MASTER EN SCIENCES INFORMATIQUES

Élaboration d'une méthode pour l'évaluation et la sélection d'ESB

Jurkiewiecz, Yves

Award date:
2020

Awarding institution:
Universite de Namur

UNIVERSITÉ
DE NAMUR

FACULTÉ
D'INFORMATIQUE

Élaboration d'une méthode pour l'évaluation et la sélection d'ESB

Yves Jurkiewiecz

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Résumé

De nos jours, les systèmes d'information des entreprises sont souvent composés de dizaines de systèmes différents pour les plus petites jusqu'à des milliers pour les plus grandes. De nombreux flux de communication doivent être établis entre ces systèmes afin de construire un écosystème applicatif efficace et souple, permettant à ces entreprises d'atteindre l'agilité nécessaire pour répondre au changement continu des besoins. Ce travail propose dans une première section d'étudier les différentes méthodes d'intégration d'applications et dans une seconde partie de se focaliser sur les Enterprise Service Bus (ESB). Nous étudierons en détail leur architecture, proposerons un modèle de leurs capacités fonctionnelles sous forme d'un diagramme de fonctionnalités ainsi qu'une méthode d'évaluation et de sélection tirant parti de ce modèle. Nous terminerons par une mise en application de l'ESB WSO2 sur 2 cas pratiques.

Mots clés
Enteprise Service Bus, ESB, architecture, modèle de capacités, sélection, évaluation, WSO2 ESB

Abstract

Nowadays, corporate information systems are often composed of dozens of different systems for the smallest to thousands for the largest. Many communication flows must be established between these systems in order to build an efficient and flexible application ecosystem, allowing these companies to achieve the agility necessary to meet the ever-changing needs. This work proposes in a first section to study the different methods of application integration and in a second part to focus on Enterprise Service Bus (ESB). We will study in detail their architecture, propose a model of their functional capabilities in the form of a functional diagram as well as an evaluation and selection method based on this model. We will end with the implementation of WSO2 ESB on 2 practical cases.

Keywords
Enteprise Service Bus, ESB, architecture, capacities model, selection, evaluation, WSO2 ESB

Remerciements

Ce travail conclut mon parcours universitaire entamé en septembre 2017.

Je tiens à remercier mes parents pour le soutien qu'ils m'ont apporté durant ces deux années, ma fille Emma qui a été ma source de motivation, ainsi que Jessica avec qui j'ai partagé cette aventure universitaire dans la joie comme dans la difficulté.

Merci aussi à mes camarades de classe pour les moments passés ensemble, leur bonne humeur et notre entraide mutuelle.

Je remercie également le professeur Vincent Englebert pour sa disponibilité et le soutien qu'il m'a apporté pour me permettre de mener ce travail à son terme.

Pour terminer, je remercie le corps professoral de la faculté d'informatique qui participe grandement à sa dimension humaine.

Table des matières

Introduction

De nos jours, les entreprises de tous domaines et de toutes tailles exigent plus de valeur de leurs processus d'affaires. Un aspect important qui contribue à améliorer la valeur de ces processus d'affaires est la capacité de créer et de modifier ces processus de manière flexible et rapide. De ce fait, il est important que les différentes applications logicielles qui supportent ces processus puissent communiquer entre elles. L'interopérabilité est cependant souvent absente des processus de conception de ces applications. L'établissement a posteriori d'une communication entre ces applications est souvent un processus lent et coûteux en raison de différents paramètres comme l'utilisation de formats de données et de protocoles de communication différents. Depuis le début des années 2000, d'importantes tendances technologiques ont été observées pour résoudre ce problème : l' architecture orientée services (SOA) ; l'intégration des applications d'entreprise (EAI) ; le Business-to-Business (B2B) ; les web services. Ces technologies tentent de relever les défis de l'amélioration des processus opérationnels en misant sur la flexibilité et l'agilité [13]. L'architecture SOA constitue le standard actuel en la matière. Tous les grands éditeurs de logiciels, dont IBM, Microsoft, Oracle et SAP, ont adopté l'architecture SOA et investi d'importantes sommes d'argent pour revoir la conception de leurs produits selon les principes des architectures SOA. Une SOA n'est liée à aucune technologie spécifique dans sa mise en œuvre, elle pourra donc intervenir à l'aide de web services, de serveurs d'applications, de bus d'entreprise ou encore d'autre middelwares. L'ESB tire les meilleures caractéristiques de ces tendances et d'autres tendances technologiques[1]. Il est une architecture d'intégration facilitatrice des SOA et qui peut être utilisée pour connecter et coordonner l'interaction d'un nombre significatif d'applications diverses.

Motivation

Il existe de nombreux ESB, chacun étant plus ou moins proche du modèle architecturale du même nom. Choisir le bon ESB pour un contexte donné est une décision complexe. La littérature scientifique comporte de nombreuse comparaisons entre les ESB mais celles-ci ne portent que sur quelques produits et/ou sur quelques aspects spécifiques de ceux-ci. Au moment d'écrire ces lignes nous n'avons pas pu trouver de références proposant une méthodologie complète d'évaluation et de sélection d'un ESB. Ce travail a précisément pour objectif de proposer une méthodologie de ce type. Pour y arriver, nous tenterons de répondre à 2 questions :

— QR1 : Quel pourrait être un modèle des capacités fonctionnelles des ESB ?

— QR2 : Quelle méthodologie pourrait être suivie pour évaluer et sélectionner un ESB sur base de ce modèle ?

Afin de répondre à ces questions, nous commencerons par consacrer un premier chapitre à l'intégration d'applications. Nous y aborderons de manière générale la notion d'intégration au sein de l'entreprise. Ensuite nous développerons en particulier l'intégration d'applications en étudiant les styles, méthodes et architectures qui s'y rapportent. Le second chapitre sera consacré à l'architecture ESB. Nous verrons qu'il n'existe pas de définition unanimement reconnue de ce qu'est un ESB et nous positionnerons par rapport à cette problématique. L'architecture ESB sera ensuite décrite en détail et nous terminerons le chapitre en fournissant des pistes de réflexion permettant de répondre à la question « Quand utiliser un ESB ? ». Nous entrerons ensuite dans le vif du sujet avec le chapitre 3 qui nous permettra de répondre à notre 1ère question de recherche. Il sera consacré à l'élaboration d'un modèle des capacités des ESB sous la forme d'un diagramme de fonctionnalités dont les catégories principales de fonctionnalités seront : la connectivité ; les interactions de services ; le traitement des messages ; la sécurité ; la gestion ; la qualité de services. Le chapitre 4 apportera une réponse à la seconde question de recherche. Nous y proposerons une méthodologie d'évaluation et de sélection des ESB adaptable à tous contextes. La fin de ce chapitre sera consacrée à la mise en application de cette méthode. Nous terminerons ce travail par un chapitre consacré à la mise en œuvre de 2 cas d'utilisation des ESB qui nous permettront de nous familiariser avec quelques concepts clés et d'illustrer quelques-uns des principes théoriques évoqués jusque là.

Cas d'accompagnement

Afin d'illustrer les concepts abordés tout au long de ce travail, nous accompagnerons celui-ci d'un cas imaginaire mais néanmoins réel, celui du système d'information d'un hôpital. L'histoire de l'informatique médicale débute entre 1955 et 1965 avec les premières expérimentations dans le champ de la logique symbolique [1], de la biostatistique et des techniques automatiques en matière de décisions médicales [44]. Elles ont aujourd'hui laissé place à des systèmes d'information hospitaliers (ou HIS pour Hospital Information System) complexes, gérant de nombreux domaines métiers sous forme d'un agrégat intégré de sous systèmes d'information et de composants internes ou externes spécialisés. La figure 1 montre un exemple non exhaustif d'un HIS tel qu'ils peuvent exister aujourd'hui.

Dans la suite de ce document, nous ferons référence à ce HIS pour illustrer les concepts abordés. La liste descriptive qui suit aidera le lecteur à identifier et comprendre les fonctions et rôles de chacun des systèmes représentés.

Systèmes internes L'hôpital choisit, organise et gère ces systèmes.

AMS - Accounting Management System Gestion de la comptabilité.

BMS - Bed Management System Système de gestion des lits dont le rôle est d'administrer les flux de patients et l'occupation des lits.

1. La logique symbolique consiste en l'utilisation de symboles pour indiquer des propositions, des termes et des relations afin de faciliter le raisonnement.

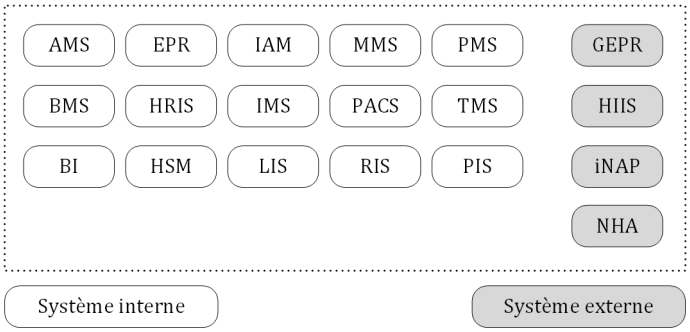


Figure 1 – Un HIS est constitué de nombreux sous-systèmes internes et externes.

BI - Business Intelligence Système permettant, à partir de la collecte des données de l'hôpital, une meilleure compréhension de ses activités et le pilotage de sa planification.

EPR - Electronic Patient Record Système prenant en charge la gestion des données médicales du patient. On parlera aussi de dossier médical informatisé.

HRIS - Human Ressource Information System Système de gestion des ressources humaines, e.g. prise en charge du recrutement, des remplacements, des contrats de travail, ou encore le paiement des employés.

HSM - Hotel Services Management Prise en charge de la logistique e.g. la gestion du matériel, la gestion du personnel d'entretien ou encore la gestion des achats.

IAM - Identity and Access Management Système de gestion de l'accès aux ressources technologiques de l'hôpital.

IMS - Invoice Management System Gestion de la facturation aux bénéficiaires de soins.

LIS - Laboratory Information System Système de gestion des opérations d'un laboratoire clinique.

MMS - Meal Management System Système de gestion des cuisines et repas.

PACS - Picture Archiving Communication System Système d'imagerie numérique permettant l'acquisition, l'archivage, la communication, la récupération, le traitement, la distribution et l'affichage d'images médicales.

PIS - Pharmacy Information System Système de gestion des pharmacies hospitalières e.g. maintien et organisation de l'approvisionnement en médicaments, préparation et distribution des médicaments.

PMS - Patient Management System Système de gestion administrative des patients. Il régit leurs admissions, leurs transferts ou mouvements et enfin leurs sorties. L'acronyme ADT pour « Admission Discharge Transfer » est également souvent utilisé.

TMS - Time Management System Système de gestion du temps avec entre autres la planification des horaires, la saisie des temps de prestations, la gestion des absences, la gestion des heures supplémentaires.

RIS - Radiology Information System Système de gestion administrative d'un département d'imagerie médicale.

Systèmes externes Ils sont gérés par des organismes externes à l'institution hospitalière. Cette dernière est néanmoins souvent légalement contrainte de les intégrer.

GEPR - Global Electronic Patient Record Dossier médical informatisé global. Il permet de centraliser les données médicales de patients moyennant leur consentement. Il est fourni et géré par des agences gouvernementales et permet l'échange d'informations entre praticiens à une échelle régionale ou nationale.

HIIS - Healthcate Insurance Information System Système de gestion des organismes assureurs.

iNAP - interNational Pharmacist Association Services (e.g. liste de médicaments ou contrôle de falsification de médicaments) fournis par des associations nationales ou internationales de pharmaciens.

NHA - National Healthcare Agency Ce système désigne les agences nationales de santé publique. Elles fournissent par exemple des services d'enregistrement de prescriptions électroniques.

Un HIS forme un tout intégré au sein duquel de nombreux flux d'information existent entre les systèmes. Une analyse de ces flux sera présentée dans la section 4.7.2.

La figure 2 montre un exemple de flux d'information au sein d'un HIS.

1. Un patient est admis et enregistré dans le système de gestion des patients (PMS). Son assurabilité est vérifiée auprès des services externes des organismes assureurs (HIIS).
2. L'admission du patient est envoyée vers son dossier médical (EPR) pour l'ouverture d'un épisode d'hospitalisation, dans le système de gestion des lits (BMS) et dans le système de gestion des repas (MMS).
3. Les mouvements du patient (transferts, congés et sorties) sont enregistrés dans le système de gestion des lits (BMS) et envoyés vers le dossier du patient, le système de gestion des patients et le système de gestion des repas.
4. Lors de la sortie du patient, les données médicales pertinentes le concernant sont envoyées vers son dossier médical global (GEPR) tandis que les informations de facturation sont envoyées au système correspondant (IMS).

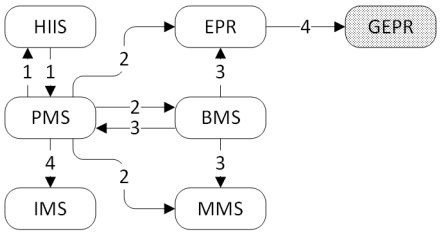


Figure 2 – Exemple de flux d'information dans un HIS

Il est à noter que parmi tous les systèmes que nous venons de décrire, beaucoup sont composés de sous-systèmes au sein desquels des flux d'information sont

également présents. Nous pourrions donc affiner notre HIS pour en obtenir une granularité plus fine. Un MMS pourrait, par exemple, être composé des sous systèmes de gestion des stocks, de gestion de la chaîne de production et de gestion de la diététique des repas. La granularité présentée est cependant suffisante dans le cadre de ce travail. Nous nous en contenterons par souci de simplicité.

Chapitre 1

L'intégration d'applications

Ce 1er chapitre introduit la notion d'intégration des systèmes d'information dans les organisations. Nous débuterons par une présentation d'un cadre général de l'intégration développant les niveaux auxquels l'intégration intervient. Nous nous focaliserons ensuite sur l'intégration d'applications en décrivant le défi qu'elle représente et nous finirons par décrire et illustrer à travers notre cas d'accompagnement les différents styles et architectures applicables à l'intégration d'applications. Les qualités et inconvénients de chacun d'eux seront énumérés.

1.1 Cadre et niveaux de l'intégration

D'un point du vue général, l'intégration est définie par Malone et Crowston comme la gestion coordonnée des dépendances entre activités [42]. Elle intervient à tous les niveaux de l'organisation et concerne tous ses acteurs, qu'ils soient humains ou non humains. Dans [66], Stohr et Nickerson proposent un cadre général de l'intégration représenté sur la figure 1.1. À un degré élevé de granularité, ils affirment que l'intégration intervient à deux grands niveaux. Le premier est le niveau organisationnel. Il fait référence à tous les moyens mis à la disposition des acteurs humains pour permettre et assurer la gestion de l'organisation. On parlera davantage de coordination que d'intégration. Le second niveau qui nous intéressera le plus dans le contexte de ce travail, est celui de l'intégration des systèmes. Elle supporte la coordination et nécessite une organisation planifiée des données, la mise en place de systèmes logiciels de collecte et d'exploitation de données, l'établissement de communications coordonnées entre ces systèmes logiciels et entre ces systèmes logiciels et les acteurs humains. Ainsi, l'intégration des systèmes est divisées en trois couches.

La première, la couche des données, fournit les mécanismes nécessaires à l'organisation au stockage et à la restitution des données qui seront consommées par la couche des applications. Elle a pour objectif de rendre disponible les données en temps opportun et avec exactitude ce qui implique une prise en compte des aspects syntaxique et sémantique. La syntaxe des données permettra leurs traitements par les systèmes applicatifs tandis que la sémantique fournira une compréhension commune et unique de celles-ci. Les mécanismes d'intégration de données pourront être de divers types comme des bases de données de toutes natures, des fichiers ou encore les classes des langages de programmation orientés objet.

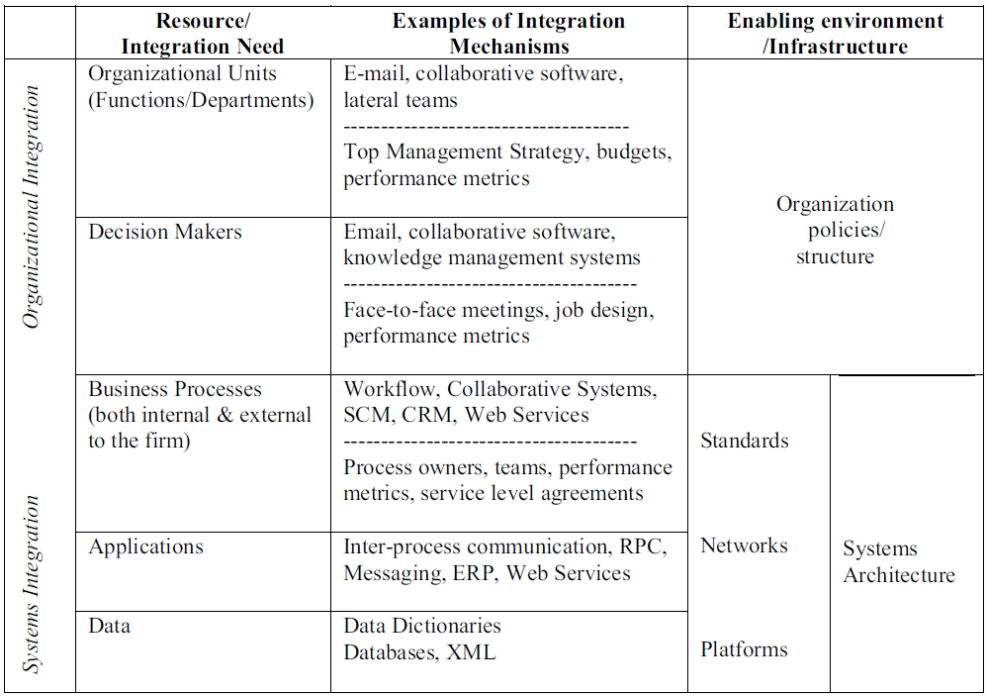


FIGURE 1.1 – Niveaux de l'intégration [66]

La couche d'intégration des applications a pour objectif de permettre la communication entre les systèmes informatiques de l'organisation. Cette communication est nécessaire pour faciliter le traitement et la circulation de l'information au sein de et entre les organisations. La section 1.2 développera davantage l'intégration d'applications.

La troisième et dernière couche de l'intégration des systèmes concerne les processus métiers. Un processus métier est constitué d'un ensemble d'activités connexes qui sont exécutées par des acteurs humains et logiciels selon des règles métiers qui peuvent être appliquées de manière plus ou moins stricte. Les processus occupent souvent un périmètre transversal et peuvent donc constituer un mécanisme de coordination intra et/ou inter-organisation. Les systèmes de gestion de worflow sont spécialement conçus pour supporter l'intégration à ce niveau en coordonnant le travail entre les acteurs humains et logiciels selon les règles définies par l'organisation. Ils fournissent des interfaces pour la conception des processus et la gestion des instances de ces processus. Les ERP dont nous discuterons dans la section 1.3.5 permettent aussi dans une certaine mesure de définir et gérer les processus métiers.

1.2 L'intégration d'applications

Aux prémices de l'informatique, dans les années 50, une organisation ne disposait que d'un mainframe qui gérait toutes ses données et son intelligence. Démocratisé dans les années 60-70, il ne contenait qu'une seule application monolithique traitant les données et gérant leur stockage. Les années 80 voient l'avènement des

ordinateurs personnels et marquent le déclin des mainframes. Ceux-ci disparaissent progressivement pour laisser place aux architectures de type client lourd-serveur. Les entreprises ayant des métiers différents et donc des besoins divers et variés, les serveurs et les applications clientes se sont multipliés, entraînant une complexification des systèmes d'information, une apparition de redondance fonctionnelle et de silos applicatifs. Ce constat va s'aggraver en premier lieu dans les années 90 qui voient l'apparition des applications web et ensuite dans les années 2000 durant lesquelles le concept de cloud computing fait son apparition.

Aujourd'hui, les entreprises sont souvent constituées de centaines voire pour les plus grosses d'entre elles de milliers d'applications. Elles assurent l'ensemble des processus métiers au sein d'une organisation mais recouvrent souvent des domaines fonctionnels communs.

> **Exemple** Dans un hôpital, un patient et sa signalétique sont en premier lieu enregistrés dans le PMS. Cependant, ce patient et sa signalétique sont également enregistrés dans l'EPR. Ceci pose quelques questions comme :
>
> — Quel est le système maître de ces données ?
>
> — Peuvent-elles être modifiées par les deux systèmes ?
>
> — Comment, avec quel délai et sous quelle forme les transférer d'un système à l'autre ?

Tous ces processus et toutes ces données devraient être pris en charge par une seule application. Ce type de solution existe et se nomme ERP. Nous les développerons dans la section 1.3.5. SAP, Oracle ERP ou encore Microsoft Dynamics en sont des exemples. En réalité, il est toutefois impossible de créer une telle application et ces ERP ne prennent en charge qu'une fraction des fonctionnalités métiers de l'entreprise [79, 46, 70]. On assiste donc à un morcellement du parc logiciel des entreprises ce qui d'une part permet à une entreprise de choisir l'application qu'elle estime être la meilleure pour prendre en charge un aspect particulier de son métier, mais d'autre part implique également qu'elles devront être capables de communiquer et d'échanger des données entre elles [46]. Or, ces applications reposent sur de nombreuses technologies, protocoles et plateformes différentes. L'intégration de ces applications dans un ensemble unifié de processus métiers émerge comme une priorité. Forester Research estime que 35% du temps de développement et 30% des coûts de développement sont consacrés à la création et à la maintenance d'interfaces pour les applications et les sources de données [65]. Il est donc primordial d'investir dans des technologies et des solutions d'intégration efficientes.

Mais qu'est ce que l'intégration d'applications ? Plusieurs définitions sont présentes dans la littérature.

Selon Woolf et Hohpe 2004 [79]

> L'intégration d'entreprise est la tâche qui consiste à faire travailler de concert des applications disparates pour produire un ensemble unifié de fonctionnalité.

Selon Gulledge 2006 [27]

> L'intégration implique que toutes les données pertinentes pour un ensemble particulier limité et fermé de processus de gestion sont traitées dans la même application logicielle. Les mises à jour d'un module ou d'une composante applicative sont reflétées dans toute la logique du processus de gestion, sans interface externe complexe. Les données sont stockées une seule fois et sont

instantanément partagées par tous les processus métiers activés par l'application logicielle.

Selon Manouvrier et Menard 2008 [43]

L'intégration d'applications est un ensemble de méthodes, d'outils et de services qui travaillent ensemble pour mettre en communication des applications hétérogènes, dans le cadre d'une entreprise traditionnelle, distribuée ou étendue.

Ces trois définitions mettent en évidence la complexité que représente l'intégration d'applications. Elle nécessite de faire communiquer des applications reposant sur des technologies différentes, s'exécutant sur des plateformes hétérogènes et distantes, d'en gérer les données et les processus avec rigueur mais souplesse et ce dans le but de les unifier en processus métiers efficaces, adaptables et extensibles.

1.2.1 Intégration d'applications : Problèmes, considérations et conséquences

Comme nous venons de le voir, l'intégration d'applications est complexe. Elle implique un travail conséquent d'analyse et de compréhension des processus de l'entreprise et des systèmes qu'elle utilise. En 2012, Soomro et Awan [65] rapportent que 70% des projets d'intégration échouent et énumèrent les principaux problèmes rencontrés dans de tels projets :

— La pénurie d'experts dans le domaine

— La méconnaissance de l'intégration d'applications. Elle n'est pas un outil ou un produit mais une architecture.

— La négligence des aspects sécuritaires, de performance et de surveillance.

— La mise en œuvre de l'intégration dans le cadre d'autres projets.

— L'absence de stratégie d'intégration.

— Les politiques internes et le manque de communication.

— Les changements constants.

— Les normes en compétition.

Pour Woolf et Hohpe, elle implique également une série de considérations et conséquences à évaluer ainsi que la tenue d'une importante réflexion préalable tenant compte de plusieurs critères [79, p. 39-41] dont :

— Le couplage des applications. Une solution d'intégration devrait minimiser le couplage entre les applications intégrées de façon à réduire les suppositions qu'elles font les unes par rapport aux autres et les impacts des modifications portées sur chacune d'elles.

— L'intrusivité. Elle mesure les modifications qu'impose la solution d'intégration choisie au sein même des applications. Elle est souvent inévitable.

— La sélection d'une technologie. Celle-ci peut nécessiter de la part des développeurs des connaissances spécifiques, augmenter leur courbe d'apprentissage et mener à des blocages (pour les solutions propriétaires). Cependant, créer une nouvelle solution est chronophage en ressources et souvent équivalent à réinventer la roue.

— Le format de données. Les applications doivent s'accorder afin d'échanger des informations. Il est en effet presque systématiquement impossible de changer le format de données qu'une application utilise pour se conformer aux besoins d'une autre application. Un intermédiaire pourra jouer ce rôle de médiation et de transformation.

— La latence des données. Elle définit le délai entre le moment ou une donnée est créée dans son application maître et le moment ou elle est disponible dans une autre application. Idéalement une donnée devrait être disponible dès le moment ou elle est produite. Le contexte des applications intégrées définira les besoins en matière de latence des données.

— Les données ou les fonctionnalités ? Dans une solution d'intégration, des applications peuvent partager non seulement des données à un faible niveau d'abstraction mais aussi des fonctionnalités. Partager des fonctionnalités permet d'élever le niveau d'abstraction de la communication entre applications. Partager des fonctionnalités n'est cependant pas aussi simple qu'un appel de procédure en mémoire au sein d'une application.

— La communication à distance. Elle est typiquement asynchrone contrairement à ce que l'on peut rencontrer dans un processus typique. Cette asynchronicité implique une réflexion différente des développeurs car elle produit des solutions plus complexes à concevoir, à développer et à débugger.

— La fiabilité. Dans les communications distantes, la fiabilité n'est pas toujours au rendez-vous. Les applications communiquant à distance ne peuvent donc supposer qu'elles seront disponibles au même moment. Des mécanismes de prise en charge de cette fragilité doivent être prévus.

A la lecture de ces quelques lignes, on comprend la difficulté que représente l'intégration. Elle nécessite

1. l'établissement d'une stratégie spécifique ;
2. le soutien de cette stratégie par l'organisation ;
3. de disposer de l'expertise nécessaire ;
4. de concevoir une architecture d'intégration suffisamment souple pour s'adapter le plus aisément possible aux changements incessants et aux normes en vigueur ;
5. la prise en compte de nombreux critères.

