answer to
                          mediator::Mediator_Supplier_port::supplier_answer;
                attach supplier2::Supplier_port::supplier_contract to
                          mediator::Mediator_Supplier_port::supplier_contract
            }
        }
    }
}
-- Définition de quelques contraintes du style Inter_Enterprises --
constraints {
    ..............
}
-- Description des analyses relatives au style Inter_Enterprises --
analyses {
    ..............
}
}
```

# 3. Le style EAI

Ce style EAI définit une architecture où ne figure un moteur de workflow, des COTS et les différents connecteurs entre le moteur de workflow et ces COTS.

```
--*********************************************************--
-- LE STYLE EAI --
--*********************************************************--
EAI is style extending Component where {
    -- Le style EAI étend le style composant --
    styles {
--*********************************************************--
-- LE STYLE MOTEUR DE WORKFLOW --
--*********************************************************--
        Engine is style extending Component where {
            -- Le style Engine étend le style composant --
            styles {
                Engine_COTS_port is style extending Port where {
                    -- Définition du port pour interfacer le connecteur pour un COTS --
                    constructors {
                        Engine_COTS_port is constructor(); {
                            -- Définition du constructeur du port --
                            port with {
                                connections {
                                    -- Définition de la connexion du port --
                                    message_Cots is connection(any)
                                }
                            }
                        }
                    }
                },
                Engine_Negociator_port is style extending Port where {
                    -- Définition du port pour interfacer le composite --
                    constructors {
                        Engine_Negociator_port is constructor(); {
                            -- Définition du constructeur du port --
                            port with {
                                connections {
                                    -- Définition des connexions du port --
                                    message_In is connection(any)
                                    message_Out is connection(any)
                                }
                            }
                        }
                    }
                }
            }
            constructors {
                Engine is constructor(); {
                    -- Constructeur du style Engine --
                    component with {
                        ports {
                            Engine_COTS_port is Engine_COTS_port,
                            Engine_Negociator_port is Engine_Negociator_port
                        }
```

```
                    computation {
                        -- Définition du comportement du composant Engine --
                        choose {
                            {
                            -- Gestion du flot de sortie du composite --
                            via message_Out send msg;
                            unobservable;
                            recursive
                            }
                        or
                            {
                            -- Gestion du flot d'entrée du composite --
                            via message_In receive msg:any;
                            unobservable;
                            recursive
                            }
                        or
                            {
                            -- Gestion du flot d'entrée d'un COTS --
                            via message_Cots send msg;
                            unobservable;
                            recursive
                            }
                        or
                            {
                            -- Gestion du flot de sortie d'un COTS --
                            via message_Cots receive msg:any;
                            unobservable;
                            recursive
                            }
                        }
                    }
                    configuration {
                        new Engine_COTS_port;
                        new Engine_Negociator_port
                    }
                }
            }
        }
    },
--*********************************************************--
-- LE STYLE COTS --
--*********************************************************--
    Cots is style extending Component where {
        -- Le style Cots étend le style composant --
        styles {
            Cots_port is style extending Port where {
                -- Définition du port pour interfacer le connecteur pour le moteur de WF --
                constructors {
                    Cots_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition de la connexion du port --
                                message is connection(any)
                            }
                        }
                    }
```

```
                }
            }
        }
        constructors {
            Cots is constructor(); {
                -- Constructeur du style Cots --
                component with {
                    ports {
                        Cots_port is Cots1_port
                    }
                    computation {
                        -- Définition du comportement du composant Cots --
                        choose {
                            {
                            -- Gestion de l'émission d'un message --
                            via message send msg;
                            unobservable;
                            recursive
                            }
                        or
                            {
                            -- Gestion de l'arrivée d'un message --
                            via message receive msg:any;
                            unobservable;
                            recursive
                            }
                        }
                    }
                    configuration {
                        new Cots_port
                    }
                }
            }
        }
    },
--*********************************************************--
-- LE STYLE CONNECTEUR ENTRE LE MOTEUR DE WF ET LE COTS --
--*********************************************************--
    ConCOTS is style extending Connector where {
        -- Le style ConCOTS étend le style connecteur --
        styles {
            ConCOTS_Engine_port is style extending Port where {
                -- Définition du port pour interfacer le moteur de WF --
                constructors {
                    ConCOTS_Engine_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition de la connexion du port --
                                message_engine is connection(any)
                            }
                        }
                    }
                }
            },
            ConCOTS_Cots_port is style extending Port where {
                -- Définition du port pour interfacer le COTS --
                constructors {
```