Parmi les critères évoqués, le couplage est primordial et mérite qu'on s'y attarde quelque peu.

1.2.2 Intégration et notion de couplage

Parmi les critères que nous venons de citer, le couplage est un des paramètres qui influencera le plus la qualité d'une solution d'intégration. Il nous parait donc important d'y consacrer quelques lignes afin de clarifier cette notion.

La couplage dans le domaine du génie logiciel De manière généraliste, le couplage pourrait se définir comme le nombre de suppositions qu'un composant fait par rapport à un autre. Deux composants qui interagissent sont couplés selon un degré plus ou moins important. Plus un composant A fait de suppositions à propos d'un composant B, plus ce degré de couplage sera élevé. Une absence de couplage signifie qu'aucune interaction n'existe entre ceux-ci. Une littérature abondante existe sur le sujet. Ce travail traitant de l'intégration d'applications, nous nous focaliserons sur l'établissement de critères permettant d'évaluer le niveau de couplage d'une communication inter-systèmes.

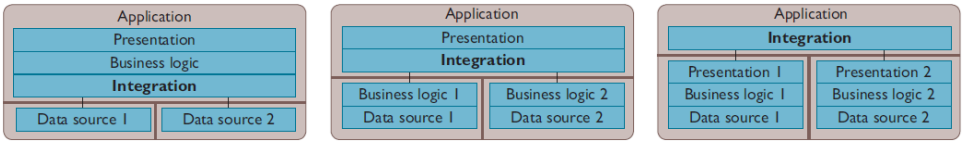
Le couplage dans le domaine de l'intégration d'applications L'intégration d'applications consiste à faire communiquer des applications reposant sur des technologies différentes, s'exécutant sur des plateformes hétérogènes et distantes, d'en gérer les données et les processus avec rigueur mais souplesse et ce dans le but de les unifier en processus métiers efficaces, adaptables et extensibles. L'adaptabilité et l'extensibilité d'une solution d'intégration dépendent fortement du couplage lié à la communication inter-applications. Debasish indique les avantages d'un faible couplage. Il permet de réduire la dépendance, d'augmenter la flexibilité, la scalabilité et de faciliter la tolérance aux pannes [18]. Eugster et al. citent 3 dimensions selon lesquelles le découplage d'une communication entre un service et un client peut être décomposé [23]. Il s'agit des dimensions spatiale, temporelle et de synchronisation. Aldred et al. les développent dans [1]. Delamer et Lastra ajoutent deux dimensions supplémentaires que sont le découplage d'implémentation et le découplage d'infrastructure [19]. Synthétiquement, le (dé)couplage entre deux systèmes communicants peut donc être évalué selon 5 critères :

1. Le (dé)couplage spatial. Deux systèmes sont faiblement couplés si ils n'ont pas besoin de se connaître l'un et l'autre. Il sont fortement couplés si ils utilisent un adressage directe entre eux.
2. Le (dé)couplage temporel. Deux systèmes sont faiblement couplés si ils ne doivent pas être disponibles au même moment. Autrement dit, une interaction entre deux systèmes peut être entamée même si un des deux systèmes est indisponible.
3. Le (dé)couplage de synchronisation. Deux systèmes sont faiblement couplés si l'interaction qui les lie n'est bloquante ni pour l'un ni pour l'autre. Le concept principal de ce critère est donc celui de communication non bloquante pour un ou pour les deux systèmes concernés par une interaction.
4. Le (dé)couplage d'implémentation. Deux systèmes sont faiblement couplés si différents langages et plateformes peuvent être utilisés pour leurs implémentations.
5. Le (dé)couplage d'infrastructure. Deux systèmes sont faiblement couplés si leurs interactions sont indépendantes de l'infrastructure de communication sous-jacente. Celle-ci peut acheminer les données, requêtes, messages directement du producteur au consommateur ou utiliser des intermédiaires.

Nous utiliserons ces critères pour évaluer le niveau de (dé)couplage des styles et architectures d'intégration d'applications que nous décrirons dans les sections 1.3 et 1.4.

1.2.3 Niveaux de l'intégration d'applications

Dans la section 1.1 nous avons vu que l'intégration s'étend aussi bien au niveau organisationnel (on parlera alors de coordination) qu'au niveau des systèmes informatiques d'un organisation. Ce dernier niveau comprend trois couches distinctes : les données, les applications et les processus métier. Selon Ruh, Maginnis et Brown[60] et Daniel et al.[17], l'intégration d'applications peut se diviser en trois approches présentés sur la figure 1.2.



(a) Intégration des données **(b)** Intégration de la logique métier **(c)** Intégration des interfaces utilisateurs

Figure 1.2 – Approches d'intégration d'applications [17]

Les applications peuvent en premier lieu être intégrées au niveau de leurs données. Dans cette approche, schématisée sur la figure 1.2a, les informations circulent directement entre les sources de données. Chaque application dispose de son propre interface utilisateur et de sa propre logique métier. Techniquement cette approche est facile à réaliser. Elle impose cependant de comprendre en profondeur les schémas de données et de résoudre les disparités syntaxiques et sémantiques. Elle imposera également d'être particulièrement attentif à la modification de l'une ou l'autre source de données. Une seconde approche propose d'intégrer la logique métier exposée par les applications. L'intégration se réalise dans ce cas en tirant parti des différents API exposés par les applications. Dans cette approche, les données sont encapsulées dans la logique métier. Il n'est donc plus nécessaire de se soucier de leur traitement. De plus, les API sont iso-fonctionnelles lors des diverses modifications apportées aux applications, stabilisant de ce fait l'intégration. La troisième et dernière approche possible est celle de l'intégration d'interfaces utilisateur. Elle propose de grouper des applications à travers leurs couches de présentation. Cette approche est particulièrement appropriée lorsque les deux approches précédentes sont inapplicables. Diverses techniques que nous ne détaillerons pas ici comme le screen scraping, l'agrégation de contenu, la composition d'interfaces utilisateur de type desktop ou plug-in web, le web mashups ou encore les portails web peuvent être utilisées.

La figure 1.3 synthétise le cadre général de l'intégration que nous avons brossé dans la section 1.1 et les approches d'intégration d'applications que nous venons de décrire.

1.3 Styles/méthodes d'intégration d'applications

Nous avons jusqu'à présent donné un cadre à l'intégration et approfondi l'intégration d'applications ainsi que la notion de coupage qui y est liée. Dans cette section, nous explorerons les différents styles d'intégration tandis que les architectures d'intégration le serons dans la section suivante. Nous les illustrerons à l'aide d'exemples inspirés de notre cas d'accompagnement, évaluerons leur degré de (dé)couplage

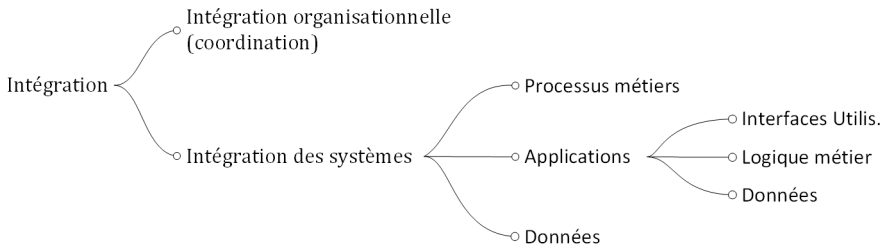


FIGURE 1.3 – Niveaux de l'intégration

selon les critères définis dans la section 1.2.2 et énumèrerons leurs avantages et inconvénients. La notation suivants sera utilisée pour ces derniers.

- \+ indiquera un avantage.
- ≈ indiquera un attribut qui peut, selon différents facteurs, constituer soit un avantage soit un inconvénient.
- – indiquera un inconvénient.

1.3.1 File transfer

Dans ce style d'intégration une application exporte dans un fichier des données dont elle est responsable et qui sont nécessaires à une autre application. Ce fichier est ensuite consommé par cette autre application. Il s'agit donc d'une méthode de transfert de données. Dans ce style d'intégration les applications devront se mettre d'accord sur le type et le format de fichier utilisés lors de l'échange. XML est un standard de fichier couramment utilisé à cette fin de même que JSON et les fichiers CSV. Enfin d'un point de vue protocole, FTP, SFTP, HDFS, SCP, SMB et CIFS sont ceux les plus souvent associés à ce style.

Exemple File transfer peut tout à fait être appliqué à notre problème. La figure 1.4 illustre l'admission d'un patient par le PMS. Celui-ci génère un fichier HL7 [1] sur un partage de fichier à l'aide du protocole SMB. L'EPR récupère le fichier à l'aide de ce même protocole et enregistre l'admission du patient.

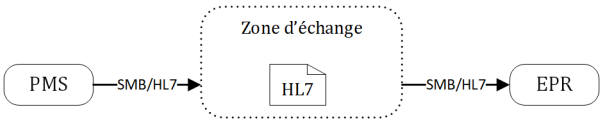


FIGURE 1.4 – Le style file transfer utilisé pour transmettre les données et mouvements de patients grâce à des fichiers HL7 transmis par l'intermédiaire d'une zone d'échange.

Découplage :

1. Norme standardisée de message utilisée dans le domaine médical pour l'échange, l'intégration, le partage et la récupération de renseignements électroniques administratifs et médicaux.

Spatial	oui	Implémentation	oui
Temporel	1	Infrastructure	non
Synchronisation	2		

1. Dépend des détails de l'accord entre les systèmes coopérants.
2. Dépend des détails d'implémentation du côté du producteur comme de celui du consommateur.

Avantages et inconvénients :

+ Transparent aux technologies et plateformes, il s'agit d'un style universel.
+ Les développeurs ne doivent pas connaître le fonctionnement des diverses applications.
+ Style simple qui ne nécessite par d'outil supplémentaire.
≈ Sera plus ou moins compatible avec un firewall en fonction des protocoles de transport utilisés.
– La gestion des erreurs de traitement peut être problématique.
– Ressources système importantes pour la génération et le traitement des fichiers
– Les applications doivent s'accorder sur :
  le nommage des fichiers ;
  le type des fichiers ;
  la localisation de ceux-ci ;
  une syntaxe des données ;
  une sémantique commune des données ;
  éventuellement s'assurer d'une séquence particulière de traitement ;
  une synchronisation ou un mécanisme de verrouillage ;
  la gestion des fichiers après le traitement.
– Une fréquence trop faible de synchronisation peut mener à des incohérences qu'il est difficile de résoudre

1.3.2 Shared database

Le concept de ce style d'intégration est simple. Il repose sur l'édification d'une base de données unique et commune à tous les systèmes informatiques d'une organisation. Cette approche implique la conception difficile d'un schéma unifié de base de données avec lequel les programmeurs peuvent trouver laborieux de travailler et qui peut rencontrer des problèmes politiques. En effet, certaines entités de l'organisation peuvent souhaiter conserver des données critiques au sein de leurs propres bases de données. De plus ce style peut amener certaines applications critiques à subir des temps de réponse trop importants, menant à des pressions importantes pour les séparer du système commun. Enfin, les éditeurs de logiciels utilisent leurs propres schémas de données et se réservent souvent le droit de les modifier lors de mises à jours de leurs applications. En résumé, bien que séduisant d'un point de vue conceptuelle, ce style est difficile à mettre concrètement en œuvre. Il mène souvent à un schéma trop complexe, trop général ou encore un mélange des deux.

Exemple Dans le cas d'un hôpital, l'ensemble ou un sous ensemble du HIS pourrait utiliser une base de données partagée. Cependant, les systèmes externes ne pourront jamais y être intégrés impliquant la nécessité d'utiliser un ou plusieurs autres styles d'intégration comme le montre la figure 1.5.

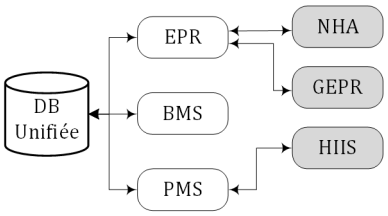


Figure 1.5 – Le style shared database appliqué aux systèmes internes du HIS. Il impose l'usage d'autre styles d'intégration vers les systèmes externes.

Découplage :

Spatial	oui	Implémentation	2
Temporel	1	Infrastructure	non
Synchronisation	oui		

1. Les systèmes sont temporellement indépendants. La disponibilité de la base de données est cependant critique dans ce style.
2. Les systèmes peuvent être implémentés dans des langages différents mais le système de gestion de la base de données sera commun.

Avantages et inconvénients :

+ Il n'est pas nécessaire d'avoir recours à une couche d'intégration. L'intégration est résolue.
+ Pas de risque d'incohérence des données.
+ Pas de risque de dissonance sémantique des données.
+ Forte indépendance fonctionnelle entre applications.
– Absence d'encapsulation des données
– Difficulté d'arriver à un schéma DB qui satisfait toutes les applications
– La plupart n'acceptent de fonctionner qu'avec leur propre schéma
– Les vendeurs se réservent généralement la possibilité de changer le schéma de leurs applications, par exemple lors de nouvelles versions
– Difficulté de modification du schéma
– Possibilité de redondance d'informations pour satisfaire des besoins orthogonaux.
– Possibilité de dégradation des performances (goulots d'étranglement)
– Faible résistance aux transactions
– Plus forte probabilité d'occurrence de deadlock

1.3.3 Remote procedure call (RPC)

L'appel de procédure distant est une des formes d'interactions distribuées la plus utilisée. RPC fut initialement utilisé dans les langages procéduraux ce qui explique sa qualification en « Remote Procedure Call » et non pas en « Remote Method Call ». Il a rapidement été appliqué à la programmation orientée objet [23]. RPC envisage un serveur comme un ensemble de procédures que les clients appellent pour effectuer une tâche [84]. RPC désigne donc une capacité d'un système à invoquer des méthodes d'un autre système possiblement exécuté sur une machine distante. De nombreuses technologies, réparties en deux grandes familles, permettent de mettre en œuvre RPC.

Objets distribués

CORBA CORBA signifie *Common Object Request Broker Architecture*. Il s'agit d'un framework qui permet l'usage d'objets distribués sur des périphériques matériels, des systèmes d'exploitation, des protocoles réseau et des langages de programmation hétérogènes. Dans CORBA, un objet est représenté par son interface et est implémenté dans un programme orienté objet comme un objet local appelé « servant ». L'interface de cet objet est défini selon un langage appelé IDL pour *Interface Definition Language*. Un compilateur IDL est ensuite utilisé pour générer le code du *stub* et du *skeleton*. Le *stub* représente le servant localement dans l'espace d'adressage du client tandis que le *skeleton* représente le client localement dans l'espace d'adressage du servant [61]. Le *stub* et le *skeleton* utilisent ensuite du marshaling pour transmettre une requête sur le réseau.

RMI RMI est l'acronyme de *Remote Method Invocation*. Il a été proposé par JavaSoft pour permettre le développement d'applications distribuées basées sur Java. Contrairement à CORBA, RMI ne fonctionne donc que sur le langage java et sur les systèmes d'exploitation utilisant la machine virtuelle Java. RMI dispose d'un registre dans lequel un objet serveur s'enregistre et que le client interroge pour obtenir l'adresse et le bytecode de l'objet server. RMI fait usage de la sérialisation pour la transmission des requêtes.

DCOM DCOM pour *Distributed Component Object Model* est également un framework d'objets distribués proposé par Microsoft comme une extension de son *Component Objet Model*. Comme CORBA, DCOM est langage agnostique mais contrairement à CORBA et RMI, il ne supporte que les plateformes Windows. Le fonctionnement de DCOM est similaire à celui de CORBA à ceci près qu'un *stub* se nomme un *object stub* tandis qu'un *skeleton* s'appellera un *objet proxy*.

gRPC gRPC est un framework RPC open-source, haute performance, multi-langage et multi-plateforme initialement développé par Google. Il est récent et utilisé dans de grosses structures tels que Google, Netflix ou encore CISCO [26]. Le principe de base est le même que celui de CORBA et DCOM. gRPC utilise *protocol buffers* comme langage de définition de l'interface et du contenu d'une requête. Le code du client et du serveur sont générés à partir de cette définition à l'aide du compilateur procotol buffers du langage cible. gRPC supporte de nombreux langages.

Services web

Comme le mentionnent Alonso et al., le terme service web est très couramment utilisé de nos jours bien que pas toujours avec la même signification. Les définitions existantes vont des plus généralistes au plus spécifiques et restrictives [2]. La définition donnée par le consortium UDDI nous semble la plus appropriée. Il définit un web service comme « Une application métier modulaire et autonome qui possède des interfaces ouverts, basés sur des standards et orientés internet ». SOAP et REST sont les deux types de web service couramment utilisés.

SOAP SOAP est un langage XML qui définit une architecture et un format de message. Une message SOAP est constitué de deux parties, l'entête et le corps. L'entête est un conteneur extensible qui peut être utilisé entre autre pour fournir des informations de routage et de configuration de qualité de service. Le corps du message contient le contenu utile du message. Un service web SOAP est donc basé sur l'échange de message SOAP. L'interface d'un web service de ce type est défini selon le langage *Web Service Description Language* (WSDL). Il permet de décrire le port ainsi que le format des opérations et messages d'un web service SOAP. Cette description est typiquement enregistrée par un fournisseur de service dans un annuaire permettant aux clients de rechercher et utiliser dynamiquement un web service SOAP. Dans la mesure ou un service web SOAP permet l'exécution d'une méthode sur un objet distant, il peut être assimilé à un mécanisme RPC [78].

REST Les services web de style REST reposent sur 4 principes [55] :

1. Identification d'une ressource par une URI - Un service web REST expose un ensemble de ressources qui identifient les cibles de l'interaction avec ses clients. Les ressources sont identifiées par des URI, qui fournissent un espace d'adressage mondial pour la découverte de ressources et de services.
2. Interface uniforme - Les ressources sont manipulées à l'aide d'un ensemble fixe de quatre opérations de création, de lecture, de mise à jour, de suppression : PUT, GET, POST, et DELETE. PUT crée une nouvelle ressource, qui peut ensuite être supprimée à l'aide de DELETE. GET récupère l'état actuel d'une ressource dans une certaine représentation. POST transfère un nouvel état sur une ressource.
3. Messages auto-descriptifs - Les ressources sont découplées de leur représentation afin que leur contenu puisse être consulté dans divers formats. Les méta-données sur la ressource sont disponibles et utilisées, par exemple, pour contrôler la mise en cache, détecter les erreurs de transmission, négocier le format de représentation approprié et effectuer l'authentification ou le contrôle d'accès.
4. Interactions avec les états par le biais d'hyperliens - Chaque interaction avec une ressource est sans état, c'est-à-dire que les messages de requête sont autonomes. Une requête doit contenir toutes les informations nécessaires à son exécution.

Contrairement à SOAP, REST n'invoque pas des méthodes distantes au sens stricte du terme. Toutefois, les verbes HTTP (PUT,GET,POST,DELETE) utilisés par REST, permettent de réaliser des opérations à distance sur des ressources. De ce fait, nous l'avons assimilé à un mécanisme RPC.

Exemple Dans un hôpital, les données médicales d'un patient sont enregistrées dans le système EPR. De nos jours, beaucoup de pays implémentent un dossier médical global pour leurs citoyens. En fin d'hospitalisation, l'EPR envoie les données médicales utiles dans ce dossier global. Dans notre exemple, le GEPR expose des services web SOAP dédicacés à cette usage. Il les a préalablement publiés sur un annuaire de services que l'EPR interroge au runtime pour obtenir la localisation du service et initier la communication.

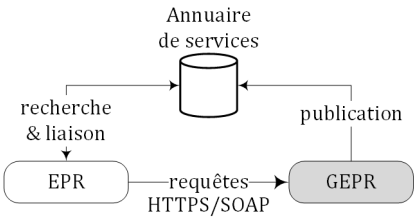


Figure 1.6 – Un exemple de communication de style RPC par l'intermédiaire d'un web service SOAP.

Découplage :

Spatial	1	Implémentation	3
Temporel	non	Infrastructure	4
Synchronisation	2		

1. Dépend de la technologie utilisée.
2. Dépend de la technologie et du paradigme d'échange.
3. Dépend de la technologie utilisée. Par exemple, RMI impose l'utilisation de Java.
4. Dépend de la technologie utilisée.

Avantages et inconvénients

+ L'encapsulation des données permet d'assurer l'intégrité de celles-ci.
+ Chaque application peut modifier son schéma de données sans effet pour les autres applications.
+ Il est plus facile de gérer la dissonance sémantique.
+ Les technologies et les frameworks sont matures.
+ Du point de vue du programmeur, les objets distribués facilitent la conception d'applications distribuées. Un appel de procédure distante est similaire à un appel local.

≈ La facilité d'intégration de systèmes séparés par un firewall dépendra de la technologie utilisée.

– Les appels de procédures distantes bien que similaires à des appels locaux de procédures du point de vue du programmeur, sont cependant plus lents et moins fiables. Ils nécessitent donc une réflexion différente de la part des développeurs.

– Les RPC ne sont pas adaptés à la communication de grosses quantités de données ou pour la communication à travers des réseaux qui présentent une latence importante [16].

1.3.4 Method wharehousing

Ce style, décrit par Laftsidis, propose de partager la logique métier d'une organisation entre les systèmes de cette organisation. Ceci est réalisé en centralisant cette logique dans un entrepôt de méthodes accessible par l'ensemble des systèmes [37]. Les processus d'une organisation sont donc agrégés et hébergés sur un serveur centralisé avec pour conséquences : la conversion de toutes les méthodes communes et de toutes les applications dans une technologie commune ; la réécriture de ces applications qui devront invoquer à distance les méthodes de l'entrepôt. D'un point de vue implémentation, n'importe quelle technologie permettant l'appel à distance de méthodes pourra être utilisée (voir section 1.3.3).

Exemple Dans un hôpital, l'admission d'un patient sera réalisée via des appels distants aux procédures de l'entrepôt selon la séquence suivante :

1. Si le patient est inconnu, son dossier est créé par le PMS via une procédure *createPatient*. S'il est connu, le PMS met éventuellement son dossier à jour via une procédure *updatePatient*. Ces procédures prennent en charge l'inscription de ces données dans l'EPR.
2. Ensuite le PMS enregistre l'admission du patient en appelant la procédure *admitPatient*. Celle-ci inscrit également l'admission dans l'EPR et le BMS.

Bien que ce style puisse être appliqué sur les systèmes internes d'un hôpital, la figure 1.7 met en évidence son inadéquation pour les systèmes externes. Une organisation ne possède en effet aucun contrôle sur le code de ceux-ci. D'autres styles d'intégration devront donc être utilisés pour communiquer avec ces systèmes.

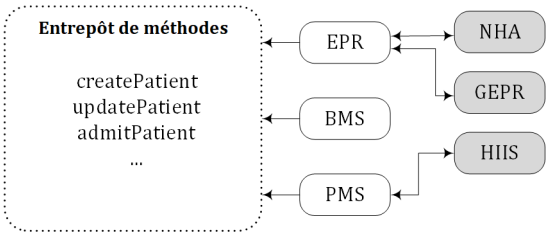


Figure 1.7 – Le style method wharehousing. Son inappliquabilité aux systèmes externes impose l'utilisation d'autres styles pour communiquer avec eux.

Découplage :

Spatial	oui	Implémentation	2
Temporel	oui(1)	Infrastructure	oui
Synchronisation	oui		

1. Un système est dépendant de la disponibilité de l'entrepôt mais pas de celle des autres systèmes.
2. Lié à la technologie utilisée pour l'entrepôt. Les web services permettront un découplage au niveau de l'implémentation, RMI ne le permettra pas puisqu'il impose le langage Java.

TABLE 1.1 – Taux de couverture des besoins informatiques par les ERP selon [70].

Level of IT requirements covered by EPR solution	Percentage
Highest (80-100%)	0
Higher (70-80%)	0
High (60-70%)	2
Medium (50-60%)	20
Low (40-50%)	50
Lower (30-40%)	22
Lowest (0-30%)	0

Avantages et inconvénients :

+ Bonne solution pour l'intégration des processus métiers internes à l'organisation.
+ La logique d'intégration est centralisée.
+ Témoigne d'une réflexion globale de stratégie d'intégration.
+ Les choix technologiques d'implémentation sont variés.
≈ En fonction de l'infrastructure mise en place, l'entrepôt pourrait :
  - Présenter une résistance aux pannes plus ou moins bonne.
  - Présenter d'éventuelles problèmes de performance.
– Très couteuse à mettre en place.
– Très longue à mettre en place.
– Nécessite d'avoir accès au code des applications. Quid des systèmes propriétaires ?

1.3.5 Enterprise Resource Planning (ERP)

ERP est l'acronyme de Enterprise Resource Planning. Pour BOTTA-GENOULAZ et MILLET il se définit comme un logiciel qui tente d'intégrer tous les départements et fonctions d'une entreprise dans un système informatique unique pouvant répondre aux besoins des différents départements [10]. Plus précisément, un ERP est un ensemble d'applications ou de modules métiers couvrant et liant une partie plus ou moins importante (possiblement l'ensemble complet) des métiers d'une organisation. Ces applications et modules partagent le même modèle de données, le même schéma de données ainsi que les mêmes interfaces [66].

Théoriquement, ces caractéristiques font des ERP une solution au problème de l'intégration. En effet, l'ensemble des processus métiers de l'organisation étant rassemblé dans un package logiciel unique, il n'est plus nécessaire d'établir de communication vers d'autres applications. Cependant, un ERP ne permet que rarement de remplacer tous les logiciels d'une organisation et ne prend en charge qu'une fraction des fonctionnalités métiers de l'entreprise [79, 46, 70]. La tableau 1.1 présente le taux de couverture des besoins informatiques par les ERP selon les résultats d'une enquête menée par THEMISTOCLEOUS, IRANI et O'KEEFE en 2001 [70]. Il illustre parfaitement la nécessité d'une cohabitation des ERP avec les autres systèmes applicatifs.

Exemple Un ERP pourrait être utilisé dans un hôpital pour couvrir des domaines fonctionnels tels que la gestion des ressources humaines, de la facturation et de la comptabilité. Cependant, les systèmes externes ne pourront être pris en charge par l'ERP de même que certains systèmes internes spécialisés. La figure 1.8 illustre la nécessité d'une intégration entre l'ERP et ces systèmes.

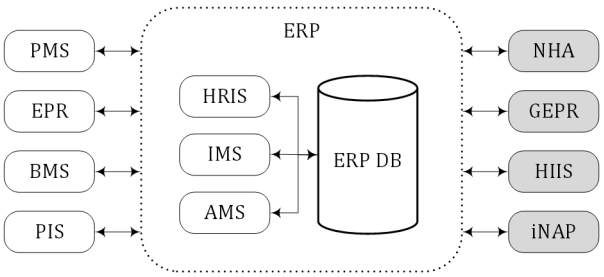


FIGURE 1.8 – Le style ERP. Alors que certains domaines fonctionnels sont pris en charge par l'ERP, d'autres le sont par des systèmes internes ou externes hors ERP. Une intégration est donc toujours nécessaire.

Découplage :

Conceptuellement, un ERP est vu comme un système unique. Il n'existe donc pas de couplage au sens de l'intégration d'applications.

Avantages et inconvénients :

+ Suppression du problème de l'intégration pour les aspects métiers pris en charge par l'ERP.
+ Au sein de la solution, l'information est disponible sans délai.
– Le coût d'une solution ERP est généralement très conséquent (Licence, analyse, implémentation, formation du personnel, achat de modules complémentaires).
– Ils peuvent être difficile à intégrer avec d'autres systèmes. Selon une enquête menée par THEMISTOCLEOUS, IRANI et O'KEEFE en 2001 82% des répondants citent l'intégration de logiciels existants et 46% l'intégration de nouveaux logiciels comme problèmes techniques rencontrés pendant (ou après) la période de mise en œuvre d'un ERP.
– Les projets de mise en place d'un ERP sont des projets risqués.

1.3.6 Extract-Transform-Load (ETL)

Le style ETL est un modèle de processus essentiellement utilisé dans le domaine des datawarehouses. Un datawarehouse est une collection de bases de données intégrées et orientées sujets conçu pour appuyer les processus de prise de décision. En intégrant différentes sources de données, un datawarehouse permet d'obtenir

une vue complète et exacte des données opérationnelles d'une organisation, synthétisée dans un ensemble d'indicateurs ou de mesures stratégiques [21]. Il représente dans un schéma global des données issues de sources hétérogènes internes et externes. Ces sources peuvent être de différentes natures comme des bases de données, des fichiers ou encore des informations issues du web [34].

Comme représenté sur la figure 1.9, ETL est un processus divisé en trois étapes [74].

1. **L'extraction**. Le « Extract » de ETL. Les données sont extraites des sources de données sous forme de snapshots complets ou différentiels.
2. **La transformation et le nettoyage**. Le « Transform » de ETL. Les données sont propagées dans la « Data Staging Area (DSA) » ou elles sont transformées et nettoyées.
3. **Le chargement**. Le « Load » de ETL. Enfin les données sont chargées de la DSA vers le datawarehouse.

Bien que trouvant ses racines dans le datawharehousing, le style ETL peut tout à fait être utilisé pour transférer des données d'une application à une autre, scénario dans lequel il sera particulièrement utile lorsque la quantité de données est importante.

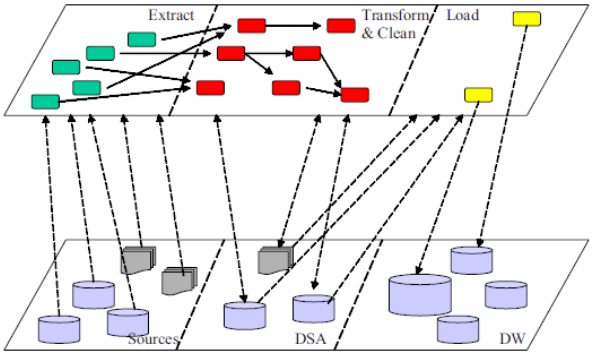


Figure 1.9 – Le style ETL [74]. La couche supérieure représente les 3 étapes du processus ETL tandis que la couche inférieure représente les collections de données sources et cibles intervenant lors de ces différentes étapes.

Exemple Un hôpital comme toute organisation moderne utilise la business intelligence à divers niveaux comme support aux prises de décisions stratégiques. Les données sont extraites des divers systèmes du HIS, subissent d'éventuelles transformations, sont éventuellement stockées dans des bases de données temporaires et sont finalement chargées dans la base de données du système BI pour être exploitées par l'organisation.

Découplage :

Spatial	oui	Implémentation	oui
Temporel	non	Infrastructure	oui
Synchronisation	oui		

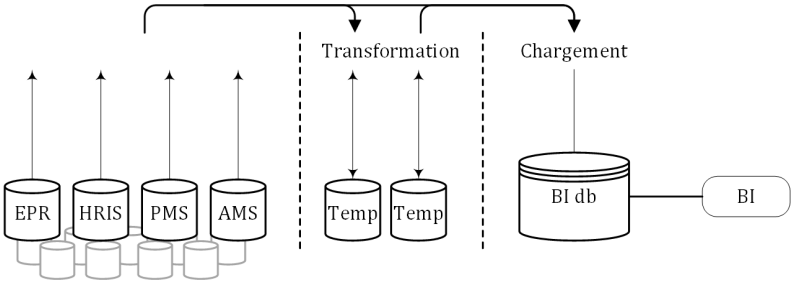


Figure 1.10 – Collecte des données des différents systèmes du HIS, transformation et centralisation dans le système BI.

Avantages et inconvénients :

+ Style adapté à la transmission de grosse quantité de données.
– Nécessite un effort important pour comprendre les schémas des sources de données.
– Nécessite des efforts importants pour maintenir le processus de transformation si le schéma de données d'une source change.
– Possibilité d'introduction d'incohérence au niveau des données cibles lorsqu'il est utilisé pour la communication entre deux applications (absence d'encapsulation des données).

1.3.7 Messaging

Le style messaging constitue une solution élégante et pragmatique au problème des systèmes distribués [79]. Le style messaging peut être comparé au système des courriers électroniques. Un destinateur envoie un mail vers un destinataire grâce à son adresse mail. Le mail transite par une série de serveurs sans que le destinateur n'ait à se soucier de la remise de celui-ci, l'infrastructure la prenant en charge. Le destinateur ne doit pas non plus se soucier de savoir si l'ordinateur et le client de messagerie du destinataire sont ouverts au moment ou il envoie son message. L'infrastructure se charge également de cette gestion. Le style messaging utilise le même paradigme de fonctionnement. Les applications envoient des messages dont la remise est gérée par un réseau de serveurs qui constituent le bus de messages.

Woolf et Hohpe en 2004 ont publié sous le titre « Interprise Integration Patterns : Designing, Building, and Deploying Messsaging Solutions » un ouvrage de référence en la matière. Ils y décrivent un langage des modèles de flux de messages constituant une référence encore aujourd'hui [85]. Ces modèles, visibles sur la figure 1.11 mettent en exergue les capacités du style. Les détails de l'ensemble du modèle sont disponibles dans l'ouvrage et sur le site qui y est consacré : `https://www.enterpriseintegrationpatterns.com/patterns/messaging/index.html`.

Exemple La figure 1.12 montre la communication de nos trois applications selon le style messaging. D'une part, elles déposent des messages sur le bus et d'autre part, elles assurent le traitement de ceux qu'elles reçoivent. On constate qu'il n'y a plus de communication directe entre les applications. Le bus prend en charge le routage du message vers le ou les bons destinataires, garantit la

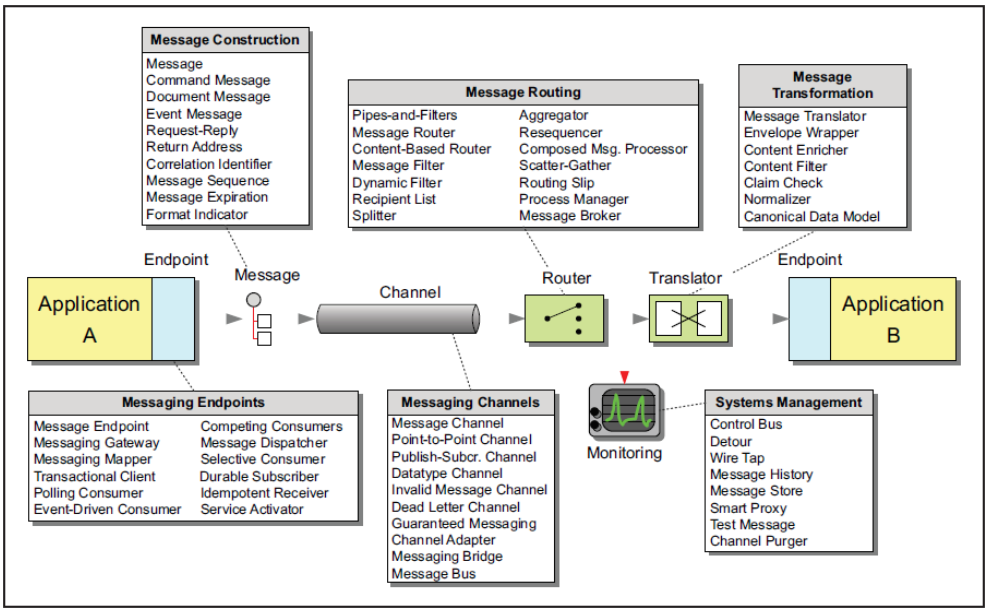


FIGURE 1.11 – Langage des modèles de flux de messages proposés par WOOLF et HOHPE en 2004.

livraison du message et enfin peut effectuer des transformations sur le format des données transmises pour convenir à celui requis par l'application cible.

Exemple

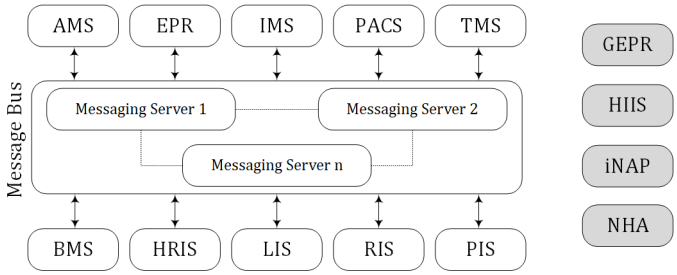


FIGURE 1.12 – Le style messaging

Découplage :

Spatial	oui	Implémentation	oui(3)
Temporel	oui(1)	Infrastructure	non
Synchronisation	2		

1. Oui dans le concept du messaging mais dépendra des paradigmes d'interactions utilisés (request/-reply, fire and forget, ...).
2. Elle sera souvent atteinte du côté du producteur de message et non du côté du consommateur.
3. Dépendra des capacités des langages utilisés à faire usage de la technologie de l'infrastructure de messaging.

Avantages et inconvénients :

+ La remise des messages peut être garantie. Il n'y a pas, dans ce cas, de nécessité que deux systèmes soient disponibles au même moment pour communiquer.
+ Le principe d'encapsulation des données est respecté.
+ Possibilité de mettre en œuvre des scénarios plus complexes comme du routage, du broadcasting ou encore de la transformation de messages.
+ Conceptions des applications et de l'intégration clairement séparées.
+ Permet une intégration aussi bien au niveau des données qu'au niveau application.
– La programmation asynchrone est moins répandue chez les programmeurs et nécessite l'adoption d'un autre paradigme de conception.
– Implémentation plus complexe.
– Les tests et le débuggage sont plus compliqués.

1.4 Architectures d'intégration d'applications

Au niveau architectural, l'intégration d'applications pourra se concevoir selon plusieurs modèles différents. Notre revue de la littérature [6, 11, 27, 36, 58] en identifie quatre. Ils utilisent, en fonction des circonstances et des possibilités offertes par les applications intégrées, les différents styles que nous avons décrits dans la section précédente . Chaque architecture présentée sera appliquée à notre cas compagnon.

1.4.1 Architecture point à point

Ce modèle d'architecture d'intégration est le plus ancien et celui qui présente la plus faible complexité. Il propose, lorsque deux applications A et B doivent communiquer, d'établir un lien direct entre ces deux applications. On crée donc un interface I_{AB} entre ces deux applications. Si une application C vient s'ajouter et qu'elle doit à son tour communiquer avec l'application A, on créera une nouvelle interface I_{AC}. Bien que ce modèle soit simple, il implique la création et la gestion d'une multitude d'interfaces. En effet, pour un système composé de n applications, il sera nécessaire de créer $(n \times (n-1))/2$ interfaces. Il est à noter que ce nombre est théorique et suppose que chaque application doit communiquer avec chaque autre application du système ce qui dans la pratique est rarement le cas. Nous reviendrons sur la prise en compte de cet aspect dans la section 2.4.2 .

Le tableau 1.2 qui indique le nombre théorique d'interfaces pour 5,10,100 et 1000 applications met parfaitement en évidence l'inadaptation de cette architecture aux parcs logiciels dépassant une certaine taille. En effet, pour une organisation utilisant 100 applications, la modification ou le remplacement d'une seule d'entre elles entraînera la modification ou la ré-écriture de 99 interfaces. L'ajout d'une 11ème application impliquera la création de 100 nouveaux interfaces. Une architecture d'intégration point à point est souvent le signe d'un manque de réflexion sur une stratégie globale d'intégration. Elle produit une architecture accidentelle ou « architecture spaghetti » résultant d'un empilage continu de solutions ponctuelles d'interfaces. Celle-ci est

fragile, rigide et ne résiste pas facilement à des ajouts ou modifications de l'environnement [13].

Applications	Interfaces/application	Interfaces
5	4	10
10	9	45
100	99	4950
1000	999	499500

TABLE 1.2 – Nombre théorique d'interfaces dans une architecture d'intégration de type point à point

Exemple La figure 1.13 montre l'utilisation d'une architecture d'intégration point à point dans le cadre de notre cas compagnon. Pour les 18 systèmes qui le constituent, un total de 153 interfaces est nécessaire. Le terme « architecture spaghetti » prend tout son sens.

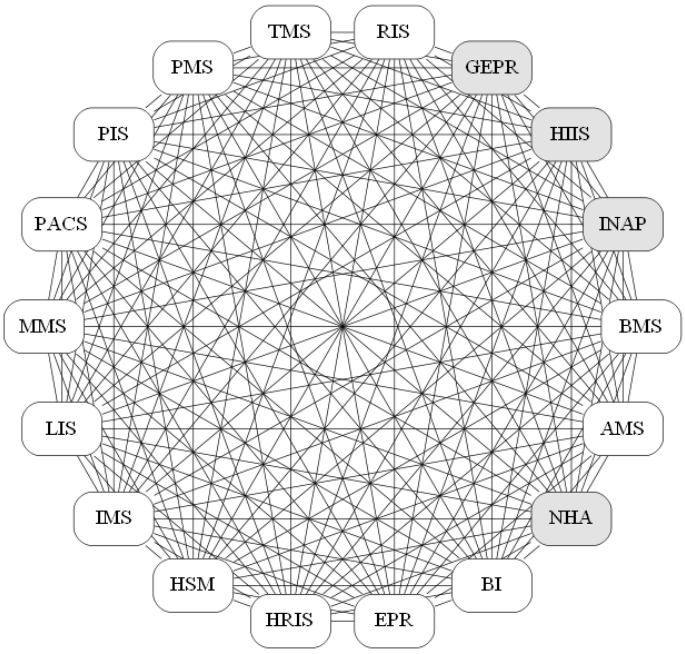


FIGURE 1.13 – Une architecture point à point. Appliquée à notre cas d'accompagnement, elle nécessite 153 interfaces.

Découplage :

Le découplage dépendra du style utilisé pour chaque interface.

Avantages et inconvénients :

+ Simplicité de l'architecture.

+ Utile dans de petites structures avec (très) peu de systèmes.
≈ La facilité d'utilisation à travers un firewall dépendra elle aussi du style utilisé pour chaque interface.
– Dispersion de la logique d'intégration dans l'ensemble du système.
– Aucune gestion centralisée.
– Peut rapidement devenir ingérable.
– Peut entraîner un surcoût important des développements. Une modification, un remplacement, un ajout d'un système peut avoir d'importantes implications.

1.4.2 Architecture hub'n spoke

Nous venons de démontrer que l'architecture point à point ne convenait que pour intégrer un petit nombre de systèmes. Dès lors que leur nombre augmente, une autre option devra être envisagée. C'est dans cette optique que le modèle hub'n Spoke a été conçu. Il repose sur 2 éléments fondamentaux.

1. Le hub - Il est le point central d'interfaçage des applications et fournit des capacités de gestion des files de message, de flux de travail, de sécurité et contrôle d'accès et enfin de transformation des formats de données.
2. Les spokes - Ils constituent le lien entre les applications et le hub. Ils sont eux mêmes composés de deux parties, le protocole de communication (souvent propriétaire) qui établit la communication entre les applications et le hub et les adaptateurs qui sont des artefacts logiciels connectant les applications au hub

Selon Gulledge une architecture hub'n spoke est typiquement utilisée à l'intérieur d'une organisation. Son objectif est d'interfacer les processus et de partager les données entre les systèmes internes. D'autres solutions devront être envisagées pour l'intégration inter-entreprise [27].

Exemple Cette architecture appliquée à notre cas compagnon est illustrée sur la figure 1.14. On notera de suite la réduction du nombre d'interfaces par rapport au modèle point à point et le rôle central du hub qui fait de lui un *single point of failure* (SPOF).

Découplage :

Spatial	oui	Implémentation	oui
Temporel	1	Infrastructure	non
Synchronisation	2		

1. Dépend du style d'échange (Request/reply, fire and forget, etc.).
2. Dépend de la nature des échanges (Synchrone ou asynchrone).

Avantages et inconvénients :

+ Le nombre d'interfaces est réduit au minimum.
+ La logique d'intégration est centralisée au niveau du hub.
+ De même que la gestion globale de la solution d'intégration.

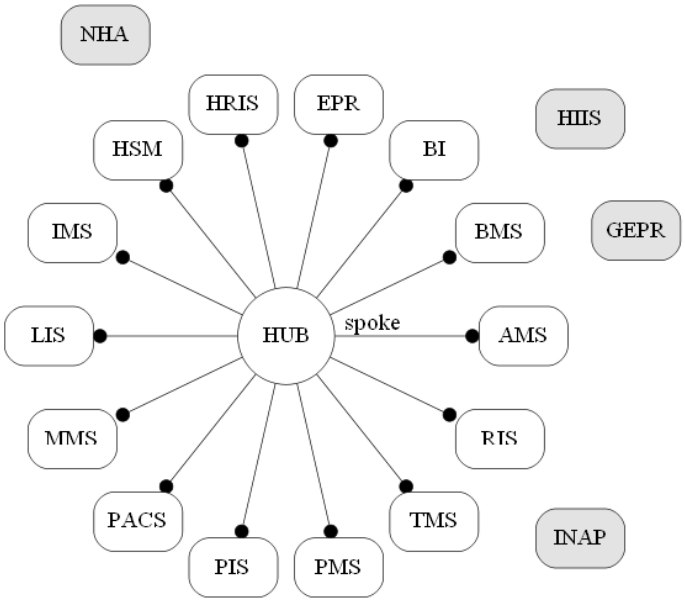


FIGURE 1.14 – Une intégration hub'n Spoke de notre cas compagnon. 1 système = 1 interface. Les spokes et leurs protocoles propriétaires rendent difficiles l'intégration des systèmes externes.

+ Une modification ou un remplacement d'application n'implique que la modification ou le remplacement d'une seule interface.

≈ Le hub constitue un SPOF à moins que des mécanismes appropriés soient mis en place.

– Risque de blocage dû au manque de standardisation des protocoles de communication pouvant mener à la création d'îlot(s) d'intégration et à la difficulté, voir l'impossibilité d'intégrer les systèmes externes.

– Difficulté de fonctionnement si un firewall est présent due à l'utilisation d'un protocole propriétaire de communication entre systèmes et hub.

– Architecture moins souple à cause des technologies propriétaires.

1.4.3 Architecture orientée service (SOA)

L'architecture orientée service est une évolution de l'architecture basée sur les composants, de la conception basée sur les interfaces et des systèmes distribués des années 1990 [73]. L'accessibilité grandissante d'internet a favorisé son apparition. L'intention d'une SOA est de concevoir une infrastructure logicielle autour de services plutôt qu'autour d'applications individuelles. Son objectif est de surmonter de nombreux défis de l'informatique distribuée (intégration d'applications, gestion des transactions et des politiques de sécurité) tout en permettant l'usage de multiples plates-formes et protocoles et en exploitant de nombreux dispositifs d'accès et systèmes existants [54]

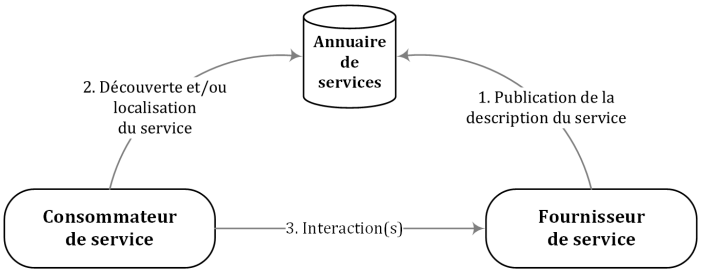


FIGURE 1.15 – Trois composants piliers d'une SOA et leurs interactions

Service Le mot clé de SOA est le mot « Service ». Un service est un module bien défini et autonome qui fourni des fonctionnalités métiers standards et est indépendant de l'état ou du contexte des autres services. Il est constitué d'une interface et d'une implémentation. L'interface définit l'identité du service et sa logistique d'invocation tandis que son implémentation réalise le travail pour lequel le service est conçu [54]. Interface et implémentation sont indépendants. L'interface d'un service est définie dans un langage technologiquement agnostique ce qui permet à un client d'utiliser ce service à partir de n'importe quel langage, système d'exploitation ou plateforme .

Composants Une SOA fait référence à un modèle de conception de systèmes distribués qui délivre des fonctionnalités en tant que services. Elle permet la mise en relation d'un élément qui exprime un besoin vers l'élément (le service) qui est capable de répondre à ce besoin grâce à trois composants fondamentaux.

1. **Les services.** Ils sont fournis par des fournisseurs de services qui publient les interfaces qui les décrivent.
2. **Les consommateurs de services (ou clients).** Ils utilisent les services offerts par les fournisseurs de services par l'intermédiaire de leurs interfaces.
3. **L'annuaire de services.** Il permet d'une part aux fournisseurs de services de publier une description de ceux-ci et d'autre part aux clients de les découvrir et les utiliser.

La figure 1.15 illustre le fonctionnement d'une SOA à travers ces trois composants. En premier lieu, un fournisseur de service s'enregistre auprès de l'annuaire. Dans un second temps, ce dernier est interrogé par le consommateur de service pour découvrir et/ou trouver un service correspondant à ses besoins. Enfin, le consommateur de service peut interagir avec le fournisseur de service grâce aux informations fournies par l'annuaire.

Caractéristiques d'une SOA Selon ERL [22] et VALIPOUR et al. [73], une SOA présente les caractéristiques suivantes :

— Les contrats de service. Les services adhèrent à une entente de communication, telle que définie collectivement par une ou plusieurs descriptions de services.

— La découvrabilité et la liaison dynamique : Une SOA supporte le concept de découverte des services. Un client qui a besoin d'un service découvre sur base

d'un ensemble de critères lequel utiliser ainsi que la localisation de celui-ci au moment de l'exécution.

— L'autonomie : Un service a le contrôle de la logique qu'il encapsule. Du point de vue du client, seul l'interface du service sera visible. La séparation claire de l'implémentation d'un service et de son interface est un aspect primordial d'une SOA. Elle différencie donc clairement le « quoi » du « comment ».

— L'interopérabilité. Une SOA met l'accent sur la capacité à communiquer de systèmes utilisant différentes technologies. Elle mise pour cela sur la définition d'interface et l'usage de protocoles et de formats de données standardisés.

— Le faible couplage. Les services entretiennent une relation qui réduit au minimum les dépendances et qui exige seulement qu'ils conservent une bonne connaissance l'un de l'autre.

— La transparence de la localistion. Un consommateur ne connaît pas la localisation d'un service avant qu'il ne le localise dans l'annuaire de service. Ceci permet le déplacement d'un service de manière transparente pour les consommateurs de ce service.

— La composabilité. Des ensembles de services peuvent être coordonnés et assemblés sous la forme d'applications, de services composites ou encore d'orchestrations.

— L'auto-gestion. Un service doit être capable de récupérer d'une erreur sans intervention humaine. Cette caractéristique devient de plus en plus importante avec l'augmentation de la taille et de la complexité des applications distribuées modernes.

— L'absence d'état. Les services minimisent la rétention d'informations spécifiques à une activité.

D'un point de vue conceptuelle, une SOA ressemblera toujours à l'architecture point à point de la figure 1.13 dans laquelle chaque système est directement en communication avec les autres. L'usage des services web et du protocole HTTP permettra néanmoins une intégration plus aisée des systèmes externes en présence d'un firewall.

Découplage :

Spatial	oui	Implémentation	oui
Temporel	non	Infrastructure	oui
Synchronisation	non		

Avantages et inconvénients :

+ La description des services qui est agnostique d'un point de vue technologique et l'utilisation de standards de communication permet une forte interopérabilité.
+ Une SOA sera souvent firewall friendly. Cet aspect est favorisé par l'usage massif des services web. Ceux-ci utilisent les protocoles HTTP/S qui sont connus et maitrisés dans le cadre d'un firewall.
+ Architecture adaptée aux environnements hétérogènes.
+ Elle fournit une vue globale d'une organisation.

+ Les services sont réutilisables.

≈ Le couplage dépendra pour une part des technologies utilisées et pour une autre des styles d'échanges mis en place.

– La complexité due aux connexions point à point reste présente.

– La logique d'intégration est dispersée et implémentée au sein des systèmes intégrés.

– Pour être mise en place à l'échelle du système, une SOA nécessitera une connaissance pointue du fonctionnement global d'une organisation.

SOA et technologies

Une SOA constitue un modèle de conception architecturale et se réfère à la conception d'un système et non son implémentation [38]. Bien que les services web sont très souvent associés et utilisés pour implémenter une SOA, ils ne constituent qu'une option technologique parmi d'autres[73]. A titre d'exemple, Debasish indique que les premières SOA ont été implémentées sur CORBA qui a la capacité de fournir une connectivité et une interopérabilité basées sur des standards [18]. Chaque technologie supportera plus ou moins bien les caractéristiques que nous venons de décrire.