```
                    ConCOTS_Cots_port is constructor(); {
                        -- Définition du constructeur du port --
                        port with {
                            connections {
                                -- Définition de la connexion du port --
                                message_cots is connection(any)
                            }
                        }
                    }
                }
            }
        }
        constructors {
            ConCOTS is constructor(); {
                -- Constructeur du style ConCOTS --
                connector with {
                    ports {
                        ConCOTS_Engine_port is ConCOTS_Engine_port,
                        ConCOTS_Cots_port is ConCOTS_Cots_port
                    }
                    routing {
                        -- Définition du comportement du connecteur ConCOTS --
                        choose
                        {
                            {
                                via message_engine receive msg:any;
                                via message_cots send msg
                            }
                        or
                            {
                                via message_cots receive msg:any;
                                via message_engine send msg
                            }
                        }
                    }
                    configuration {
                        new ConCOTS_Engine_port;
                        new ConCOTS_Cots_port
                    }
                }
            }
        }
    },
    Inter_Enterprises_port is style extending Port where {
        -- Définition du port du composite Inter_Enterprises --
        constructors {
            Inter_Enterprises_port is constructor(); {
                -- Définition du port pour interfacer le port du composite avec le Moteur de
                    WF --
                port with {
                    connections {
                        -- Définition des connexions du port --
                        input is connection(any),
                        output is connection(any)
                        }
                    }
                }
            }
```

```
            }
        }
    }
    constructors {
        EAI is constructor ( numCOTS : Natural ); {
            component with {
                constituents {
                    cots is COTS,
                    engine is Engine,
                    conCots is ConCots
                }
                configuration {
                    iterate sequence(#1..numCOTS) by i:Natural do {
                        -- Création des éléments COTS --
                        new cots;
                        -- Création des éléments ConCOTS --
                        new conCots;
                    }
                    new engine;
                    iterate sequence(#1..numCOTS) by j:Natural do {
                        -- Création des attachements entre les COTS et les ConCOTS --
                        attach cots::j::Cots_port::message to
                                  conCots::j::ConCOTS_Cots_port::message_cots;
                        -- Création des attachements entre les ConCOTS et le moteur de WF --
                        attach conCots::j::ConCOTS_Engine_port::message_engine to
                                  engine::Engine_COTS_port::j::message_Cots;
                    }
                    -- Création des attachements entre le moteur de WF et le composite
                       Inter_Enterprises --
                    bind engine::Engine_Negociator_port::message_In to
                              Inter_Enterprises_port::input;
                    bind engine::Engine_Negociator_port::message_Out to
                              Inter_Enterprises_port::output;
                }
            }
        }
    }
    -- Définition de quelques contraintes du style EAI --
    constraints {
        ..............
    }
    -- Description des analyses relatives au style EAI --
    analyses {
        ..............
    }
}
```

# *Modélisation des processus « métier » mis en œuvre dans une approche EAI en vue de leur pilotage « Le pilotage des applications intégrées »*

***par Blanc Dit Jolicoeur Lionel***

## *Résumé :*

Les entreprises doivent tenir compte d'informations provenant d'applications hétérogènes. Des solutions technologiques permettent d'intégrer ces applications mais ces solutions souffrent d'une absence de cadre de modélisation pour fournir une vision globale de l'entreprise et de son système d'information (SI) qui doit masquer l'hétérogénéité des applications en proposant une vision unifiée des données. Ceci nécessite un cadre de modélisation capable de fournir une vision globale de l'entreprise.

Pour être agiles, les entreprises s'appuient sur une approche processus. Le SI doit donc s'appuyer sur un cadre de modélisation formalisant les processus et proposant un modèle de pilotage. De plus, ce cadre de modélisation doit garantir la cohérence entre les perceptions des acteurs de l'entreprise. Le paradigme d'ambivalence propose d'étudier la réalité opérationnelle de l'entreprise à travers les points de vue « métier » et « donnée ».

De plus, quand le nombre de composants d'un système est important et que ses étapes de développement font intervenir plusieurs métiers, nous formalisons ce système par son architecture. Aussi, la représentation de la réalité opérationnelle de l'entreprise est obtenue par la formalisation d'architectures industrielles et logicielles. Le SI doit donc s'appuyer sur un cadre de modélisation capable de formaliser ces architectures.

Enfin, le paradigme d'ambivalence précise qu'il faut définir des correspondances entre les concepts des deux points de vue. Nous considérons que ces correspondances sont facilitées par l'utilisation d'un langage formel centré architecture. Le SI doit donc intégrer ces correspondances pour garantir la cohérence des perceptions des divers acteurs.

Le travail réalisé dans cette thèse intègre les concepts du modèle OLYMPIOS, qui fournit un cadre de modélisation pour formaliser des processus, dans un ensemble de langages centrés architecture, issus du projet ArchWare dédié à l'ingénierie des systèmes logiciels évolutifs.

## *Mots-clés :*

Système d'information, Modélisation des processus, Pilotage, Modèle OLYMPIOS, Styles architecturaux, Evolution