SOA et micro-services

A première vue, microservices et SOA qui reposent tous deux sur le concept de service, ont beaucoup en commun du point de vue des caractéristiques et de la technique. Ils permettent la conception d'un système sous forme de services disponibles sur un réseau. Cependant, du point de vue conceptuel, ils sont très différents. Une SOA est une architecture système. Elle vise à organiser globalement un système d'information selon les caractéristiques qui la définissent. Les microservices sont une architecture d'application et s'appliquent à des projets individuels [78]. Elle propose d'organiser une application sous forme de services autonomes, chacun s'occupant d'un et un seul aspect précis du domaine fonctionnel traité par l'application. Ce découpage mène à la création de services de tailles réduites, implémentables par de petites équipes de développement et facilitant la scalabilité de la solution. Une architecture microservices produit des bases de code réduites et appréhendables par un seul programmeur [9].

1.4.4 Architecture enterprise service bus (ESB)

Un ESB est une architecture distribuée de messagerie [20]. Elle est constituée d'un bus de service auquel les systèmes à intégrer sont connectés directement ou indirectement grâce à des adaptateurs. Le bus de service est l'épine dorsale de l'architecture ESB par laquelle transitent, sous forme de messages, toutes les interactions entre systèmes intégrés. En opposition à l'architecture hub'n spoke, l'architecture ESB est distribuée. En effet le bus est constitué d'un ensemble de noeuds connectés à travers lequel les messages sont acheminés d'un consommateur de service à un fournisseur de service. Pour Masternak et al. un ESB résulte d'une évolution et d'une convergence naturelle des services web, de l'architecture hub'n spoke

et du style messaging. Ils caractérisent un ESB comme un intermédiaire qui permet de rendre largement disponible un ensemble de services métiers ré-utilisables [45]. Schmidt et al. ajoutent qu'un ESB facilite la mise en place d'une SOA en fournissant une couche de connectivité entre les services [62]. La transformation, la médiation, le routage des messages et l'orchestration sont cités par Garcia-Jimenez, Martinez-Carreras et Gomez-Skarmeta comme les fonctionnalités qu'un ESB doit fournir afin de remplir ce rôle. Selon nous, la médiation doit être vue comme la fonction principale d'un ESB. La transformation (la capacité entre autres à effectuer une conversion entre les protocoles et les formats de données) le routage (la capacité à diriger une requête vers le ou les bons services) ainsi que l'orchestration (la capacité à diriger les étages de transformation et routage) sont des capacités au service de la médiation. Comme le mentionnent Kress et al. dans [35], il n'existe par de spécification qui définit exactement ce qu'est un ESB ou quelles fonctions il devrait fournir. Le présent travail tentera précisément de proposer un modèle de ces fonctionnalités. Ils différencient également la notion d'orchestration tel que précédemment citée de celle d'orchestration ou de chorégraphie de processus métiers. Pour eux, cette fonctionnalité fait partie de la catégorie des système de gestion de workflow. Ceux-ci offrent des environnements d'exécution dédiés aux processus d'entreprise à long terme, optimisés pour cette usage et supportant des langages tels que BPMN ou BPEL. Les ESB doivent être sans état et configurés pour traiter les messages aussi efficacement que possible. Il est à noter que ceci va dans le sens du cadre de l'intégration défini par [66] dont nous avons parlé dans la section 1.1.

Composants d'une architecture ESB Une architecture ESB repose sur 3 composants clés. La figure 1.16 de [11] montre ces 3 composants et leurs relations. Le MOM (Message Oriented Middleware) est constitué d'un réseau de serveurs de messages et constitue l'épine dorsale de l'architecture. Il permet d'établir des canaux

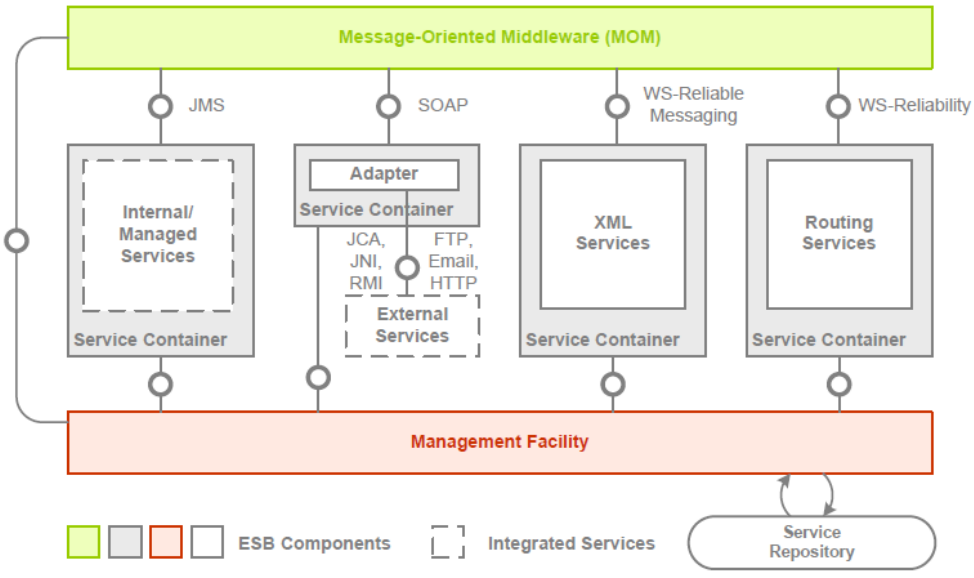


Figure 1.16 – Composants clés d'une architecture ESB et leurs relations.

de communication gérables, sûrs et fiables. Les conteneurs de services permettent de gérer les services internes (e.g. le services de routage ou de manipulation du format XML) de l'ESB et de connecter des applications externes via des adaptateurs. Ils rendent ces applications disponibles sous forme de services. C'est la combinaison du MOM et des conteneurs de services qui fournit à l'architecture ESB son caractère décentralisé. Enfin, une architecture ESB comprend un centre de gestion auquel le MOM et les conteneurs de services sont connectés ce qui lui permet de fournir une gestion centralisée de ceux-ci. Nous décrirons en détail l'architecture ESB dans le chapitre 2.

Exemple L'application illustrée sur la figure 1.17 de l'architecture ESB à notre cas compagnon donne un aperçu des capacités de celle-ci. Elle est capable, via les adaptateurs, de communiquer avec les systèmes intégrés selon de nombreux scénarios et technologies. Par exemple, une admission de patient dans le PMS produira un fichier HL7 contenant les données de cette admission et qui sera stocké sur un système de fichiers. L'ESB sera capable de scruter ce système de fichiers afin de prendre en charge ce fichier HL7, de router et transformer les données pour finalement les communiquer à chaque système selon la technologie qu'il supporte.

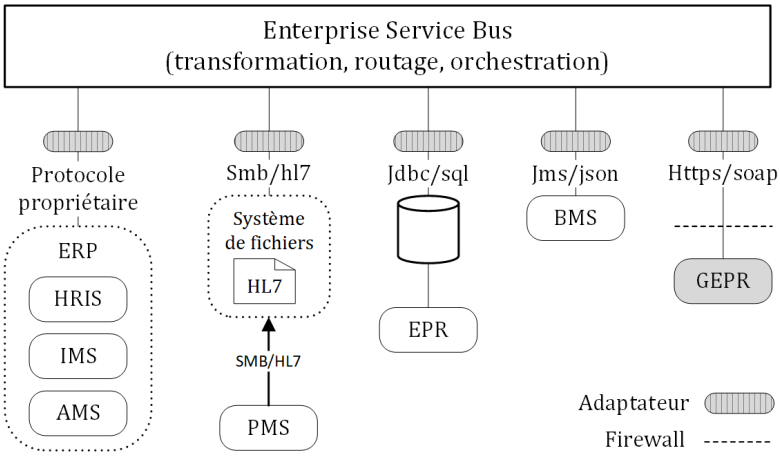


FIGURE 1.17 – Une intégration ESB de notre cas compagnon

Découplage :

Spatial	oui	Implémentation	oui
Temporel	1	Infrastructure	non
Synchronisation	2		

1. Dépend du style d'échange (Request/reply, fire and forget, etc.).
2. Dépend de la nature des échanges (Synchrone ou asynchrone).

Avantages et inconvénients :

+ Le nombre d'interfaces est réduit au minimum.

+ La logique d'intégration se situe au niveau de l'ESB et pas au niveau des applications.
+ De même que la gestion globale de la solution d'intégration.
+ Découplage des applications.
+ La décentralisation améliore la robustesse.
+ La décentralisation permet une répartition de la charge.
+ L'utilisation des standards permet une interopérabilité accrue.
+ L'utilisation des protocoles HTTP/HTTPS dont ils font massivement usage est connue et maitrisée. Ils peuvent donc facilement être utilisés à travers un firewall.
– Les tests et le débuggage sont plus compliqués.

1.5 Résumé

L'intégration est un domaine qui couvre tous les aspects d'une organisation. Stohr et Nickerson définissent un cadre de l'intégration. Celle-ci couvre aussi bien les aspects organisationnels que les aspects systèmes (au sens systèmes informatiques). Les aspects systèmes sont divisés en 3 couches : les données, les applications et les processus métiers. L'intégration d'applications concerne la couche applications et consiste à instaurer une collaboration entre les systèmes utilisés au sein de et entre les organisations. C'est un domaine complexe qui occupe une part importante des ressources informatiques d'une organisation. 30% à 35% des coûts de développement y sont liés [65].

La mise en place de solutions d'intégration d'applications nécessite la prise en compte de nombreux critères et implique de nombreuses conséquences. Ces solutions peuvent prendre la forme de plusieurs styles :

1. File transfer.
2. Shared database.
3. RPC.
4. Method wharehousing.
5. L'intégration d'interfaces utilisateur
6. ERP.
7. ETL.
8. Messaging.

et de plusieurs architectures :

1. L'architecture point à point.
2. L'architecture hub'n spoke.
3. L'architecture SOA.
4. L'architecture ESB.

Chapitre 2

Enterprise Service Bus

Dans le chapitre précédent, nous avons défini la notion d'intégration d'applications et avons exposé son importance dans la conception du système d'informations d'une organisation. Nous avons ensuite décrit les différents niveaux, méthodes et architectures d'intégration en mettant en évidence leurs avantages et inconvénients. Nous allons maintenant approfondir l'architecture du style ESB. Pour ce faire, nous commencerons par définir ce qu'est un ESB. Nous décrirons ensuite ses différents composants et leurs rôles à la suite de quoi, nous exposerons ses grandes caractéristiques. Nous terminerons en répondant à la question « Quand utiliser un ESB ? ».

2.1 Qu'est-ce qu'un ESB ?

2.1.1 Définir un ESB

Le terme ESB a été utilisé pour la première fois en 2002 par Roy Schulte, un analyste de Gartner Inc [11]. Il décrivait une catégorie de logiciels émergeant alors sur le marché :

> L'enterprise service bus (ESB), une nouvelle forme d'infrastucture combinant un middleware orienté message, des web services, une intelligence de transformation et de routage, sera exploitée dans la majorité des entreprises d'ici 2005

Ce nouveau type de logiciel répondait alors au besoin d'une nouvelle solution d'intégration comme nous l'avons vu dans le chapitre 1. Dix-sept ans plus tard, il n'existe toujours aucun consensus sur une définition de ce qu'est ou représente un ESB [46, 35]. Voici en effet quelques définitions tirées de la littérature sur le sujet.

Kress et al. en 2013 [35] répertorient trois définitions généralistes :

> Un style d'architecture d'intégration qui permet la communication via un bus de communication commun consistant en une variété de connexions point à point entre fournisseurs et clients de services.
>
> Une infrastructure qu'une entreprise utilise pour intégrer des services dans le paysage applicatif.

Un modèle d'architecture orientée service qui permet l'interopérabilité entre des environnements hétérogènes.

Menge en [46] présente un ESB comme une infrastructure :

Un ESB est une infrastructure distribuée d'intégration basée sur des standards ouverts et orientée message. Elle fournit des services de routage, d'invocation et de médiation pour faciliter l'interaction sûre et fiable d'applications et de services disparates et distribués.

Bhadoria, Chaudhari et Tomar en 2017 [8] décrivent un ESB comme un framework :

Un ESB est un framework logiciel qui gère les services et leur intégration avec d'autres services sur une plate-forme commune. Il fournit également l'infrastructure nécessaire à la mise en œuvre du routage des messages, de la traduction des protocoles et de la transformation des messages. Il fédère également les services du domaine applicatif existant et assure un couplage lâche entre eux. Avec une ESB, on peut concevoir, développer, déployer et surveiller des services au moment de l'exécution.

IBM Corporation en 2009 [29] voit un ESB avant tout comme un modèle d'architecture d'intégration :

Nous décrivons le bus de service d'entreprise d'abord et avant tout comme un modèle architectural. Il est possible de construire des ESB à partir d'une variété de technologies d'intégration sous-jacentes. Le modèle d'architecture reste valide et constitue un principe directeur pour permettre l'intégration et la fédération d'instanciations d'ESB multiples.

Un ESB permet l'intégration basée sur des normes entre des applications et des services lâchement couplés à l'intérieur et à l'extérieur de :

- Architectures orientées services - les applications distribuées sont composées de services réutilisables granulaires avec des interfaces bien définies, publiées et conformes aux normes.
- Architectures basées sur les messages - les applications envoient des messages aux applications par l'intermédiaire de l'ESB
- Architectures événementielles - les applications génèrent et consomment des messages de manière anonyme.

Un ESB permet l'intégration d'applications sur différentes plates-formes, modèles de programmation et normes de messagerie.

Woolley en 2006 [80] décrit un ESB comme un composant fondateur d'une SOA :

L'Enterprise Service Bus (ESB) est le composant de base d'une architecture orientée services (SOA) efficace. Un ESB fournit des services sécurisés d'interopérabilité et de transport de messages entre applications utilisant une variété de services Web et de technologies connexes. Il en résulte un ensemble de services aux entreprises plus ou moins couplés et interopérables qui peuvent être développés une seule fois et facilement partagés au sein de l'entreprise publique.

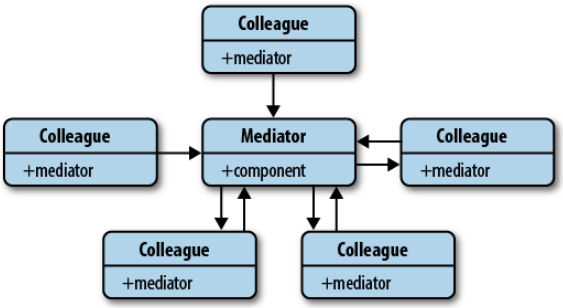


Figure 2.1 – Le design pattern Mediator.

La lecture de ces différentes définitions met en évidence le flou existant autour des ESB qui sont tour à tour catégorisés de framework logiciel, de modèle architectural ou encore d'infrastructure. Quoi que soit un ESB, ses fonctions semblent par contre assez unanimement reconnues. La première est d'établir une communication faiblement couplée entre applications différentes à travers divers mécanismes. La seconde est de supporter une architecture orientée service.

Ce travail n'a pas pour objectif d'établir une définition universelle du terme ESB. Nous pouvons cependant fournir notre propre définition.

> Un ESB est un modèle de conception architecturale fournissant une couche d'abstraction d'une infrastructure logicielle de type SOA. Par ses capacités d'abstraction, une architecture ESB permet d'établir une connectivité plus ou moins faiblement couplée entre les systèmes hétérogènes qui composent l'infrastructure logicielle. Ses trois composants principaux (MOM, conteneur de service et centre de gestion) prennent pour cela en charge toutes les tâches de médiation entre les systèmes publiés sur le bus :
>
> — connectivité des systèmes à l'ESB ;
>
> — orchestration des communications entre systèmes par :
>
> — la traduction des protocoles ;
>
> — la transformation des données ;
>
> — le routage des communications.

Dans un sens on pourrait dire que l'architecture ESB est à la conception d'une infrastructure logicielle ce que le design pattern *Mediator* est à la conception d'une architecture d'application. Le *Mediator*, illustré en figure 2.1 est un design pattern comportemental permettant d'exposer une interface unifiée à travers laquelle les différentes parties d'un système peuvent communiquer. Le modèle *Mediator* favorise le couplage lâche en garantissant qu'au lieu de composants (les *Colleague*) se référant les uns aux autres de manière explicite, leurs interactions sont gérées via un point central (le *Mediator*) dissociant les systèmes et améliorant le potentiel de réutilisation des composants. Appliqué à l'architecture ESB le médiateur est l'ESB tandis que les collègues sont les systèmes à intégrer. Ces derniers n'ont aucune conscience de leur existence mutuelle et ne communiquent que par l'intermédiaire de l'ESB.

2.2 Caractéristiques d'un ESB

Nous avons maintenant une définition générale d'un ESB. Explorons un peu plus en profondeur ses caractéristiques. Il est possible de trouver dans la littérature comme dans [80] et [13] des descriptions exhaustives de celles-ci. Cette section les compile en 6 caractéristiques.

Une architecture événementielle et orientée service. Dans un ESB, les services sont traités sous forme de points de connexion abstraits répondant à des évènements. Cette abstraction des services permet à ces derniers de ne pas avoir à se soucier des détails des communications sous-jacentes. Ils répondent simplement à des évènements que l'ESB leurs soumet ou envoient des évènements sur l'ESB sans conscience du ou des services qui traiteront ceux-ci. L'aspect événementiel des ESB permet également une transmission rapide des données contrairement à des solutions de traitements en lots planifiés à heures fixes. La latence des données est donc réduite.

Une architecture basée sur des standards. C'est un aspect essentiel des ESB. Ils utilisent ces standards pour la connectivité ESB-ESB et systèmes-ESB. Ils sont capables de s'interfacer avec des systèmes écrits dans différents langages comme Java, .Net ou C/C++ et prennent parfaitement en charge les web services comme SOAP et REST. Enfin, XML est utilisé comme type natif de représentation des données en transit dans un ESB.

Une architecture répartie Un ESB est capable de s'étendre sur l'ensemble de l'entreprise, quelque soit sa taille. Il est également capable de communiquer au delà des frontières de l'entreprise. Bien que dispersé dans toute l'entreprise, un ESB fournit les outils qui permettent de le configurer et le gérer de manière centralisée.

Une architecture distribuée et modulaire Elle permet de construire une topologie sans SPOF et d'ajouter des composants pour fournir de nouveaux services ou s'adapter aux besoins fonctionnels et non fonctionnels.

Une architecture sûre et fiable. Un ESB prend en charge les mécanismes d'authentification et d'autorisation. Une communication entre deux nœuds ESB est possible même en présence d'un firewall. Enfin le MOM est capable d'assurer la fiabilité de la transmission des messages.

2.3 Le style architectural ESB

Maintenant que nous savons ce qu'est un ESB et quelles sont ses caractéristiques, plongeons nous un peu plus en détail sur son architecture. Elle est constituée de 3 composants fondamentaux [11, 86]. Le premier d'entre eux est le MOM. Il constitue l'épine dorsale de l'ESB en assurant la transmission des messages. Les conteneurs de service quant à eux ont pour rôle d'assurer la connexion des fournisseurs et clients de service sur le MOM en transformant les requêtes qu'ils émettent

en message et les messages qu'ils reçoivent en requêtes qu'ils sont capables d'interpréter. Enfin, bien que ces composants forment une infrastructure distribuée, un ESB doit permettre une gestion centralisée de ceux-ci. C'est le rôle du centre de gestion qui est le 3ème et dernier composant essentiel d'un ESB. La figure 2.2 illustre ces composants ainsi que les liens qui les unissent. Le centre de gestion communique avec le MOM et tous les conteneurs de services. Ces derniers communiquent également avec le MOM.

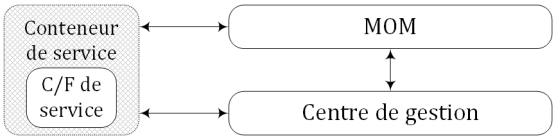


FIGURE 2.2 – Les 3 composants fondamentaux d'un ESB et les relations qui les unissent.

2.3.1 Le MOM

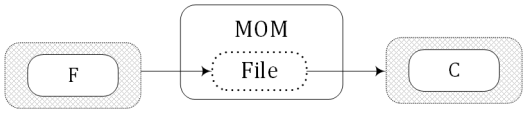
Le MOM constitue l'épine dorsale d'un ESB. Il est constitué :

— d'au moins un serveur pour les petites implémentations jusqu'à un nombre indéfini de serveurs pour les implémentations plus conséquentes.

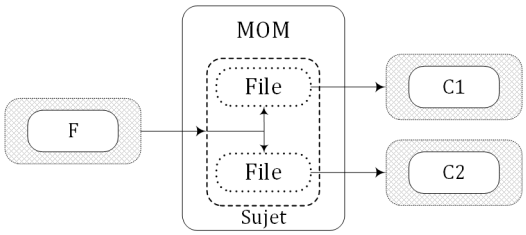
— de clients de messagerie qui transforment les requêtes des applications en message transmis sur le MOM et les messages reçus du MOM en invocation de service des applications.

Le MOM assure l'échange fiable, sécurisé et asynchrone de messages grâce à un mécanisme de type store and forward. Lorsqu'un serveur reçoit un message, il le stocke et envoie un accusé de réception au serveur destinateur. Il tente ensuite de l'envoyer au destinataire suivant jusqu'à ce que l'envoi réussisse et qu'il ait reçu l'accusé de réception du destinataire. Ce n'est qu'une fois ces conditions remplies et donc sa mission achevée qu'il supprime le message.

Les canaux de communication virtuels Le MOM établit des canaux de communication virtuels entre clients et fournisseurs de services. Ceux-ci remplacent les traditionnels appels synchrones des applications par un échange asynchrone de messages qui permet un découplage des systèmes. Ceux-ci ne doivent plus attendre une réponse des systèmes avec lesquelles ils communiquent et n'ont même plus besoin de les connaître. En effet, un MOM est capable de gérer un ensemble de files d'attente (queue) et de sujets lui permettant d'établir deux types de canaux de communication de base. Dans le premier, le type point à point (figure 2.3a), un fournisseur émet un message sur une file gérée par le MOM. Un consommateur est connectée à cette file et réceptionne le message. Dans le second, le type publish and subscribe (figure 2.3b), un fournisseur envoie des messages à un sujet. Dès qu'un consommateur s'abonne à ce sujet, le MOM crée une file dédicacée à celui-ci au sein du sujet. Un message envoyé à un sujet est donc dupliqué en autant d'exemplaires qu'il y a de consommateurs abonnées au sujet.



(a) Canal de communication point à point



(b) Canal de communication publish and subscribe

FIGURE 2.3 – Deux types de communication de base dans un ESB

Les messages Les messages sont le moyen de communiquer des données dans un ESB. Pour ce faire, ils contiennent différentes sections que sont l'entête qui contient des informations d'identification et de routage, les propriétés qui permettent aux applications de fournir des informations qui leur sont propres et enfin le corps qui contient les données du message [11]. La plupart des ESB utilisent le format d'échange SOAP. Celui-ci est standardisé et contient les sections décrites ci-dessus ce qui le rend particulièrement adapté à cette tâche. Les données transmises dans le corps du message le sont souvent sous le format XML qui dispose de nombreux outils de manipulation comme XSLT, XQuery ou encore XPath.

2.3.2 Les conteneurs de services

Dans un ESB, tous les services et applications qui y sont connectés sont vus comme des endpoints abstraits qui peuvent représenter divers types d'artefacts tels que :

- un service externe auquel l'entreprise doit faire appel ;
- un service interne spécialisé e.g. le calcul de la facturation de l'hospitalisation d'un patient ;
- un service interne de l'ESB comme un service de transformation XML ou d'enregistrement d'informations sur le fonctionnement de l'ESB ;
- une application de l'entreprise ;
- un îlot d'intégration réalisé à l'aide d'une autre technologie.

Ces endpoints d'un point de vue architectural sont des abstractions logiques de services connectés à l'ESB. Les conteneurs de service sont la manifestation physique, sous forme de processus distant, de ces endpoints abstraits. Ils fournissent l'implémentation de l'interface du service [13]. Ils produisent une grande partie des fonctionnalités d'un ESB et c'est à leur niveau que toute la logique d'intégration est hébergée. Ils représentent donc en un sens, bien plus l'ESB que la couche de communication que constitue le MOM. Concrètement, un conteneur de service est constitué de 3 composants qui sont illustrés sur la figure 2.4.

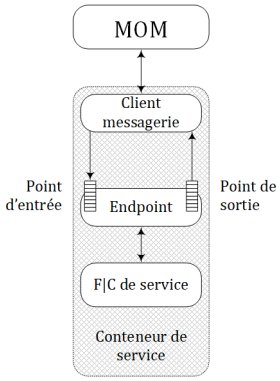


FIGURE 2.4 – Un conteneur de service est constitué d'un service ou d'une application, d'un ou plusieurs endpoints et d'un client de messagerie qui communique avec le MOM.

Le client de messagerie Dans la section précédente, nous avons vu que le MOM était constitué de serveurs et de clients de messagerie. Ces clients de messagerie sont un des composants des conteneurs de service. Ils permettent de connecter ces derniers aux serveurs MOM. Ils communiquent au endpoint les messages entrants et envoie sur le MOM les messages sortants.

Les endpoints Ils sont les intermédiaires entre les applications ou les services gérés par le conteneur et le client de messagerie. Pour ce faire, il expose un interface standardisé constitué d'un point d'entrée et d'un point de sortie, chacun d'eux gérant une file de message. Le client de messagerie place chaque message entrant dans la file du point d'entrée et traite les messages de la file du point de sortie. L'arrivée d'un nouveau message déclenchera son traitement qui pourra consister en diverses opérations comme transformer le message XML en objet, enrichir le message par l'appel d'autres services, ou encore invoquer des méthodes de l'application associée au conteneur de services.

Le service ou l'application L'utilité d'un conteneur de service est de connecter une application ou un service sur le bus. La nature de l'application ou du service pourra être variée. Il pourra s'agir d'un web service externe SOAP ou REST comme le service Recip-e illustré sur la figure 2.4, d'une application exposant ses fonctionnalités sous forme de RPC ou de services web, d'une ancienne application qui nécessitera l'utilisation ou la création d'un adaptateur, d'un client mail ou encore d'un client FTP. Les possibilités sont nombreuses.

2.3.3 Centre de gestion

Une architecture de type ESB possède un caractère fortement distribué et décentralisé qui lui est donné par les serveurs MOM et les conteneurs de services qui la constituent. Ceux-ci nécessitent d'être configurés, gérés et surveillés. Afin d'assurer cette gestion tout en conservant son caractère, un ESB possède un centre de gestion qui comprend un répertoire central, un ensemble de serveurs de gestion, des

interfaces de gestion au niveau des serveurs MOM et des conteneurs de service et enfin divers outils de configuration, de gestion et de surveillance.

2.3.4 Vue complète de l'architecture ESB

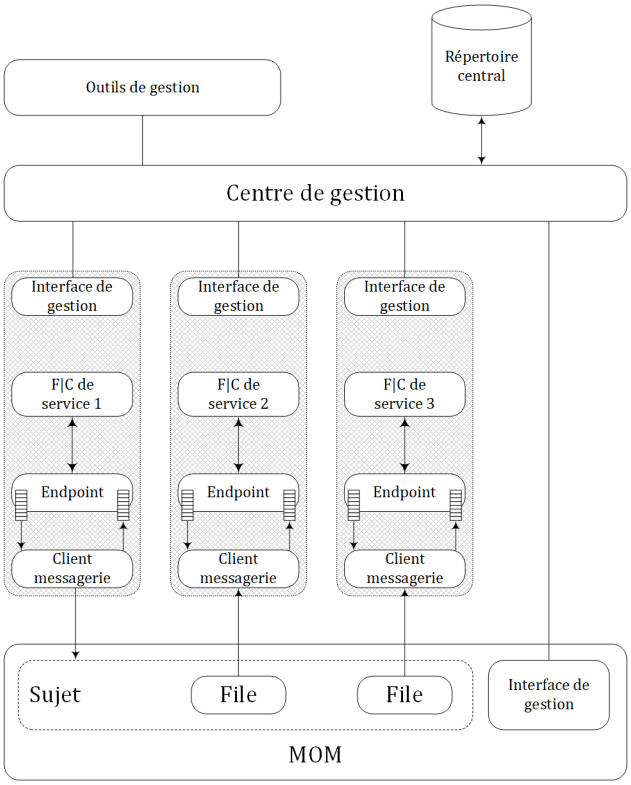


FIGURE 2.5 – Vue global de l'architecture d'un ESB.

La figure 2.5 montre une illustration complète de l'architecture d'un ESB. Chaque fournisseur ou client de services est encapsulé dans un conteneur de services. Ce dernier communique avec le MOM grâce à son client de messagerie. Un endpoint joue le rôle d'intermédiaire entre ce client de messagerie et le fournisseur/client de service. D'un part il permet au fournisseur de service de recevoir des requêtes via sa file d'entrée. D'autre part, il permet au client d'en envoyer via sa file de sortie. Le MOM gère des sujets et des files que le client de messagerie peut alimenter ou consommer. Conteneurs de services et MOM sont tous deux pourvus d'une interface de gestion reliée au centre de gestion. Ce dernier possède des outils de gestion et un répertoire central stockant toutes sortes d'artefacts de l'ESB. Une communication de type publish and subscribe est ici représentée entre le fournisseur 1 qui envoie des messages sur le sujet A auquel sont abonnés les clients 2 et 3.

2.4 Quand utiliser un ESB ?

Le début de ce chapitre a été consacré à la compréhension de l'architecture ESB. Nous allons à présent tenter d'identifier quand l'adoption d'un ESB est opportune. Pour cela nous commencerons par extraire de la littérature des indicateurs d'usage. Nous définissons un indicateur d'usage comme une situation ou un scénario problématique qu'un ESB peut aider à solutionner. Nous présenterons ensuite un modèle des coûts qui nous permettra d'évaluer les aspects financiers d'une architecture ESB. Nous finirons par relever les bénéfices qu'un ESB peut apporter.

2.4.1 Signaux d'usage

Cette section décrit les signaux qui s'ils sont rencontrés, devraient alerter le gestionnaire et/ou l'architecte que l'usage d'un ESB peut être envisagé. Plus ils seront nombreux, plus l'usage d'un ESB sera adapté. La liste présentée ci-dessous a été établie et compilée à partir de la littérature étudiée.

— Si trois ou plus de trois applications doivent être intégrées. En effet, il existe des ESB dit légers dont la plupart sont open-source et dont la mise en place peut déjà s'envisager dans des scénarios impliquant peu d'applications [35, 56].

— Le système d'information est en constante évolution. Des applications internes ou externes sont remplacées et/ou ajoutées régulièrement [3].

— Il est nécessaire de réduire la complexité du développement et de l'intégration de vos applications [80].

— L'évolution des besoins métiers de l'entreprise nécessite une adaptation rapide et efficiente du système d'information. [80].

— Des services de fournisseurs externes doivent être utilisés. L'impact des modifications effectuées sur ces services est alors localisé au niveau du conteneur de service. L'interface de celui-ci reste stable pour les applications clientes, l'ESB prenant en charge les modifications nécessaires [35].

— De nombreux protocoles comme HTTP, SOAP, REST, FTP et d'autres doivent être utilisés. L'ESB prend en charge leurs transformations [35, 3, 56].

— De nombreux styles d'interaction sont nécessaires [29].

— Il est nécessaire d'accéder à un ensemble de composants et d'adaptateurs prédéfinis, ce qui permet d'intégrer de manière standardisée différents protocoles et applications externes [35].

— Les communications entre applications et services doivent être traitées de manière fiable et sécurisée [35].

— Il est nécessaire d'orchestrer les communications entre les applications ou de construire des applications composites comme un portail présentant des informations provenant de plusieurs applications [3].

— Les données sont de petites tailles et doivent être disponibles avec une faible latence [3].

— La solution d'intégration doit pouvoir être adoptée selon un processus itératif et incrémental [3].

2.4.2 Coûts d'une architecture ESB

Du point de vue du gestionnaire, les aspects économiques liés au choix d'une architecture d'intégration sont un critère important. Dans [32], KOKORCENY propose un modèle comparatif théorique des coûts de l'implémentation des architectures point à point et ESB. Dans son modèle, l'auteur considère que dans le cas de l'architecture ESB, un système à intégrer ne nécessite qu'une interface quel que soit le nombre de systèmes avec lesquels il doit communiquer. A notre sens ce postulat est faux. En effet, bien qu'un système intégré ne communique qu'avec l'ESB, une chaîne de médiation spécifique devra bel et bien être mise en place, au sein de l'ESB, entre ce système et chacun des autres systèmes avec qui une communication doit avoir lieu. Affirmer qu'un système connecté à un ESB ne nécessite qu'une interface est donc partiellement faux.

Cependant, l'auteur met en avant quelques facteurs qu'il est intéressant d'aborder pour évaluer le coût de la mise en place d'une architecture ESB par rapport à celui d'une architecture point à point.

Coût de la mise en place Le coût de la mise en place d'une architecture d'intégration peut être divisé en deux parties :

1. Le coût de la mise en place de la solution.
2. Le coût d'implémentation de chaque interface.

En ce qui concerne la mise en place, l'architecture point à point est moins coûteuse. Elle nécessite peu ou pas d'infrastructure spécifique contrairement à l'ESB qui imposera le déploiement de serveurs dédiés. Pour ce qui est du coût d'implémentation de chaque interface, l'ESB possède un avantage en fournissant une gamme d'outils et de fonctionnalités accélérant et facilitant leur création.

Coût de l'ajout d'un nouveau système Dans le même ordre d'idées, le coût de l'ajout a posteriori d'un nouveau système est moins coûteux dans le cas de l'ESB pour les raisons que nous venons d'évoquer.

Degré d'intrégration L'auteur dans son article souligne le fait que la littérature traitant du sujet suppose très souvent que chaque système d'un système d'information doit communiquer avec chacun des autres systèmes qui le composent. Il indique que cette affirmation est fausse et se base pour cela sur son expérience dans l'analyse et la mise en place de nombreux projets d'intégration. Il définit le degré d'intégration comme le rapport entre le nombre effectif d'interfaces à développer et le nombre maximal théorique de ces interfaces.

En conclusion, on observe que l'avantage financier induit par la mise en place d'une architecture ESB sera d'autant plus grand que le nombre de systèmes à intégrer est élevé, que le nombre planifié de nouveaux systèmes est important et que le degré d'intégration est élevé.

2.4.3 Bénéfices d'un ESB

Le modèle des coûts de l'architecture ESB et les signaux d'usage constituent une base de décision pour les gestionnaires et les architectes système quant à l'opportunité d'adoption d'un ESB. Dans le chapitre 3 nous proposerons un modèle des fonctionnalités permettant de comparer et de choisir un ESB. Quel qu'il soit, un ESB apportera les bénéfices suivants :

- Il constitue un support à l'élaboration d'une architecture SOA [80].
- Une connectivité universelle entre les applications et les services grâce à l'échange de messages souvent au format XML [12, 80].
- L'utilisation de standards de communication qui rendent l'entreprise indépendante des vendeurs de solutions d'intégration[12].
- Des qualités de service comme la fiabilité de transmission des messages, le support des transactions ou encore une architecture adaptable aux besoins non fonctionnels [12].
- Des fonctionnalités de médiation assurent un faible couplage entre les applications et les services [12].
- Une meilleure flexibilité du système d'information permettant de répondre plus rapidement aux besoins métiers [56, 80].
- Une réduction du TCO global du parc logiciel de l'entreprise essentiellement grâce à une économie sur les ressources nécessaires à la gestion et à la maintenance de l'intégration [56, 80, 32].

2.5 Résumé

Nous définissons un ESB :

> Un ESB est un modèle de conception architecturale fournissant une couche d'abstraction d'une infrastructure logicielle de type SOA. Par ses capacités d'abstraction, une architecture ESB permet d'établir une connectivité plus ou moins faiblement couplée entre les systèmes hétérogènes qui composent l'infrastructure logicielle. Ses trois composants principaux (MOM, conteneur de service et centre de gestion) prennent pour cela en charge toutes les tâches de médiation entre les systèmes publiés sur le bus :
>
> - connectivité des systèmes à l'ESB ;
> - orchestration des communications entre systèmes par :
>   - la traduction des protocoles ;
>   - la transformation des données ;
>   - le routage des communications.

Une architecture ESB est :

- événementielle et orientée service ;
- basée sur des standards ;
- répartie ;

— distribuée et modulaire ;

— sûre et fiable.

Un ESB est composé de 3 éléments fondamentaux :

— le MOM ;

— les conteneurs de services ;

— le centre de gestion.

De nombreux signaux peuvent être utilisés pour identifier le besoin d'un ESB au sein d'une entreprise. L'économie potentielle du passage d'une architecture point à point à une architecture ESB peut être évaluée grâce au modèle proposé par Kokorceny. Enfin les bénéfices de l'architecture ESB sont identifiés et explicités.

Chapitre 3

Modèle des capacités des ESB

Dans la première partie de ce travail, nous avons introduit la notion d'intégration des systèmes d'information et avons approfondi l'intégration d'applications à travers les niveaux, les styles et les architectures d'intégration. Nous nous sommes ensuite attardé sur l'architecture ESB en décrivant ses différents composants et leurs rôles respectifs, ses caractéristiques et avons fourni des pistes permettant de répondre à la question « Quand utiliser un ESB ? ». Ce chapitre sera consacré à l'élaboration d'un modèle des capacités des ESB. Il sera ensuite opérationnalisé dans le chapitre 4 sous la forme d'une méthode d'évaluation des ESB. Elle permettra d'identifier parmi un ensemble de plusieurs alternatives, l'ESB le plus adapté à un contexte donné.

3.1 Méthodologie

L'établissement de notre modèle des capacités des ESB a suivi quatre étapes. Premièrement, une recherche de littérature basée sur notre question de recherche a été effectuée sur trois moteurs de recherche. Deuxièmement, l'ensemble des publications résultant de notre recherche a été filtré avec pour objectif de ne conserver que les documents dignes d'intérêt. Troisièmement, un formalisme de notre modèle a été choisi. Quatrièmement, le corpus de publications final a été analysé et a débouché sur l'élaboration de notre modèle des capacités des ESB.

3.2 Recherche et filtrage de littérature

3.2.1 Édification de la recherche

Cette première étape de collecte de littérature a été réalisée sur 3 moteurs de recherche : ACM, IEEE Xplore et Springer Link. La construction de notre requête est basée sur notre première question de recherche : « Quel pourrait être un modèle des capacités des ESB ? ». Nous y avons identifié 3 concepts. Il s'agit des concepts de modèle, de capacité et enfin d'ESB. Pour chacun d'eux, une liste de synonymes ou de termes sémantiquement proches a été composée. La table 3.1 présente pour

Table 3.1 – Les concepts de notre question de recherche et leurs termes proches

Concepts		Termes possibles
ESB	x	Enterprise Service Bus
		Enterprise SOA
		Enterprise Systems Connection
	x	ESB
Fonctionnalité		Ability
	x	Capability
		Capacity
		Competence
	x	Feature
	x	Functionnality
		Preoccupation
	x	Requirement
		Skill
Modèle		Diagram
	x	Map
	x	Mapping
	x	Model
		Picture
		Plan
	x	Summary
	x	Survey
	x	Taxonomy

chaque concept, les termes qui y sont associés et ceux qui ont été retenus pour la construction de notre requête de recherche.

Les termes non retenus l'ont été pour les raisons suivantes :

— « Enterprise SOA » et « Enterprise Systems Connection » sont à la fois trop généraux et trop éloignés du concept d'ESB.

— « Capacity » est lié à une notion de performance qui ne se rapporte pas à l'objet de ce travail.

— « Ability », « Competence », « Preoccupation », « Skill » ont une sémantique trop éloignées de la notion de fonctionnalité ou sont peu usités dans le domaine de l'informatique.

— « Diagram », « Picture », « Plan » sont des termes trop généraux ou dont la sémantique est trop éloignées du concept de modèle.

Les requêtes utilisées pour chaque moteur de recherche ainsi que les filtrages appliqués en supplément sont indiqués dans le tableau 3.2. Globalement seuls les articles publiés à partir de 2002 ont été retenus. Il s'agit en effet de l'année ou le concept d'ESB est apparu pour la première fois [11]. La collecte de la littérature dans les trois moteurs de recherche utilisés a permis de répertorier un total de 231 publications.

3.2.2 Filtrage des résultats

Les 231 publications obtenues lors de nos recherches ne constituant qu'un ensemble de littératures brut et inutilisable, un filtrage a été appliqué avec pour objectif

TABLE 3.2 – Requêtes de collecte de publications par moteur de recherche

Librairie	Requête de recherche
IEEE Xplore	((esb OR "enterprise service bus*") AND (functionality OR functionalities OR capability OR capabilities OR requirement OR requirements OR feature OR features) AND (model OR map OR mapping OR survey OR summary OR taxonomy)) Uniquement les publications de type « article de conférence » et « article de journaux » publiées à partir de 2002
ACMDL	((esb OR ""enterprise service bus*"") AND (functionality OR functionalities OR capability OR capabilities OR requirement OR requirements OR feature OR features) AND (model OR map OR mapping OR survey OR summary OR taxonomy)) Uniquement les publications publiées à partir de 2002 en y incluant celles du « Guide ACM pour la littérature scientifique ».
Springer Link	((esb OR "enterprise service bus*") AND (functionality OR functionalities OR capability OR capabilities OR requirement OR requirements OR feature OR features) NEAR (model OR map OR mapping OR survey OR summary OR taxonomy)) Uniquement les publications de la section « Sciences Informatiques » de type « Article » et « Article de conférence » publiées à partir de 2002

de rendre cet ensemble exploitable dans le cadre de notre recherche en excluant : les publications en doublons ; les publications hors sujet ; les publications inutilisables pour établir notre modèle. Ce travail de dégrossissement a été conduit en suivant les 5 étapes décrites ci-après. La table 3.3 résume ce processus de filtrage et indique pour chaque étape le nombre de publications retirées ou ajoutées avec entre parenthèses le nombre résultant de publications.

Étape 1 (ED) : élimination des doublons Le résultat de chaque moteur de recherche a été exporté en CSV et chargé dans Microsoft Excel. La fonction d'identification des doublons de l'application a permis d'identifier 4 doublons entre des publications de ACM et Springer et 18 entre ACM et IEEE Xplore. Aucun doublon entre Springer et IEEE Xplore n'était présent. Le champ DOI [1] a servi de clé d'identification des doublons.

Étape 2 (T1) : Tri sur base du titre et de l'abstract A partir de la liste résultante de l'étape 1, un premier filtrage des publications se basant sur le contenu des champs « Titre » et « Abstract » a été effectué avec pour critères d'exclusion :

— Les publications qui ne concernent ni l'intégration d'applications, ni les ESB, ni l'architecture SOA.

— Les publications qui citent les ESB comme une option parmi d'autres à une problématique exposée.

Tous les articles qui ne rencontrent aucun de ces critères d'exclusion ont été retenus pour la seconde étape de sélection. Si la lecture du titre ou du résumé de la

1. Digital Object Identfier : Méthode standardisée d'identification permanente d'un objet publié électroniquement.

Table 3.3 – Vue synthétique des étapes de filtrage du résultat de la recherche.

Librairie	Résulats	ED	T1	T2	SB	AL
IEEE Xplore	65	-0(65)	-33(32)	-29(3)	+4(15)	+3(18)
ACMDL	82	-22(60)	-23(37)	-33(4)		
Springer Link	84	-0(84)	-56(28)	-24(4)		

publication ne permettait pas d'établir avec certitude la présence d'un des critères d'exclusion, la publication concernée a également été retenue pour la seconde étape de sélection.

Étape 3 (T2) : Tri sur base d'une lecture complète Cette dernière étape de sélection s'est réalisée sur base d'une lecture complète des publications. Celles présentant un contenu intéressant pour la construction de notre modèle ont été retenues. Par contenu intéréssant, nous entendons un contenu suffisamment détaillé sur la globalité des capacités des ESB ou encore sur un aspect plus précis de celles-ci.

Par conséquent ont été éliminées, les publications présentant une description ou un contenu trop généraliste comme « Un ESB est une plateforme d'intégration qui combine messagerie, services, transformation de données, routage intelligent et fourni une connexion sûre et de la coordination entre des services divers [13] ».

Étape 4 (SB) : ajout de références à partir de celles des publications sélectionnées (snowballing) Le filtrage à travers les étapes 2 et 3, a réduit notre littérature à un ensemble de 11 publications. Une 4ème étape d'analyse des références a ensuite eu lieu. Elle a consisté à analyser les références des différentes publications retenues en fonction du contexte de citation de celles-ci. Celles où sont abordées les capacités des ESB ou des aspects particuliers de ces dernières ont été ciblées. Ceci a débouché sur l'ajout de 4 publications supplémentaires portant ainsi le total à 15 publications.

Étape 5 (AL) : ajout libre de référence Pour terminer, nous avons ajouté des publications découvertes lors des recherches menées dans le cadre de l'élaboration des deux premiers chapitres de ce travail. Ces publications, au nombre de 3, ont été ajoutées car elles comportent des informations pertinentes et utiles à l'élaboration de notre modèle.

La table 3.4 indique les références restantes après l'étape T2 et celles ajoutées lors des étapes SB et AL.

Table 3.4 – Références ajoutées à chaque étape

Étape	Références
T2	[8, 31, 41, 47, 48, 51, 54, 53, 62, 67, 83]
SB	[28, 52, 56, 79]
AL	[7, 35, 46]

3.3 Conception du modèle

3.3.1 Analyse du corpus de publication

La conception de notre modèle a en premier lieu été conduite par l'analyse approfondie du corpus de publications. Il est à noter le faible nombre d'articles qu'il contient. Parmi ceux-ci, aucun ne propose un modèle complet des capacités des ESB. La table 3.5 présente l'ensemble des publications de notre corpus et pour chacune d'elle son apport à la conception de notre modèle. L'analyse a consisté en une lecture approfondie des différents articles. Pour chacun d'eux, nous avons répertorié les catégories de capacités et capacités qu'il proposait. L'ensemble a été représenté à l'aide de vignettes organisées par article, par catégorie et sous catégorie et enfin par capacité. Cette manière de faire nous a permis d'obtenir une vision globale du contenu des publications, vision qu'un écran d'ordinateur n'aurait pu nous offrir. A partir de l'analyse, une hiérarchie des capacités des ESB a été construite en suivant une approche top-down consistant à créer en premier lieu les catégories principales puis par raffinements successifs à définir l'arborescence complète sous chaque catégorie. Selon [57], cette approche est la plus appropriée à l'élaboration manuelle d'une taxonomie. En opposition, l'approche bottom-up consiste à démarrer par le bas de la taxonomie pour atteindre ensuite les catégories supérieures. L'approche bottom-up est plus souvent utilisée dans les techniques automatiques de génération de taxonomie.

3.3.2 Formalisme du modèle

Afin de représenter les capacités des ESB, il a été nécessaire de choisir un formalisme adapté répondant aux exigences suivantes :

— Il possède la capacité de représenter hiérarchiquement les différentes fonctionnalités. De cette façon, les liens entre les fonctionnalités englobantes et englobées sont visibles.

— Il possède la capacité d'effectuer une distinction entre les fonctionnalités de base et les fonctionnalités étendues (optionnelles) des ESB.

— Il possède la capacité de visualiser le modèle sous forme graphique et intuitive.

Notre choix s'est porté sur les diagrammes de fonctionnalités.

Modèle de fonctionnalités (Feature model) Les modèles de fonctionnalités sont des artefacts couramment utilisés pour décrire, en termes de fonctionnalités, un ensemble de produits d'une ligne de produits. Une ligne de produits désigne le développement d'un ensemble de produits qui partagent des fonctionnalités communes [24]. Nous avons choisi les modèles de fonctionnalités comme formalisme de notre modèle car il rencontre toutes les exigences exprimées. En effet, [5] nous indique qu'un modèle de fonctionnalités est un ensemble de fonctionnalités organisées hiérarchiquement dans lequel chaque fonctionnalité est (si sa fonctionnalité parente est sélectionnée) obligatoire ou optionnelle et dans lequel une relation entre une fonctionnalité parente et ses fonctionnalités enfantes doit être d'un des types suivant :

— Et (And) - Sous cette relation, toutes les fonctionnalités renseignées comme obligatoires seront sélectionnées tandis que celles renseignées comme optionnelles pourront l'être.

TABLE 3.5 – Composition du corpus de publications

BHADORIA, CHAUDHARI ET THARINDA NISHANTHA VIDANAGAMA [7]
Abordent principalement les aspects de qualité et de sécurité des ESB.
BHADORIA, CHAUDHARI ET TOMAR [8]
Proposent une vue global et catégorisée des capacités des ESB.
HÉRAULT ET THOMAS [28]
Proposent quelques modèles de médiation couramment utilisés dans les ESB.
KEEN ET AL. [31]
Proposent une catégorisation globale des capacités des ESB sans rentrer dans le détail de chaque catégorie. Distinguent les catégories indispensables des catégories optionnelles.
KRESS ET AL. [35]
Proposent une catégorisation globale des capacités des ESB sans rentrer dans le détail des catégories.
LLAMBÍAS, GONZÁLEZ ET RUGGIA [41]
Identifient les modèles d'intéractions entre clients et fournisseurs de service.
MENGE [46]
Propose un découpage intéressant des capacités des ESB en catégories mais restent très général dans la description de chacune d'elles.
MI ET AL. [47]
Abordent uniquement la notion de routage.
MORAND, GARCIA ET LALANDA [48]
Abordent surtout les capacités non fonctionnelles des ESB tel que la qualité des services.
MULIK [51]
Seul référence à identifier la fonctionnalités de BAM pour Business Activity Monitoring.
ORACLE [52]
Propose une vue des fonctionnalités du produit AquaLogic Service Bus classées par catégorie.
PAPAZOGLOU [53]
Propose un aperçu des fonctionnalités les plus communes des ESB en rapport avec les SOA.
PAPAZOGLOU ET VAN DEN HEUVEL [54]
Parcourent les fonctionnalités des ESB sans adopter de catégorisation particulière
RADEMAKERS ET DIRKSEN [56]
Parcourent de façon assez exhaustive les fonctionnalités principales des ESB et les illustrent par des exemples dans Mule ESB et Apache Service Mix.
SCHMIDT ET AL. [62]
Proposent quelques modèles de médiation couramment utilisés dans les ESB.
TANG ET AL. [67]
Répertorient exhaustivement les modèles d'interactions entre client et fournisseur de services ainsi que les attributs de qualité d'une SOA.
WOOLF ET HOHPE [79]
Traitent exclusivement de la partie MOM et donc messagerie des ESB. Proposent un catalogue complet des opérations possibles sur les messages.
WU ET CHANG [82]
Abordent les fonctionnalités de médiation des ESB et en particulier les aspects de transformation des données.

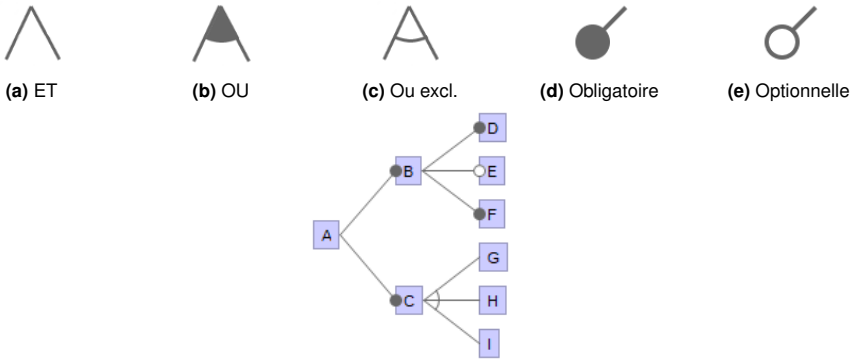
— Ou exclusif (Alternative) - Une seule fonctionnalité enfante doit être sélectionnée.

— Ou (Or) - Au moins une ou plusieurs fonctionnalités enfantes peuvent être sélectionnées.

De plus, les modèles de fonctionnalités peuvent être représentés graphiquement sous la forme de diagrammes de fonctionnalités.

Diagramme de fonctionnalités (Feature diagram) Un diagramme de fonctionnalités exprime graphiquement un modèle de fonctionnalités. Ce dernier est représenté sous la forme d'un arbre dans lequel les feuilles représentent les fonctionnalités pri-

mitives tandis que les nœuds intérieurs représentent les fonctionnalités composées [5].



(f) Exemple d'un diagramme de fonctionnalités

FIGURE 3.1 – Notation des diagrammes de fonctionnalités.

La notation du caractère obligatoire ou optionnel d'une fonctionnalité est illustré respectivement par les figure 3.1d et 3.1e. La notation des relations dans un diagramme de fonctionnalités est représentée sur les figure 3.1a pour le ET, 3.1b pour le OU et 3.1c pour le OU exclusif. La figure 3.1f montre un exemple de diagramme de fonctionnalités dans lequel une ligne de produit *A* contient deux fonctionnalités *B* et *C*. *B* est une fonctionnalité obligatoire composée de *D* et *F* et optionnellement de *E*. *C* est une fonctionnalité alternative composée soit de *G*, *H* ou *I*. La ligne de produit *A* présente une variabilité de 6 signifiant que 6 produits différents peuvent y être produits. Dans notre exemple, ces 6 produits pourront être notés par les ensembles suivants :

1. {*A*,*B*,*C*,*D*,*E*,*F*,*G*}
2. {*A*,*B*,*C*,*D*,*E*,*F*,*H*}
3. {*A*,*B*,*C*,*D*,*E*,*F*,*I*}
4. {*A*,*B*,*C*,*D*,*F*,*G*}
5. {*A*,*B*,*C*,*D*,*F*,*H*}
6. {*A*,*B*,*C*,*D*,*F*,*I*}

3.4 Modèle des capacités des ESB

Notre modèle des capacités des ESB sera donc représenté sous la forme d'un diagramme de fonctionnalités. Idéalement, un diagramme global devrait être présenté. Cependant, pour une raison de lisibilité due au format du présent document, ce diagramme global sera scindé en plusieurs diagrammes. Un premier diagramme montrera les catégories principales de fonctionnalités qui composent un ESB. Chaque catégorie sera ensuite détaillée dans un diagramme dédié. Le lecteur remarquera que certaines fonctionnalités sont postfixées par un « + ». Il indique que la liste des sous fonctionnalités correspondantes est non exhaustive et limitée aux fonctionnalités les plus communes. Il aurait été par exemple inutile de détailler tous

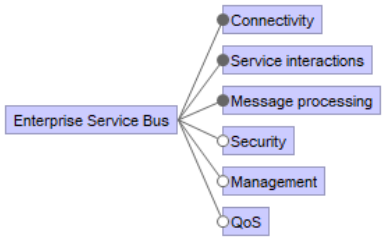


FIGURE 3.2 – Catégories principales de fonctionnalités des ESB.

les protocoles ou tous les formats de données existants. Lorsque le modèle sera opérationnalisé, il faudra, si nécessaire, y ajouter les fonctionnalités spécifiques au domaine d'application de l'ESB. Nous y reviendrons dans le chapitre suivant. L'outil que nous avons utilisé (FeatureIDE [2]) pour créer les diagrammes n'acceptant pas les caractères accentués, l'anglais a été utilisé au sein des divers diagrammes.

3.4.1 Catégories

La figure 3.2 montre les catégories principales de fonctionnalités des ESB. Celles-ci sont :

Connectivité : Les fonctionnalités de connectivité fournissent les mécanismes nécessaires afin de pouvoir connecter clients et fournisseurs de services sur le bus.

Interactions de services : Cette catégorie contient les capacité permettant aux clients et fournisseurs de services d'interagir une fois qu'ils sont connectés au bus.

Traitement des messages : Cette catégorie de fonctionnalités contient toutes les capacités dont l'ESB dispose pour traiter les messages.

Sécurité : Elle contient les options possibles de gestion de la sécurité entre clients et fournisseurs de services.

Gestion : Elle fournit les capacités de gestion de l'ESB.

Qualité de service : Elle contient les options de gestion de la qualité des services.

3.4.2 Connectivité

Un ESB facilite la mise en place d'une SOA. Il endosse pour cela un rôle de médiateur entre les services et libèrent ceux-ci de la responsabilité de s'adapter aux modèles de communication et de formats de données des autres services. Afin de remplir ce rôle de médiation, un ESB doit en premier lieu permettre aux clients et aux fournisseurs de services de s'y connecter. Les capacités nécessaires sont précisément fournies par cette catégorie et illustrées sur la figure 3.3.

2. https://featureide.github.io/

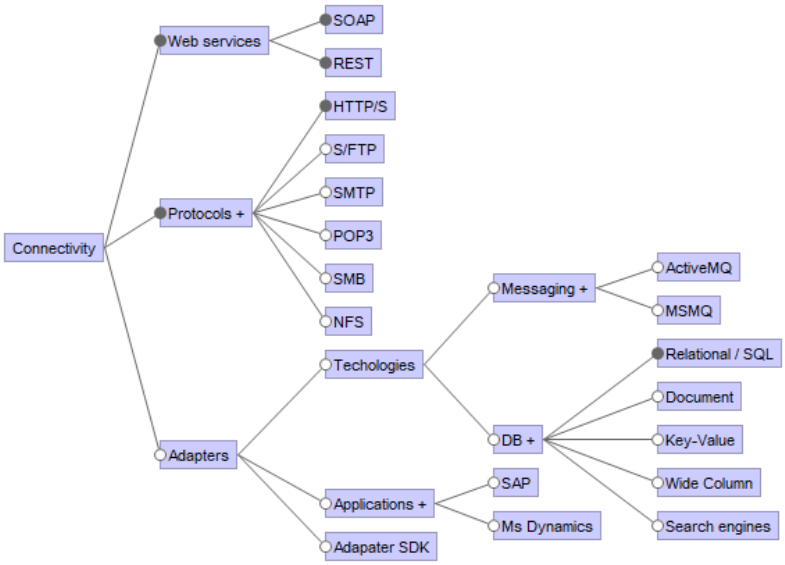


FIGURE 3.3 – Capacités de la catégorie « Connectivity ».

Web services

Les ESB étant fortement liés au concept de SOA, il est impératif que les web services puissent être connectés au bus. Actuellement et selon [69], REST occupe 83% de part de marché tandis que les web services SOAP n'en occupent plus que 15%. Toutefois, ces derniers sont intrinsèquement liés au format XML qui est lui-même le standard utilisé dans les ESB. En conséquence, un ESB doit prendre en charge ces deux types de web services.

Protocoles

Clients et fournisseurs de services peuvent utiliser des protocoles divers et variés pour se connecter à l'ESB. Ce dernier assurera la traduction d'un protocole à l'autre en fonction des besoins des systèmes connectés. Les protocoles présentés ici sont les plus couramment utilisés. Cette liste pourra être complétée en fonction du contexte d'utilisation de l'ESB.

HTTP/S : Protocole de communication entre un client et un serveur web fonctionnant sur le principe de requête et réponse. HTTPS est une combinaison de HTTP avec le protocole SSL/TLS. Il fournit une communication chiffrée et une identification sécurisée.

S/FTP : Protocole standardisé de transfert de fichiers entre ordinateurs sur un réseau d'ordinateurs. SFTP est une version sécurisée de FTP utilisant le protocole SSH.

SMTP : Protocole de communication utilisé pour le transfert de courriers électroniques vers un serveur de messagerie.

POP3 : Protocole permettant d'interroger et rapatrier des courriers électroniques situés sur un serveur de messagerie.

SMB : Protocole réseau utilisé par les ordinateurs Windows qui permet aux systèmes d'un même réseau de partager des fichiers.

NFS : Protocole de système de fichiers distribués permettant à un utilisateur sur un ordinateur client d'accéder aux fichiers sur un réseau informatique de la même manière que sur un stockage local.

Adaptateurs

Les adaptateurs sont des modules individuels pour la connexion à des applications ou des technologies spécifiques. Ils permettent de ne pas avoir à s'inquiéter de savoir comment intégrer une application ou une technologie au bus. L'appel de l'adaptateur est aussi simple que l'appel à un service. L'utilisation d'adaptateurs permet d'éviter aux équipes de développement une importante écriture de code. Deux types d'adaptateurs peuvent être recensés :

Adaptateurs d'application : Ils permettent de faciliter la connexion au bus d'applications telles que SAP ou Microsoft Dynamics. Cette liste d'applications devra être complétée en fonction du contexte et des besoins.

Adaptateurs de technologie : Ils ont pour mission de faciliter la connexion à des technologies spécifiques comme :

Les bases de données : Un ESB peut présenter des capacités de connectivité vis à vis de systèmes de stockage de données. Ceux-ci pourront être de divers types :

Relationnel : Les bases de données relationnelles telles que Oracle, SQL Server, MySql et autres constituent le type le plus répandu de base de données en entreprise. Un ESB doit donc offrir la possibilité d'y réaliser des opérations.

Clé-Valeur : Systèmes de gestion de données très simples qui ne stockent que des paires clé-valeur et fournissent des fonctionnalités de base pour récupérer la valeur associée à une clé connue. Exemple : Redis, Amazon DynamoDB.

Wide-Column : Systèmes proposant un stockage des données sous forme de colonnes (plutôt que sous forme de lignes). Ils sont particulièrement adaptés aux applications d'analyse de données car ils permettent une extraction très rapide de colonnes de données. Exemple : Cassandra, Scylla.

Document : Systèmes sans schéma qui stockent les données sous forme de documents JSON. Ils sont similaires au type clé-valeur mais la clé est constituée par le nom du document tandis que la valeur stocke le contenu du document. Exemple : MongoDB, Couchbase.

Moteur de recherche : Systèmes de stockage de données utilisant des documents JSON libres de schéma. Ils sont similaires au type précédent mais mettent d'avantage l'accent sur la facilité d'accès aux données non structurées ou semi-structurées via des recherches textuelles avec des chaînes de complexité variable. Exemple : ElasticSearch, Splunk, Solr.

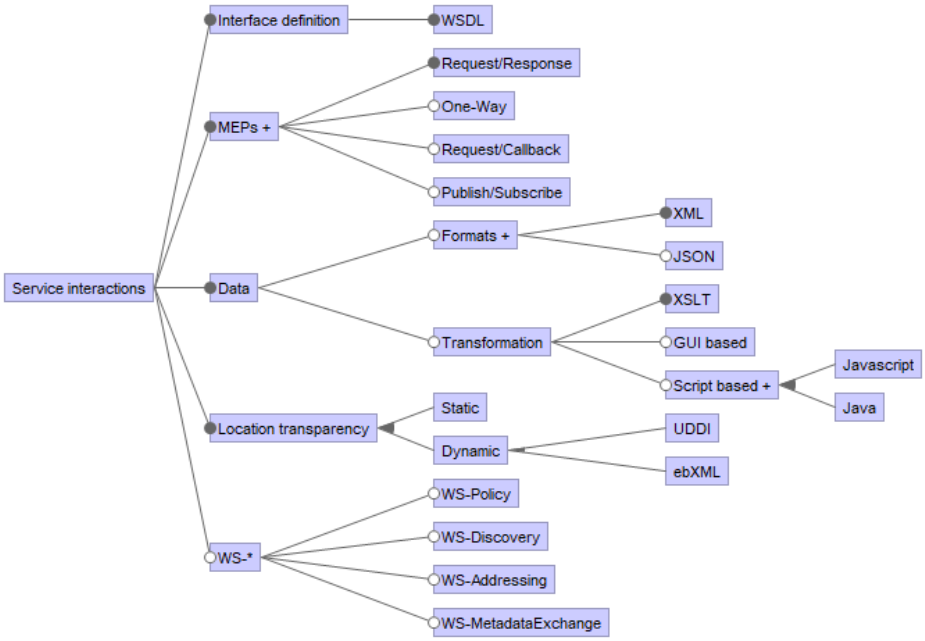


FIGURE 3.4 – Capacités de la catégorie « Service Interactions ».

Les systèmes de messagerie (MOM) : Il peut s'avérer nécessaire de permettre l'envoi et la réception de messages vers et de MOM externes à l'ESB tels que ActiveMQ, MSMQ ou d'autres.

Enfin, si l'ESB ne fournit pas d'adaptateur pour une application particulière, il pourra néanmoins fournir un framework de développement permettant la création d'adaptateurs personnalisés.

3.4.3 Interactions de services

Une fois les acteurs connectés au bus, ces derniers doivent pouvoir entamer des interactions. Cette catégorie contient les fonctionnalités permettant de créer ces interactions. Elles sont illustrées sur la figure 3.4.

Définition d'interface

Un fournisseur de service a besoin d'un moyen de décrire les fonctionnalités qu'il offre aux clients. Il est pour cela préférable d'utiliser une notation standardisée qui permet au plus grand nombre de clients potentiels de comprendre et utiliser ce service. WSDL est un langage basé sur XML utilisé pour décrire les fonctionnalités offertes par un web service. Une description WSDL d'un web service est compréhensible par une machine et lui permet d'identifier comment un web service peut être appelé, quels paramètres il requiert et quelle structure de données il retourne.

Modèles d'échange de messages

Un modèle d'échange de messages (MEP) est un modèle orienté réseau qui décrit comment les clients et fournisseurs de services se connectent et communiquent entre eux. Les MEP cités décrivent les modèles d'échange principaux qu'un ESB doit permettre de mettre en place. Nous ne traitons pas ici des modèles d'échange spécifiques à des technologies comme SOAP, REST ou encore Corba.

Request/Response : Modèle d'échange de message le plus connu. Il décrit une communication synchrone dans laquelle un client envoie une requête à un fournisseur et attend une réponse (ou un timeout) en retour. Un ESB doit être capable d'établir une communication selon ce modèle.

One-Way : Dans ce modèle, le client envoie une requête et continue aussitôt son exécution sans attendre une réponse du fournisseur. Il est aussi appelé « Fire and Forget ».

Request/Callback : Dans ce modèle, le client envoie une requête à un fournisseur de service mais ne reste pas bloqué sur l'attente de la réponse. Il prévoit un mécanisme de callback qui prend en charge la réponse à son arrivée.

Publich/Subscribe : Dans ce modèle, des applications réceptrices définissent le type de données qui les intéresse. Lorsqu'une application émettrice transmet des données à l'ESB, c'est-à-dire qu'elle les publie, l'ESB distribue ces données conformément aux spécifications de l'application réceptrice.

Données

Un ESB facilite l'interaction entre clients et fournisseurs de services au niveau des données selon deux catégories de capacités :

Formats : Un ESB doit être capable de prendre en charge divers formats de données. XML en tant que standard de représentation des données dans les ESB, doit être pris en charge. JSON dont la popularité est grandissante sera souvent pris en charge. D'autres format de données pourront naturellement venir compléter la liste présentée en fonction des besoins.

Transformation : il doit être capable non seulement de prendre en charge des formats de données mais il doit aussi fournir les mécanismes permettant le passage d'un format à un autre. Ces mécanismes permettront une traduction à la fois sur le plan syntaxique mais également sur le plan sémantique. Ils pourront être d'un ou plusieurs des types suivant :

XSLT : Langage de transformation de documents XML. Il décrit comment transformer la structure d'un document XML en un document XML de structure différente.

GUI based : Interface graphique permettant de concevoir ces transformations.

Script based : Langages permettant de scripter les transformations nécessaires. JavaScript sera souvent le langage supporté mais d'autres comme Java pourraient s'ajouter à la liste.

Transparence de localisation

Lorsqu'un client communique avec un fournisseur de service à travers un ESB, le client n'a pas besoin de connaître la localisation réelle du fournisseur. La transparence de localisation est cette capacité de l'ESB qui permet ce découplage spatial entre les acteurs. Elle pourra être de deux natures :

Statique : Il s'agit de l'implémentation la plus simple. Elle est le plus souvent réalisée dans un fichier de configuration qui sera souvent au format XML. Sa modification entraînera la nécessité d'un redémarrage des services ESB.

Dynamique : Dans une configuration plus avancée, un ESB permettra une modification dynamique de la localisation des fournisseurs de services soit via un fichier de configuration déployable à chaud, soit via une base de données interne de l'ESB, soit via l'utilisation d'un registre de services permettant de communiquer la qualité de service offerte ou encore d'inclure des informations métiers importantes. Deux types de registre ont été recensés :

UDDI :[3] Registre basé sur XML destiné aux web services professionnels. Un fournisseur peut explicitement inscrire un service auprès d'un registre de web services tel que UDDI ou publier des documents supplémentaires destinés à faciliter la découverte. Les utilisateurs de services ou les consommateurs peuvent rechercher des web services manuellement ou automatiquement.

ebXML :[4] Ensemble de spécifications basées sur XML permettant la publication d'informations sur les services afin d'assurer les échanges électroniques entre professionnels (B2B).

WS-*

Afin de garantir l'interopérabilité, l'interaction entre les clients et les fournisseurs de services devrait être soutenue par des normes. Il existe une variété de spécifications associées aux web services. Elles peuvent se compléter, se chevaucher et se concurrencer les unes les autres. Nous avons réparti les normes disponibles entre les catégories identifiées de notre modèle. Cette catégorie ne comprend que les normes qui incluent des informations sur l'interaction avec les web services dont :

WS-Policy : Spécification permettant aux web services de communiquer leurs politiques par exemple en matière de sécurité ou de qualité de service et aux clients d'indiquer leurs exigences en la matière.

WS-Discovery : Spécification technique qui définit un protocole de découverte multicast pour localiser les services sur un réseau local.

WS-Addressing : Spécification d'un mécanisme neutre pour le transport qui permet aux services Web de communiquer des informations d'adressage. Il s'agit d'un moyen normalisé d'inclure les données de routage des messages dans les headers SOAP.

WS-MetadataExchange : Spécification d'un protocole de web service conçu pour fonctionner en conjonction avec WS-Addressing, WSDL et WS-Policy pour permettre la récupération de méta-données concernant un web service.

3. Universal Description, Discovery, and Integration
4. Electronic Business using eXtensible Markup Language

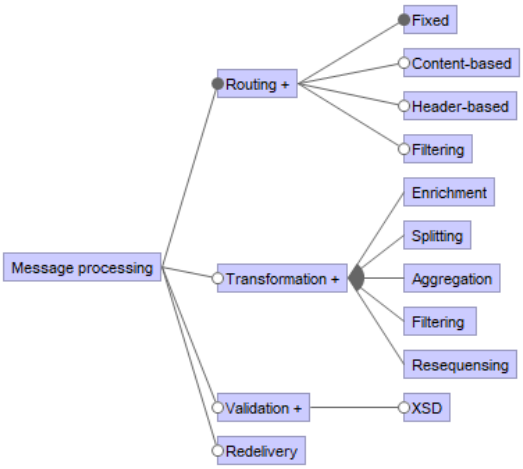


FIGURE 3.5 – Capacités de la catégorie « Message Processing ».

3.4.4 Traitement des messages

Le MOM constitue le coeur de l'ESB. L'ensemble des communications entre les systèmes connectés passe par lui sous la forme de messages qui pourront subir divers traitements permettant la médiation entre les systèmes qui communiquent. La figure 3.5 illustre ces capacités de traitement.

Routage

Un ESB doit permettre d'orienter les messages selon plusieurs méthodes dont :

Fixed : Lorsqu'un client doit simplement être découplé d'un fournisseur de service, aucun routage particulier n'est nécessaire. Le MOM joue alors simplement un rôle de queue entre les deux systèmes.

Content-based : L'ESB accède au contenu des messages et analyse ceux-ci afin de déterminer, selon des règles métiers pré-définies, la destination finale des messages.

Header-based : Le client ajoute une entête au message désignant la destination de la requête. L'ESB consulte cette entête afin de déterminer où transférer le message.

Filtering : Il ne s'agit pas à proprement parler d'un routage mais un ESB doit être capable de filtrer les messages sur base de règles pré-définies afin que seuls les messages adéquats parviennent aux fournisseurs de services.

Transformation

Lors du transit des messages dans le MOM, il peut être nécessaire d'effectuer des modifications sur ces messages. Ces transformations pourront être de natures variées :

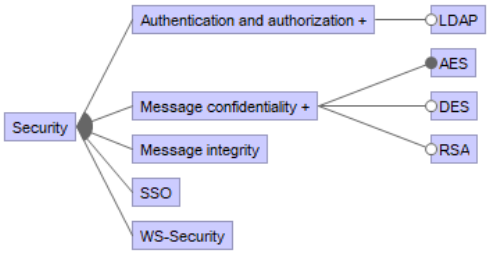


Figure 3.6 – Capacités de la catégorie « Security ».

Enrichement : Un message peut être augmenté avec des informations supplémentaires provenant par exemple d'un service web, d'une base de données ou encore d'un fichier.

Splitting : Permet de séparer un message composé de données répétitives en autant de messages individuels.

Aggregation : Il permet la collecte et le stockage de messages corrélés. Une fois l'ensemble complet de messages reçus, un message unique est constitué à partir de l'ensemble et envoyé vers un destinataire.

Filtering : Il s'agit de l'inverse de l'augmentation de message. Certaines parties d'un message peuvent ainsi être retirées.

Resequencing : Il s'agit de collecter des messages corrélés mais reçu en désordre afin de les transférer de manière ordonnée.

Validation

La validation de messages s'assure qu'un message est valide. Dans le cas d'un message XML, ceci signifie qu'il contient du code bien défini correspondant à un certain schéma défini en XSD[5]. D'autres mécanismes pourraient cependant intervenir. Ils complèteront dès lors le modèle proposé.

Redélivrance

Si nécessaire, un ESB doit pouvoir assurer qu'un message reçu sera bien transmis à son destinataire. Il doit dans ce cas être capable de stocker un message qui n'a pu être remis et ce tant que la remise n'a pu se réaliser avec succès.

3.4.5 Sécurité

Les aspects de sécurité tels que l'authentification et l'autorisation peuvent être gérés par l'ESB. Même si un service ne dispose pas d'un mécanisme d'authentification et d'autorisation, un ESB peut l'exiger dans l'interface de service qu'il expose aux clients potentiels de ce service. Les capacités de cette catégorie sont décrites ci-dessous.

5. XML Schema Definition

Authentication and authorization

Capacité d'identification des applications ou utilisateurs et de gestion des droits de ceux-ci. Un LDAP sera souvent utilisé pour identifier et autoriser les entités. D'autres technologies pourraient cependant être utilisées.

Message confidentiality

Capacité permettant de s'assurer qu'un message ne pourra être lu que par les acteurs concernés. On fait ici référence à des capacités de chiffrement et déchiffrement comme :

DES[6] : Algorithme de chiffrement symétrique par bloc utilisant des clés de 56 bits.

AES[7] : Algorithme de chiffrement symétrique actuellement le plus utilisé et le plus sûr.

RSA : Algorithme de cryptographie asymétrique

Message integrity

Capacité permettant à l'ESB d'assurer qu'un message ne peut être modifié durant son transport grâce à l'utilisation de signatures numériques.

Single Sign On (SSO)

Service qui permet aux administrateurs de mapper un compte utilisateur avec un ou plusieurs comptes alternatifs. Ces comptes sont mappés par application afin qu'ils puissent être utilisés pour accéder en toute sécurité à des applications qui nécessitent des informations d'identification autres que celles fournies initialement par l'utilisateur.

WS-Security

Extension de SOAP permettant à ce dernier de spécifier comment l'intégrité et la confidentialité peut être appliquées aux messages et permettant la communication de divers formats de jetons de sécurité (SAML[8], Kerberos, X.509).

3.4.6 Gestion

Un ESB doit pouvoir être configuré, géré et surveillé.

6. Data Encryption Standard
7. Advanced Encryption Standard
8. Security Assertion Markup Language

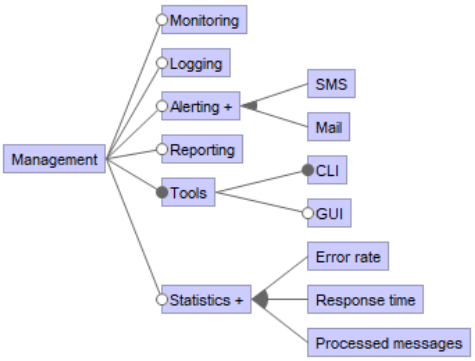


FIGURE 3.7 – Capacités de la catégorie « Management ».

Tools

Un ESB possède fondamentalement un caractère fortement distribué et décentralisé. Cependant il doit permettre d'être configuré, géré et surveillé de manière centralisée via une seule application. Celle-ci pourra être d'une ou des deux natures suivantes :

CLI : Interface en ligne de commande

GUI : Interface graphique

Monitoring

Le monitoring permet de surveiller les paramètres de l'ESB. Utilisé en parallèle à des SLA [9], il permet la définition de règles qui si elles sont enfreintes, déclenchent une alerte éventuellement signalée via les capacités d'alerting de l'ESB.

Logging

Il permet d'enregistrer les messages et de les rechercher facilement par la suite.

Alerting

Un ESB peut éventuellement fournir des capacités d'alerte. Elles consistent à avertir les bons intervenants d'une situation anormale. Ceci pourra être accompli grâce à des mécanismes basiques comme :

Mail : Les alertes sont envoyées sous forme d'emails.

SMS : Les alertes sont envoyées sous forme de SMS.

Éventuellement, d'autres canaux pourraient être nécessaires pour par exemple intégrer des systèmes de surveillance existants.

Reporting

Capacité de création de rapport ou de sortie de données sous divers formats.

9. Service Level Agreement

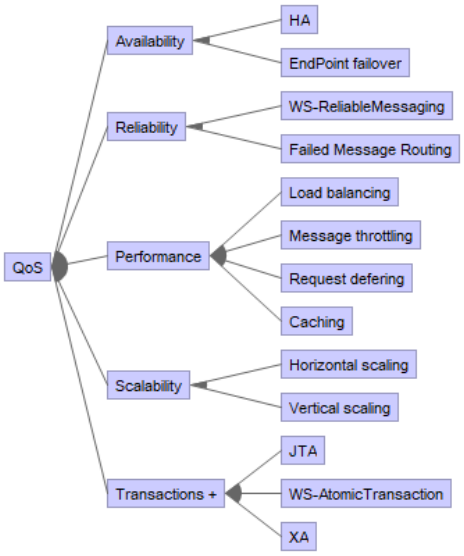


FIGURE 3.8 – Capacités de la catégorie « QoS ».

Statistics

Capacité de fournir des statistiques comme :

Error rate : Statistiques sur le nombre d'erreurs

Response time : Statistiques sur les temps de réponse (minimim, maximum, moyen).

Processed messages : Statistiques sur le nombre de messages traités

3.4.7 Qualité de services

Les options de qualité de service (QoS) sont liées à la qualité de l'utilisation d'un web service et précisent les exigences associées à la fiabilité globale des services. La qualité de service pourra couvrir les aspects suivants :

Availability

La disponibilité des services est importante dans une SOA car il est probable que plusieurs applications dépendent d'un seul service. Si ce service unique devient indisponible ou ne répond plus, toutes les applications qui dépendent de ce service deviennent indisponibles. Une implémentation d'une SOA réussie doit être capable d'assurer un niveau de disponibilité à chaque service et à chaque application qui dépend de ce service. Les capacités de cette catégorie sont :

High Availability : Haute disponibilité d'un service quel que soit le statut des serveurs sur lesquels il fonctionne

Endpoint failover : Mécanisme permettant de grouper un ensemble de services similaires. Si un message est envoyé à un service et qu'il ne répond pas, l'ESB envoie le message au service suivant jusqu'à ce qu'un service réponde.

Reliability

Les bénéfices du faible couplage apporté par l'ESB se réalisent au détriment du contrôle du processus de communication entre les acteurs de services. En effet, après qu'un service ait transmis un message, il n'a aucun moyen immédiat de savoir :

— Si le message est parvenu à destination.

— Si le message n'est pas arrivé et nécessite une retransmission.

— Si une série de message est arrivée dans l'ordre dans lequel il devrait être reçu.

Des messages fiables répondent à ces préoccupations en établissant une mesure d'assurance de la qualité et fournissent également une notification garantie du succès ou de l'échec de la livraison. Les capacités suivantes pourront être utilisées :

WS-ReliableMessaging : WS-ReliableMessaging décrit un protocole qui permet aux messages d'être délivrés de manière fiable entre des applications distribuées en présence de pannes de composants logiciels, de systèmes ou de réseaux.

Failed Message Routing : Lorsqu'un message échoue sur un port de réception, il est acheminé vers un endroit où des mesures supplémentaires peuvent être prises.

Performance

Les capacités de médiation d'un ESB font toute son utilité. Cependant, elles ont un coût sur les performances des communications. Le coût encouru pour l'appel d'un service via un ESB par rapport à l'appel direct dépend de l'architecture de l'ESB. Ce dernier pourra augmenter ses performances via diverses stratégies comme :

Load balancing : Possibilité de déployer plusieurs instances d'un service et d'utiliser un répartiteur de charge pour répartir les demandes.

Message throttling : Configuration permettant de n'autoriser qu'un nombre spécifique de messages à atteindre le service dans un laps de temps donné.

Request defering : Possibilité de retarder des requêtes en attendant un moment plus favorable tenant compte de la charge du service ciblé.

Caching : Possibilité de stocker les dernières requêtes effectuées sur un service. Si une requête similaire survient, il renvoie le résultat de la requête précédente sans appel réel au service diminuant par la sorte la charge de ce dernier.

Scalability

Pour faire face à des charges de travail élevées, un ESB peut prendre en charge différentes options d'évolutivité :

Horizontal scaling : Augmentation des performances et de la tolérance aux pannes en exécutant plusieurs instances d'ESB sur différentes machines.

Vertical scaling : Augmentation des performances en utilisant un matériel (processeur et mémoire) plus nombreux et/ou plus performant.

Transaction

Une transaction est constituée d'un ensemble d'opérations qui doivent être exécutées comme une seule unité de travail. Elle est caractérisée par les propriétés ACID pour Atomicité, Cohérence, Isolation et Durabilité. L'atomicité est la propriété selon laquelle une transaction est effectuée en entier ou pas du tout. La cohérence permet d'assurer que chaque transaction n'affectera pas l'état valide du système. L'isolation assure qu'une transaction ne dépend pas d'une autre. Enfin, la durabilité permet d'assurer qu'une fois une transaction validée, son résultat est durable même en cas de panne du système. Un ESB peut prendre à sa charge la gestion des transactions de manière à éviter au client de devoir le faire. Pour cela, un ESB pourra offrir différentes options comme :

JTA : Java Transaction est un API présent dans la spécification JavaEE et fournissant des interfaces standards permettant une validation à deux phases.

WS-AtomicTransaction : Elle définit un ensemble spécifique de protocoles et de services qui assurent l'activation, l'enregistrement, la propagation et la terminaison atomiques des web services.

XA : XA est un protocole de validation en deux phases qui est nativement pris en charge par de nombreuses bases de données et moniteurs de transactions.

3.5 Orchestration

À ce stade et bien que nous ayons déjà abordé le sujet dans le section 1.4.4, il nous semble important de revenir ici sur les capacités d'orchestration des ESB. Le lecteur aura en effet remarqué qu'il n'en est nullement fait mention dans le modèle proposé. À notre sens, l'orchestration peut être de deux types. En premier lieu, on distinguera les orchestrations longues, accompagnées d'un état, complexes par nature et impliquant souvent des actions humaines (approbation, délégation, assignation).

Un exemple d'une orchestration de ce type est le flux de travail lié à la prise de rendez-vous, l'exécution et enfin la communication des résultats d'une consultation médicale pour un patient hospitalisé dont une version simplifiée pourrait être :

1. Le service d'agenda fournit les disponibilités pour un type de consultation donné.
2. Le personnel soignant choisit un créneau.
3. Le service d'agenda réserve le créneau choisi.
4. Le médecin réalise la consultation.
5. Le médecin dicte les résultats de l'examen et envoie la dictée au secrétariat médical.
6. La secrétaire tape le protocole d'examen et l'envoie pour validation au médecin.
7. Le médecin valide le procotocle.
8. Le protocole est envoyé dans le dossier médical électronique du patient.

Ce type de flux comporte des interactions de différents types. Toutes ne sont pas longues mais le processus dans son ensemble l'est. Ce type de flux ne fait pas partie des missions d'un ESB mais s'apparente plus aux workflow ou processus métiers. Ces derniers peuvent être modélisés avec BPMN et confiés à des moteurs d'exécution de workflow comme Activiti, Orchestra ou encore jBPM [35, 66].

Le second type d'orchestration est caractérisé par des processus de courte durée, sans état et traitant des données selon des modèles de messagerie [4]. Ce type d'orchestration est généralement plus simple et peut faire appel à plusieurs services qu'il enchaine afin de traiter une demande. Ces services permettront de diviser, agréger, augmenter, filtrer, journaliser ou encore valider des messages. On retrouve ici les diverses notions de notre modèle des capacités. Un ESB est par nature capable d'orchestrer ces diverses opérations comme le montre l'exemple suivant :

> L'étape 1 du flux décrit ci-dessus peut constituer un exemple de ce second type d'orchestration. Imaginons que l'hôpital de notre exemple comporte plusieurs sites et que chacun d'eux possède son propre agenda.
>
> 1. Une application envoie à l'ESB une demande de disponibilité pour une consultation pour une spécialité médicale donnée.
> 2. L'ESB clone le message pour qu'il soit envoyé au service agenda de chaque site. Il effectuera des transformations si nécessaires.
> 3. L'ESB une fois qu'il a reçu les réponses de chaque agenda, les agrège en un seul message qu'il retourne à l'application.

3.6 Résumé

Dans ce chapitre, nous avons décrit la méthodologie suivie pour rechercher, trier et analyser la littérature scientifique à la source de l'élaboration de notre modèle des capacités. En premier lieu, nous avons construit notre requête de recherche sur base des concepts présents dans notre question de recherche. Un corpus final de 18 publications a ensuite été établi en suivant les 5 étapes suivantes :

1. élimination des publications en doublons ;
2. triage sur base du titre et de l'abstract des publications ;
3. triage sur base d'une lecture complète des publications ;
4. analyse des références (snowballing) ;
5. ajout libre de références.

Les diagrammes de fonctionnalités ont été choisis comme formalisme du modèle car ils permettent une représentation visuelle et hiérarchique de celui-ci. De plus ils permettent de définir et visualiser le caractère optionnelle ou obligatoire des fonctionnalités ainsi que la nature des relations entre ces dernières. Une analyse des publications a été menée et a débouché sur l'élaboration d'un modèle détaillé des capacités des ESB comprenant 6 catégories de fonctionnalités principales :

1. La connectivité.
2. Les interactions de services.
3. Le traitement des messages.
4. La sécurité.

5. La gestion.
6. La qualité des services.

Chapitre 4

Évaluation et sélection d'un ESB

Dans les chapitres précédents, nous avons parcouru le paysage de l'intégration des systèmes d'information. Nous nous sommes attardés sur l'intégration d'applications en décrivant les niveaux auxquels elle intervient ainsi que les styles et architectures qui y sont utilisés. Nous nous sommes ensuite attelés à décrire précisément ce qu'est un ESB, quelles sont ses caractéristiques et dans quel contexte les utiliser. Un modèle de leurs capacités a été élaboré dans le chapitre précédent. Dans ce chapitre, nous proposerons une méthode d'évaluation et de sélection d'un ESB en tirant parti de ce modèle. Pour cela, nous rechercherons en premier lieu les travaux éventuellement disponibles sur ce thème. Nous constaterons qu'ils sont très peu nombreux. La recherche sera élargie à l'évaluation et la sélection de logiciel de manière plus générale et nous proposerons, à partir de la littérature récoltée, une méthode d'évaluation et de sélection d'ESB. Nous terminerons par appliquer notre méthodologie dans le choix d'un ESB dans le contexte de notre cas d'accompagnement.

4.1 Méthodes de sélection et d'évaluation des ESB

Pour établir notre méthode, nous avons en premier lieu rechercher la littérature sur les méthodes déjà proposées. Nous avons pour cela construit notre requête selon la même méthodologie que dans la section 3.2.1 et ce à partir de la question « Quelles sont les méthodes d'évaluation et de sélection d'un ESB ? ». La chaîne de recherche déduite, les moteurs de recherche utilisés ainsi que les résulats obtenus sont indiqués dans la table 4.1. La colonne « N. » y indique le nombre de résultats retournés par le moteur correspondant tandis que la colonne « F. » y indique le nombre de publications restantes après l'application d'une étape de tri basée sur la lecture du titre et de l'abstract. Seul les articles traitant d'une méthode de sélection des ESB ont été conservés. Google Scholar a été ajouté suite à l'absence d'un résultat probant après consultation des trois premiers moteurs de recherche. Il a fourni 35700 résultats dont seuls les 150 premiers ont été analysés pour fournir au final un ensemble de 3 publications utiles.

[63, 64] présentent tous deux une évaluation des ESB qui utilisent la méthode

Table 4.1 – Requêtes de collecte de publications par moteur de recherche.

Librairie	Requête de recherche	T.	F.
ACM	((ESB OR "Enterprise Service Bus") AND (select OR selection OR choice) AND ("best practice*" OR selection OR method OR methodolog* OR strateg* OR procedure OR recommandation OR guideline*))	27	0
IEEE Xplore	((ESB OR "Enterprise Service Bus") AND (select OR selection OR choice) AND ("best practice*" OR selection OR method OR methodolog* OR strateg* OR procedure* OR recommandation OR guideline*))	22	0
Springer Link	((ESB OR "Enterprise Service Bus") AND (select OR selection OR choice) AND ("best practice*" OR selection OR method OR methodolog* OR strateg* OR procedure* OR recommandation OR guideline*)) La recherche a été limitée au contenu de type « Article » dans la discipline « Computer Science »	137	0
Google Scholar	((ESB OR "Enterprise Service Bus") AND (select OR selection OR choice) AND ("best practice*" OR selection OR method OR methodolog* OR strateg* OR procedure* OR recommandation OR guideline*))	150+	3

AHP [1]. Dans [63], Shezi et al. évaluent les ESB selon 5 critères : haute disponibilité, transformation de messages et de données ;routage intelligent ;orchestration de service ; découverte dynamique de service. L'article se focalise sur la description de la méthode AHP sans apporter d'explication sur la manière dont les différents critères sont évalués. Dans [64], Siddiqui, Abdullah et Khan évaluent et sélectionnent les ESB sur base de 3 critères : sécurité de l'information ; interopérabilité ; haute disponibilité. Comme dans [63], une partie importante de l'article est consacrée à l'application chiffrée de la méthode AHP.

[76] est un rapport « Forrester Wave » de l'année 2011 sur les ESB. Il propose une évaluation basée sur 109 critères dont la liste n'est pas fournie et qui sont répartis en 3 groupes : offre fonctionnelle ; stratégie du fournisseur ; présence sur le marché. Chaque groupe reçoit une pondération de même que chaque critère au sein des différents groupes. Le total de chaque groupe est calculé en fonction de la pondération de chaque critère tandis que l'évaluation finale d'un ESB est calculée d'après la pondération de chaque groupe. Bien qu'il n'en soit pas fait explicitement mention, la méthodologie d'évaluation utilisée est à nouveau empruntée aux méthodes de décision multicritère. En l'occurrence, il s'agit de la méthode WSM [2].

En conclusion de la recherche et de l'analyse des articles concernant l'évaluation d'ESB, quelques enseignements peuvent déjà être tirés. Premièrement, il existe très peu de référence traitant de l'évaluation et de la sélection des ESB. En second lieu, sur les 3 articles trouvés, 2 traitent d'aspects spécifiques des ESB tandis que le dernier aborde le problème de manière plus générale. Aucun ne décrit précisément une méthodologie de sélection. Troisièmement, les 3 références appliquent des techniques de décision multicritère à savoir l'AHP et la WSN.

1. Analytic Hierarchy Process : méthode de décision multicritère développée par le professeur Thomas L. Saaty
2. Weighted Scoring Method

4.2 Méthodes générales de sélection et d'évaluation de logiciels

Au vu du faible nombre d'articles identifiés par notre recherche sur les méthodes de sélection d'un ESB, nous avons étendu notre recherche aux méthodes générales d'évaluation et de sélection de logiciels. Nous avons en particulier rechercher les articles établissant une revue de ces méthodes sur les moteurs de recherche suivant : IEEE Xplore ; ACM ; Science Direct.

Table 4.2 – Requêtes de recherche d'articles généralistes sur la sélection de logiciel.

Librairie	Requête de recherche
ACM	acmdlTitle :(software cots) AND acmdlTitle :(selection evaluation selecting evaluating) AND acmdlTitle :(review survey)
IEEE Xplore	(("All Metadata" :software OR cots OR select* OR evalut*) AND "Document Title" :survey review)
Science Direct	Recherche effectuée sur les champs Title, abstract ou author-specified keywords. (software* OR cots) AND ((evaluation AND selection) OR (evaluating AND selecting)) AND (review OR survey)

Après analyse des résultats retournés par les différents moteurs de recherche, nous avons retenu un seul article. Il s'agit de [30] qui présente une revue systématique de la littérature concernant la sélection et l'évaluation de logiciels et qui est fortement référencé dans la littérature. Cette référence tente d'apporter des réponses à différentes questions de recherche. Celles qui nous intéressent sont les suivantes :

1. Quelle est la contribution de la littérature dans le champ de l'évaluation et de la sélection de logiciels ?
2. Quelles sont les méthodologies de sélection d'un logiciel ?
3. Quelles sont les techniques d'évaluation de logiciels ?

La première va nous permettre d'explorer la littérature à la recherche d'informations pertinentes. Nous nous focaliserons pour cela sur 10 références traitant de la sélection ou de l'évaluation des logiciels en général et non sur celles se focalisant sur un type de logiciel particulier comme un logiciel de comptabilité ou de gestion des connaissances. Nous recherchons en effet des informations générales qui nous permettront de construire notre propre méthode et souhaitons donc éviter des informations trop spécifiques. Les deuxième et troisième questions nous permettront d'identifier respectivement les différentes méthodologies de sélection et techniques d'évaluations utilisées. Elles nous permettront de construire et proposer notre méthodologie d'évaluation et de sélection d'un ESB.

4.2.1 Revue de la littérature

À partir de notre article de référence ([30]), 10 références traitant de manière générale de la sélection et de l'évaluation de logiciels ont été identifiées. Cette section décrit la contribution de chacun de ces articles à l'élaboration de notre méthodologie.

Dans [14], Chau présente une étude sur les critères de sélection de logiciels dans les petites entreprises. Ces dernières ne possèdent généralement par de procédure formelle de sélection. Les auteurs présentent trois groupes de critères selon lesquels les logiciels sont évalués et étudient l'importance donc le poids accordé à chacun d'eux. Les trois groupes de critères sont les critères liés au logiciel, au vendeur et aux opinions. Chacun d'eux possède une part de critères techniques et une autre de critères non techniques. Son enquête permet de constater qu'une plus grande importance est accordée au groupe de critères liés au logiciel suivi par ceux liés au vendeur et qu'une plus grande importance est donnée aux opinions des experts internes plutôt qu'à celles des externes.

Comella-Dorda et al. dans [15] constatent les problèmes et erreurs rencontrés dans la sélection de logiciels COTS [3] et proposent un processus d'évaluation adaptable en fonction des besoins. Ils affirment que la sélection d'un logiciel ne peut être efficace que si elle suit un processus formel bien défini. Le processus proposé, nommé PECA, est composé de 4 éléments basiques : planifier l'évaluation ; établir les critères ; collecter les données ; analyser les données. La planification comprend la formation d'une équipe d'évaluation, la création d'une charte définissant la portée et les contraintes, l'identification des parties prenantes, le choix d'une approche d'évaluation et enfin l'estimation des ressources nécessaires et du timing. L'établissement des critères d'évaluation est constitué de l'identification des exigences et de la construction des critères qui seront composés de deux éléments :un énoncé du critère et une échelle de quantification. La collecte des données consistera à exécuter le plan d'évaluation pour déterminer l'adéquation des divers produits. Enfin, l'analyse des données permettra de fournir des recommandations aux décideurs.

Kontio propose dans [33] une méthode systématique, ré-utilisable et orientée exigences de sélection de logiciel, baptisée OTSO [4]. Elle repose sur 4 principes : la définition explicite des tâches du processus de sélection ; une définition détaillée et hiérarchique des critères d'évaluation ; la construction d'un modèle d'évaluation de chaque alternative ;l'utilisation d'une méthode de prise de décision appropriée pour analyser et résumer les résultats d'évaluation. L'auteur développe les aspects de définition des critères et d'analyse qualitative des données d'évaluation. Il utilise pour cela la méthode AHP.

Une méthode d'évaluation basée sur le modèle de Brown–Gibson est proposée par Larry Sanders, Ghandforoush et Austin dans [39]. Il s'agit à nouveau d'une technique de décision multicritère. Elle possède cependant la particularité de prendre en compte trois types de critères : critique ; objectif ; subjectif. Un critère critique représente un critère qui s'il n'est pas satisfaisant peut conduire à l'élimination d'une alternative [5].

Dans [40], Lawlis et al. présentent la méthode RCPEP [6]. Il s'agit d'une méthode générale d'évaluation de logiciels, orientée exigences et composées de deux grandes étapes. La première consiste à : définir les critères d'évaluation à partir des exigences ; définir une liste de l'ensemble des produits susceptibles de satisfaire ces exigences ; évaluer chaque produit en questionnant les vendeurs et si possible des utilisateurs ; établir une short liste de produits. La seconde étape consiste à tester

3. Commercial Off-The-Shelf
4. Off-The-Shelf Option
5. Dans les méthodes de prise de décision multicritère, une alternative désigne un candidat au problème proposé.
6. Requirements-driven COTS Product Evaluation Process

chaque produit de la short liste par plusieurs analystes et sur base de scénarios pratiques. Chaque étape de sélection utilise la méthode WSN.

Une autre méthode d'évaluation et de sélection est proposée par Morera dans [49]. Elle est composée de deux grandes étapes, chacune adaptant une méthodologie couvrant des objectifs différents. La première étape consiste à évaluer tous les candidats potentiels à un problème donné et à fournir une short liste de candidats. La seconde étape consiste à évaluer ces candidats selon la méthode AHP afin de désigner le candidat le plus adéquat. Au final, l'auteur propose donc une méthodologie de sélection reposant sur 9 étapes réparties en deux grands processus :filtrage de l'ensemble des candidates ;identification du meilleur candidat de la short liste.

Morisio et Tsoukiàs proposent une méthodologie basée sur une approche multicritère d'aide à la décision [50]. Ils définissent un processus d'évaluation qui comprend deux phases principales : la conception d'un modèle d'évaluation et son application. Chaque phase est elle-même découpée en plusieurs étapes. Une particularité de cette technique est que la méthodologie multicritère d'aide à la décision est non fixée. Elle est choisie en fonction de chaque projet de sélection.

Dans [71], Toshtzar propose une méthodologie de sélection d'un logiciel en utilisant uniquement la technique multicritère d'aide à la décision AHP. Elle consiste à définir un modèle d'évaluation sous la forme d'un hiérarchie de critères. Cette hiérarchie est ensuite utilisée dans la méthode AHP.

[77] Welzel et Hausen proposent dans [77] une méthodologie de sélection de logiciel organisée en 5 étapes. La première consiste à définir les exigences de l'évaluation c'est à dire analyser les conditions et contraintes de l'évaluation et à définir les objectifs. La seconde spécifie l'évaluation en fournissant une description formelle des exigences de l'évaluation. La troisième définit le processus d'évaluation en choisissant des modules d'évaluation parmi une librairie de modules. La quatrième consiste à effectuer l'évaluation telle que définie lors de l'étape précédente et enfin la cinquième et dernière étape est la production du rapport d'évaluation qui décrit les résultats mais reprend également une documentation de chaque étape.

4.2.2 Enseignements

L'analyse de ces différents articles nous permet de tirer quelques enseignements. Premièrement, les méthodes basées sur des étapes représentent une forte partie des méthodes proposées. Seul [71] ne propose pas une méthodologie par étape. Les étapes proposées varient en nombre et objectif mais on peut néanmoins constater une tendance à procéder en premier lieu à un écrémage des alternatives. Il produit une short liste qui est évaluées à la recherche de la meilleure alternative. Secondement, les méthodes multicritères d'aide à la décision sont presque systématiquement utilisées. Elles sont utilisées avec succès depuis des décennies dans différents domaines [75] et permettent d'objectiver un processus de décision. Enfin troisièmement, la sélection d'un logiciel peut s'apparenter à un projet. On retrouve d'ailleurs des mots couramment utilisés dans la gestion de projet : analyse ; contraintes ; exigences ; méthodologie ; parties prenantes ;planification ; phases ; processus.

4.2.3 Contraintes de la méthodologie

Dans la suite de ce chapitre, nous allons proposer une méthodologie d'évaluation et de sélection dédié aux ESB. Celle-ci a été élaborée en tenant compte de

diverses contraintes. D'abord, elle devra couvrir l'entièreté du processus de sélection, de l'émergence d'un besoin jusqu'à l'identification du bon candidat. Ensuite, elle devra tenir compte des divers intervenants et parties prenantes. Enfin la méthodologie devra être adaptable au contexte d'utilisation. Elle devra par exemple être utilisable aussi bien dans une PME qu'au sein d'une grande entreprise.

4.3 Méthodologie ASD (Analyse, Screening, Décision)

Notre méthodologie, baptisée ASD a été fondée sur 3 grandes phases couvrant l'entièreté du processus de sélection d'un ESB :

1. La phase d'**A**nalyse qui a pour fonction : la mise en place des équipes ; l'analyse du problème d'intégration sous-jacent ; la définition des critères d'évaluation.
2. La phase de **S**creening dont la mission est de : produire une liste des ESB candidats ; collecter une première série d'informations les concernant ; effectuer un écrémage des candidats sur base de cette collecte de données.
3. La phase de **D**écision qui comme son nom l'indique doit aboutir à l'identification du meilleur ESB en : collectant des données complètes sur les candidats restants ; analysant et évaluant les informations collectées ; prenant une décision finale.

Chacune de ces phases est composée d'étapes recevant en entrée les données nécessaires à son accomplissement et fournissant en sortie les données requises à l'accomplissement d'une ou plusieurs des étapes suivantes. Chaque étape sera elle même composée d'une ou plusieurs activités. La figure 4.1 expose la méthodologie dans son ensemble. Les phases y sont symbolisée par la coloration des différentes étapes : en rouge l'analyse ; en orange le screening ; en vert la décision. Les prochaines sections décriront précisément pour chaque phase, les différentes étapes qui la constituent. Cette description suivra un canevas unique composé des éléments suivants :

Entrée Description des données nécessaires à l'accomplissement de l'étape.

Equipe Équipe chargée de l'accomplissement de l'étape.

Activité(s) Description des activités qu'il est nécessaire d'entreprendre pour compléter l'étape courante.

Sortie Description du ou des éléments produits par l'étape et nécessaires au démarrage d'une ou plusieurs étapes suivantes.

Notes Informations supplémentaires utiles à la compréhension de l'étape.

4.4 Analyse

Première phase de notre méthodologie, l'analyse poursuit essentiellement trois objectifs : analyser le problème d'intégration ; constituer les équipes qui travailleront sur le projet ; établir les critères qui serviront à l'évaluation des ESB candidats. Ces trois objectifs seront accomplis à travers les trois étapes suivantes.

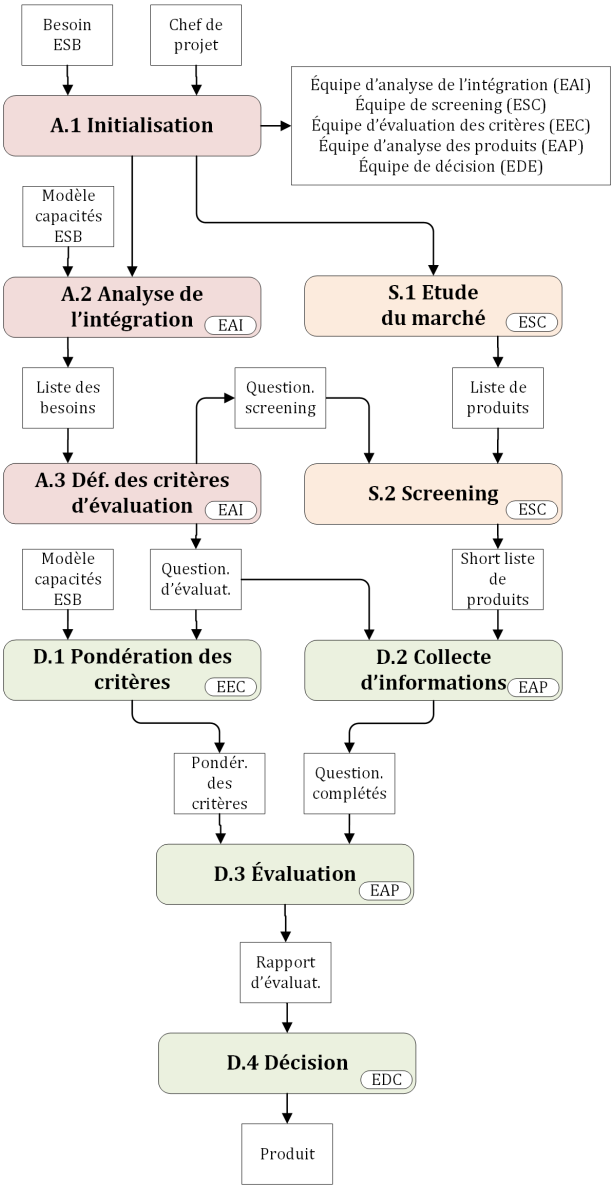


FIGURE 4.1 – Méthodologie de sélection et d'évaluation des ESB.

4.4.1 A.1 Initialisation

la première étape de notre phase d'analyse sera de constituer les équipes chargées d'exécuter les différentes étapes de la méthodologie.

Entrée

— Besoin d'un ESB. Il constitue l'élément déclencheur de l'ensemble du processus de sélection.

— Un chef de projet dont la mission sera de mener à bien le choix d'un ESB.

Équipe

— Le chef de projet.

Activités

— Identifier les entités de l'entreprise concernées par le problème d'intégration.

— Identifier au sein des entités concernées, les ressources humaines nécessaires aux différentes équipes.

— Constituer l'équipe d'analyse de l'intégration. Elle sera chargée de l'étude de l'intégration (étape A.2) et de la définition des critères d'évaluation (étape A.3).

— Constituer l'équipe de screening. Elle aura en charge toute la phase de screening (étape S.1 et S.2).

— Constituer l'équipe d'évaluation. Elle aura pour mission de définir la pondération et donc l'importance relative des différents critères (étape D.1).

— Constituer l'équipe d'analyse des ESB qui aura pour missions de collecter des informations aussi complètes que possible (étape D.2) et d'évaluer les ESB sur base de ces données (étape D.3).

— Constituer l'équipe de décision qui comme son nom l'indique aura la responsabilité du choix final de l'ESB (étape D.4).

Sortie

— Équipe d'analyse de l'intégration.

— Équipe de screening.

— Équipe d'évaluation.

— Équipe d'analyse des ESB.

— Équipe de décision.

Notes

Les différentes activités proposées devront être adaptées en fonction du contexte. La constitution d'une équipe d'analyse pourra se résumer à la désignation d'une personne dans une PME tandis qu'elle pourra nécessiter un travail conséquent dans le cadre d'une grande entreprise possiblement implantée dans différents pays voir continents. Cette remarque vaut également pour les autres équipes.

4.4.2 A.2 Analyse de l'intégration

Entrée

— Modèle des capacités des ESB. Il servira de support à l'identification des besoins.

Table 4.3 – Matrice des flux d'intégration.

de \ vers	Sys1	Sys2	Sys3	Sys4	Sys5	Sys6	Sys7	Sys8	Sys9	Sys10
Sys1	x									
Sys2		x								
Sys3			x							
Sys4				x						
Sys5					x					
Sys6						x				
Sys7							x			
Sys8								x		
Sys9									x	
Sys10										x

Équipe

— Équipe d'analyse de l'intégration

Activités

— Répertorier tous les systèmes internes et externes concernés.

— Établir une matrice des flux. Elle indiquera pour chaque paire de systèmes la présence ou non d'une interface entre eux. La table 4.3 montre un modèle vierge de matrice des flux. Un numéro à l'intersection de deux systèmes indiquera la présence d'une interface entre ces deux systèmes. Celui en entête de ligne constitue l'origine du flux (le client) tandis que celui en entête de colonne indique la destination (le fournisseur). Une fois complété, l'équipe disposera d'un inventaire numéroté des interfaces nécessaires.

— Établir les besoins pour chaque interface identifiée dans la matrice des flux. Le modèle des capacités constituera un support de choix pour cette activité. Chaque catégorie principale du modèle de capacités sera investiguée à la recherche des besoins qui y sont liés. Le tout sera consigné dans la liste des besoins. Chaque interface, identifié par son numéro dans la matrice des flux, y sera décrit et ses besoins consignés (voir table 4.3).

Sortie

— Liste des besoins.

Notes

En fonction de la documentation disponible et de la taille de l'entreprise, cette étape pourra se révéler très rapide ou au contraire extrêmement lente. Elle est, dans tous les cas, essentielle à l'identification du produit adéquat.

Table 4.4 – Liste des besoins.

	Description	Connectivité	Service interactions	Message processing	Security	Management	QoS
I_1							
I_2							
I_3							
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
I_n							
Globalisation							

4.4.3 A.3 Définition des critères d'évaluation

Entrée

— Liste des besoins

Équipe

— Équipe d'analyse de l'intégration

Activités

— Identifier les besoins critiques et composer le questionnaire correspondant. Un besoin critique est défini comme un besoin qui s'il n'est pas rencontré par un des produits candidats entraine l'élimination automatique de celui-ci lors de la phase de screening.

— Composer le questionnaire d'évaluation. Cette activité est facilitée par la liste des besoins établie à l'étape précédente. Grossièrement, cela revient à poser la question « La solution prend elle en charge le besoin ? » et ce pour chaque besoin.

— Établir l'échelle de cotation qui sera utilisée pour évaluer les différents critères.

Sortie

— Questionnaire de screening.

— Questionnaire d'évaluation.

4.5 Screening

4.5.1 S.1 Etude du marché

Équipe

— Équipe de screening.

Activités

— Rechercher et lister les ESB du marché qu'il conviendra d'évaluer.
— Collecter les données qui faciliteront l'évaluation.

Sortie

— Liste des ESB présents sur le marché.

Notes

Il s'agira lors de cette étape d'identifier les ESB présents sur le marché. Des renseignements comme l'adresse web correspondant à chaque produit, la localisation des documentations officielles ou encore des références de fournisseurs et de personnes de contact pourront aider lors des prochaines étapes.

4.5.2 S.2 Screening

Entrée

— Liste des ESB présents sur le marché
— Questionnaire de screening

Équipe

— Équipe de screening.

Activités

— Activation des sources d'informations.
— Agrégation des données collectées pour chaque ESB.
— Écrémage de la liste d'après les réponses apportées aux critères critiques.

Sortie

— Short liste des candidats ESB.

Notes

Les informations pourront provenir de diverses sources comme : les documentations des produits ; des avis d'experts internes et/ou externes ; les fournisseurs.

4.6 Décision

4.6.1 D.1 Pondération des critères

Entrée

— Modèle de capacités des ESB.

— Questionnaire d'évaluation.

Équipe

— Équipe d'évaluation des critères.

Activités

— Évaluation de la pondération des catégories principales de notre modèle des capacités.

— Découverte d'un consensus de pondération.

Sortie

— Questionnaire d'évaluation pondéré.

Notes

La pondération des critères est un processus qui peut s'avérer complexe a fortiori s'il implique plusieurs intervenants. Elle peut s'effectuer de diverses manières. Nous proposerons deux méthodes (jetons et AHP) qui seront décrites dans la section 4.7.6.

4.6.2 D.2 Collecte d'informations

Entrée

— Short liste des ESB à évaluer.

— Questionnaire d'évaluation.

Équipe

— Équipe d'analyse des produits.

Activités

— Activation des sources d'informations.

— Collecte des questionnaires complétés.

— Agrégation des résultats par source et produit.

Sortie

— Questionnaires d'évaluation complétés (1/ESB/Source).

Notes

Tout comme pour l'étape de screening, les informations pourront provenir de diverses sources comme : les documentations des produits ; des avis d'experts internes et/ou externes ; les fournisseurs. Il conviendra néanmoins de ne pas agréger les questionnaires provenant de sources de nature différente. Il est en effet nécessaire de pouvoir différencier les réponses fournies par un fournisseur de celles fournies par un expert externe ou encore de celles collectées par une recherche documentaire.

Il est ici important de noter que le questionnaire utilisé ne contient pas la pondération des critères. Ceci permet d'éviter une quelconque influence de celle-ci sur l'opération de collecte des informations, en particulier lorsque des fournisseurs sont sollicités et/ou si l'équipe d'analyse des produits et différentes de celle d'évaluation des critères.

4.6.3 D.3 Évaluation

Entrée

— Questionnaires complétés.

— Pondération des critères.

Équipe

— Équipe d'analyse des produits.

Activités

— Évaluation des produits de la short liste à partir des questionnaires complétés et de la pondération des critères.

— Rédaction d'un rapport d'évaluation.

Sortie

— Rapport d'évaluation.

Notes

La mise en parallèle de la pondération des critères et des réponses aux questionnaires permet de réaliser l'évaluation des produits de la short liste. Comme dans l'étape précédente, il convient de différencier les évaluations qui seraient éventuellement basées sur des réponses de sources différentes afin de permettre à l'équipe de décision d'effectuer un choix en toute connaissance de cause.

4.6.4 D.4 Décision

Entrée

— Rapport d'évaluation

Équipe

— Équipe de décision.

Activités

— Étude du rapport d'évaluation.

— Choix du produit le plus adapté.

Sortie

— ESB.

4.7 Mise en application

Afin d'illustrer la méthodologie d'évaluation et de sélection d'ESB proposée, nous allons effectuer une simulation théorique de la méthode sur notre cas d'accompagnement. Pour rappel celui-ci fait référence au système d'information d'un hôpital et contient 19 sous systèmes. Une première étape d'initialisation est effectuée. Elle a permis de composer les différentes équipes. Nous ne nous attarderons pas sur cette étape.

4.7.1 Initialisation

La phase d'initialisation lance officiellement la sélection d'un ESB suite à l'identification d'un besoin d'intégration. Elle est précédée par la désignation d'un chef de projet dont le rôle sera de diriger l'entièreté du processus. Elle implique l'établissement des équipes dont la composition pourra grandement varier en fonction du contexte dans lequel se déroule la sélection : taille, domaine, étendue et organisation de l'entreprise, compétences disponibles et collaborations externes. Nous ne nous attarderons pas davantage sur cette étape.

4.7.2 Analyse de l'intégration

Cette première étape est d'une importance capitale. Elle va permettre de déterminer les besoins que notre futur ESB devra satisfaire. L'équipe d'analyse de l'intégration à pour cela répertorié toutes les interfaces nécessaires dans une matrice de flux. Le résultat de cette analyse est visible dans la table 4.5.

Elle a permis d'identifier 58 interfaces. On peut y constater qu'un certain nombre de ces interfaces sont dédiées à la business intelligence (BI). Elles n'ont pas été prises en compte dans les besoins de notre ESB. En effet, elles seront gérées par un ETL. De même les interfaces liées au système d'identification et d'autorisation

Table 4.5 – Flux d'informations dans le HIS de notre cas compagnon.

de \ vers	AMS	BMS	BI	EPR	HRIS	HSM	IAM	IMS	LIS	MMS	PACS	RIS	PMS	TMS	PIS	GEPR	HIIS	iNAP	NHA
AMS	x		1			2		3	4						5				
BMS		x	6	7		8				9			10		11				
BI			x																
EPR			12	x					13	14		15			16	17			18
HRIS	19		20		x		21							22					
HSM	2		23			x													
IAM	24	25	26	27	21	29	x	30	31	32		33	34	35	36				
IMS	3		37					x											
LIS			38	13				39	x										
MMS			40					41		x									
PACS											x	42							
RIS			43	15				44			42	x				45			
PMS	46	10	47	48				49	50	51		52	x		53		54		
TMS			55		22									x					
PIS	5		56	16				57							x			58	
GEPR				17												x			
HIIS													54				x		
iNAP															28			x	
NHA				18															x

(IAM) sont directement prises en charges par celui-ci. Le nombre d'interfaces que notre ESB doit prendre en charge est dès lors fixé à 33. Celles-ci sont décrites dans la table 4.6. Une étude détaillée de chacune d'elle nous a permis de globaliser nos besoins :

Connectivité La prise en charge des web services SOAP et REST est nécessaire de même qu'un support des protocoles HTTP, HTTPS, SFTP, SMB et MLLP. Les adapteurs suivants seront nécessaires : Bases de données relationnelles ; MSMQ. Enfin il devra être possible de créer nos propres adaptateurs.

Interactions de services Une transparence de localisation statique est jugée suffisante. Il devra être possible de décrire des web services avec WSDL. Les modèles d'interactions Request/Response, One-Way et Publish/Subscribe seront pris en charge. En plus du format XML, JSON, HL7 et DICOM devront être gérés. La transformation sera possible avec Xslt, XQuery, XPath ou un langage de script.

Traitement des messages Il sera possible de router des messages sur base de leur contenu et d'effectuer un filtrage. La re-délivrance de message devra être possible.

Securité Un authentification LDAP sera possible de même que le chiffrement à l'aide de AES.

Gestion Monitoring, Logging et alerting seront possibles. Une console de gestion de type GUI est souhaitée mais non requise.

Qualité de service La fiabilité devra être au rendez-vous. WS-ReliableMessaging devra être pris en charge.

4.7.3 Étude du marché

L'étude du marché a permis de répertorier une liste de 27 produits estampillés ESB. Cette liste est visible sur la table 4.7.

Table 4.6 – Interfaces du système d'information de notre cas d'accompagnement.

nº	Systèmes	Description
2	AMS-HSM	Signalétiques fournisseurs, factures
3	AMS-IMS	Impayés, factures patients
4	AMS-LIS	Signalétiques fournisseurs, factures
5	AMS-PIS	Signalétiques fournisseurs, factures
7	BMS-EPR	Mouvements des patients (transferts, congés, sorties)
8	BMS-HSM	Mouvements des patients (transferts, congés, sorties)
9	BMS-MMS	Mouvements des patients (transferts, congés, sorties)
10	BMS-PMS	Mouvements des patients (transferts, congés, sorties)
11	BMS-PIS	Mouvements des patients (transferts, congés, sorties)
13	EPR-LIS	Demandes et résultats d'analyses
14	EPR-MMS	Régimes et allergies alimentaires
15	EPR-RIS	Demandes d'examens, allergies
16	EPR-PIS	Prescriptions et consommations de médicaments, Signalétiques médicaments
17	EPR-GEPR	Informations médicales pertinentes
18	EPR-NHA	Prescriptions ambulatoires de médicaments, timestamping
19	HRIS-AMS	Liste des agents
21	HRIS-IAM	Liste des agents et mouvements
22	HRIS-TMS	Liste des agents, mouvements, affectations, pointages
39	LIS-IMS	Analyses effectuées
41	MMS-IMS	Consommations de denrées
42	PACS-RIS	Fichiers d'imageries
44	RIS-IMS	Examens effectués
45	RIS-GEPR	Images médicales
46	PMS-AMS	Signalétiques patients
48	PMS-EPR	Admissions de patients
49	PMS-IMS	Signalétiques patients
50	PMS-LIS	Signalétiques patients
51	PMS-MMS	Signalétiques patients
52	PMS-RIS	Signalétiques patients
53	PMS-PIS	Signalétiques patients
54	PMS-HIIS	Assurabilité des patients
57	PIS-IMS	Consommations de médicaments
58	PIS-iNAP	Liste des médicaments

Table 4.7 – Résultats de l'étude du marché des ESB.

AdroitLogic UltraESB-X	Apache ServiceMix
Aurea CX Messenger	Divante ESB
ESBeetle	Fiorano ESB
IBM WebSphere ESB	Intersystems Ensemble
Kovair Omnibus	Micro Focus Artix
Mule ESB	Microsoft BizTalk
Neuron ESB	Nextgen Connect
NServiceBus	Open ESB
Oracle Service Bus	Petals ESB
Red Hat JBoss Fuse	RSSBus Connect
Software AG webMethods	Talend Data Services Platform
Talend Open Studio for ESB	Tibco BusinessWorks
WSO2 ESB	

4.7.4 Définition des critères d'évaluation

A partir des besoins identifiés lors de l'étape d'analyse de l'intégration, l'équipe d'analyse de l'intégration a en premier lieu identifié les besoins critiques. Ils ont ensuite été retranscrit dans un questionnaire spécifique. Un questionnaire général contenant tous les besoins a également été créé.

Questionnaire des critères critiques

Le questionnaire de screening a pour objectif l'élimination rapide des candidats ne remplissant pas les critères critiques définis par l'équipe d'analyse. Elles pourront porter sur les capacités de l'ESB mais également sur d'autres aspects de la sélection comme le prix, la réputation du produit, la réputation du fournisseur, des avis d'experts ou encore la qualité de la documentation fournie. Les critères critiques constituent donc des conditions sine qua non pour qu'un produit passe l'étape de screening. Dans notre exemple, le questionnaire des critères critiques est le suivant :

CC1 Le produit supporte-t-il le format HL7 et le protocole MLLP [7] ? Ce standard de communication est essentiel dans un contexte hospitalier.

CC2 Le produit supporte-t-il DICOM [8] Les raisons sont les mêmes que pour CC1.

CC3 Le produit est-il disponible gratuitement ? Des restrictions budgétaires imposent le choix d'un ESB gratuit.

CC4 Le produit est-il compatible avec les systèmes d'exploitation Microsoft Windows ? Les équipes systèmes ne disposant que de connaissances sur ces plateformes, le produit devra l'être. La direction n'étant pas disposée à envoyer des données médicales sur le cloud, les solutions de ce type seront également écartées.

Questionnaire d'évaluation

Préalablement à la création du questionnaire à partir des besoins, il est nécessaire de définir l'échelle de cotation qui sera utilisée dans celui-ci. Quelle qu'elle soit, cette échelle devra être composée de valeurs numériques progressives. La plus petite valeur indiquera la moins bonne cotation tandis que la valeur la plus grande indiquera la meilleure. Pour la mise en application de notre méthodologie, nous avons défini l'échelle suivante :

0 Le produit ne fournit pas la fonctionnalité ou celle-ci n'est pas documentée.

2 Le produit fournit une partie de la fonctionnalité ou la fonctionnalité n'est pas clairement documentée.

4 Le produit fournit presque complètement la fonctionnalité et elle est clairement documentée.

6 Le produit fournit complètement la fonctionnalité et elle est clairement documentée.

7. Minimal Lower Layer Protocol

8. Digital imaging and communications in medicine. Standard international pour la gestion informatique des données issues de l'imagerie médicale.

Table 4.8 – Questionnaire d'évaluation pour la catégorie connectivité.

Connectivité				Cote
Web services	soap			
	rest			
protocols	http			
	https			
	sftp			
	smb			
	mllp			
adapters	sdk			
	technologies	db	relationnelle	
		messaging	MSMQ	
		file systems	ntfs	
tot				

Il est à noter que l'échelle présentée ici ne favorise pas les produits dont les fonctionnalités sont cotées 6. L'équipe d'analyse pourrait cependant vouloir favoriser les cotations les plus hautes et définir une échelle progressive comme 0,2,6,12. Ceci est laissé à l'appréciation de l'équipe.

Une fois l'échelle définie, il conviendra de construire le questionnaire à partir des besoins. Il comprendra à la fois les capacités minimales de notre modèle des capacités ainsi que celles définies par les besoins. Chaque catégorie principale du modèle sera ainsi questionnée. Un exemple de questionnaire pour la catégorie connectivité est représenté dans la table 4.8.

4.7.5 Screening

À partir des critères critiques définis lors de l'étape d'analyse de l'intégration et de la liste des ESB établie lors de l'étape d'étude du marché, l'équipe de screening a effectué une collecte rapide d'informations concernant uniquement ces critères. Elle a été réalisée uniquement par une recherche d'informations en ligne. Seules les informations provenant des sites des différents éditeurs ont été prises en compte. Si aucune information sur les critères CR1 et CR2 n'y ont été trouvées, il a été considéré que les produits correspondants ne prenaient pas en charge ces critères. Les résultats de cette recherche d'informations sont visibles dans la table 4.9.

Ils montrent clairement que seul un des produits (Nextgen Connect) satisfait au 4 critères critiques définis. Dans un souci de ne pas se limiter à un seul candidat, l'équipe de screening décide d'inclure également les produits satisfaisant les critères critique 1, 3 et 4. L'étape de screening débouche donc sur une liste de 3 produits : Apache ServiceMix ; Nextgen Connect ; WSO2 ESB. De cette manière si Nextgen Connect ne présente pas un bilan d'évaluation suffisamment bon lors de l'étape d'évaluation, une cohabitation de deux ESB pourra être envisagée.

4.7.6 Pondération des critères

Comme indiqué dans la section 4.6.1, la pondération des critères portera sur les catégories principales de notre modèle des capacités. Pour rappel, celles-ci sont :

Table 4.9 – Résultats de l'étape de screening. O = oui / N = non

Produit	CC1	CC2	CC3	CC4
AdroitLogic UltraESB-X	O	N	N	O
Apache ServiceMix	O	N	O	O
Aurea CX Messenger	N	N	N	O
Divante ESB	N	N	N	O
ESBeetle	N	N	N	O
Fiorano ESB	O	N	N	O
IBM WebSphere ESB	O	O	N	O
Intersystems Ensemble	O	O	N	O
Kovair Omnibus	N	N	N	O
Micro Focus Artix	N	N	N	O
Mule ESB	O	N	N	O
Microsoft BizTalk	O	O	N	O
Neuron ESB	N	N	N	O
Nextgen Connect	O	O	O	O
NServiceBus	N	N	N	O
Open ESB	N	N	O	O
Oracle Service Bus	O	O	N	O
Petals ESB	N	N	O	O
Red Hat JBoss Fuse	O	N	N	O
RSSBus Connect	O	N	N	O
Software AG webMethods	O	N	N	O
Talend Data Services Platform	O	N	N	O
Talend Open Studio for ESB	N	N	O	O
Tibco BusinessWorks	O	N	N	O
WSO2 ESB	O	N	O	O

— La connectivité

— Les interactions de services

— Le traitement des messages

— La sécurité

— La gestion

— La qualité de service

La pondération représente l'importance relative accordée à chaque catégorie. Elle est exprimée en pourcentage pour chacune d'elle et leur somme doit être égale à 100%. L'élicitation des valeurs de pondération peut être réalisée selon différentes méthodes. Elle pourrait être décidée de façon arbitraire et plus méthodiquement. Nous présenterons et utiliserons ici deux méthodes de construction de la pondérations des critères.

Méthode des jetons Elle est simple et directe. Elle consiste à attribuer à chaque membre de l'équipe d'évaluation des critères un nombre N de jetons. Chaque membre les répartit ensuite selon ses priorités sur chaque critères. Notre équipe est ici constituée de 4 membres.

Le résultat de la répartition des jetons est visible sur la figure 4.2. Le membre n°1 en bleu a donné une plus grande importance aux aspects le concernant directement (connectivité, interactions de services et traitement des messages). Le n°2 en

Figure 4.2 – Élicitation de la pondération des critères par la méthode des jetons.

jaune a réparti ses jetons sur l'ensemble des catégories en donnant une importance légèrement plus grande à la qualité de service. Le n°3 en vert a adopté un comportement similaire en privilégiant la sécurité. Pour terminer le n°4 a mis l'accent sur la sécurité et dans une moindre mesure sur la gestion. L'équipe a dès lors pu calculer la priorité de chaque catégorie selon la formule 4.1 où PC_i indique la pondération du i^{me} critère, N_i le nombre de jetons attribués au i^{me} critère et N_t le nombre total de jetons attribués. Le calcul de la pondération de chaque critère selon cette méthode est visible dans la table 4.10.

$$PC_i = \frac{100 \times N_i}{N_t} \tag{4.1}$$

Table 4.10 – Pondération des critères selon la méthode des jetons

Critère	Jeton(s)	Pondération
Connectivité	4	16,67%
Interactions de service	3	12,50%
Traitement des messages	3	12,50%
Sécurité	7	29,16%
Gestion	4	16,67%
Qualité de service	3	12,50%

Méthode AHP La méthode d'aide à la décision multicritère AHP propose un processus d'élicitation de la pondération selon une approche plus mathématique et moins directe que la méthode des jetons proposée ci-dessus. Elle reste néanmoins simple à mettre en œuvre et suit les étapes suivantes :

1. Comparer chaque paire de catégorie de critères et définir l'importance de l'un par rapport à l'autre sur une échelle fournie.
2. Reporter les résultats dans une matrice de comparaison.
3. Calculer la matrice de comparaison réciproque.
4. Sommer les colonnes.
5. Normaliser la matrice.

6. Calculer le vecteur de priorité.
7. Vérifier la consistance.

La comparaison de chaque paire de critères est réalisée selon l'échelle suivante :

1 Les deux critères sont d'une importance égale.

3 Un critère est légèrement plus important que l'autre.

5 Un critère est plus important que l'autre.

7 Un critère est beaucoup plus important que l'autre.

9 Un critère est absolument plus important que l'autre.

L'usage de cette échelle n'est pas obligatoire mais recommandée par Thomas L. Saaty, créateur de la méthode AHP [72]. Un exemple de comparaison est donné en figure 4.3. Dans celui-ci, la connectivité est indiquée comme étant légèrement plus importante que le traitement des messages. La méthode implique un certain nombre

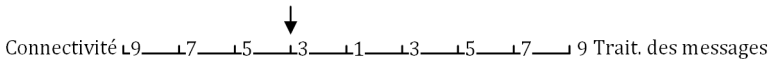


Figure 4.3 – Comparaison d'une paire de critères.

de comparaisons dépendant du nombre de critères n. La formule $\frac{n \times (n-1)}{2}$ permet de calculer ce nombre de comparaisons. Notre modèle des capacités comportant 6 catégories principales de critères, leur pondération nécessitera 15 comparaisons. Le résultat de ces opérations est reporté dans une matrice de comparaison visible dans la table 4.11. On y constate que la connectivité est jugée comme légèrement plus importante que le traitement des messages, la gestion et la qualité de service, d'une importance égale aux interactions de services et comme légèrement moins importante que la sécurité. Les interactions de services sont légèrement plus importantes que le traitement des messages, la gestion et la qualité de service mais légèrement moins importante que la sécurité et ainsi de suite pour le reste de la matrice.

La matrice réciproque de comparaison est ensuite simplement construite en complétant les éléments vides par les valeurs réciproques de la diagonale supérieure. Autrement dit, si E_{ij} est l'élément de la ligne i et de la colonne j alors l'élément E_{ji} dans la diagonale inférieur est égale à

$$E_{ji} = \frac{1}{E_{ij}}$$

Les colonnes sont ensuite sommées et la matrice normalisée en divisant chacun de ses éléments par la somme de la colonne qui lui correspond. La somme d'une

Table 4.11 – Matrice de comparaison.

	Con	IdS	TdM	Sec	Ges	QoS
Con	1	1	3	1/3	3	3
IdS		1	3	1/3	3	3
TdM			1	1/5	1/3	3
Sec				1	5	5
Ges					1	1
QoS						1

colonne de la matrice normalisée est dès lors égale à 1. Le vecteur de priorité et donc le poids de chacun de nos critères peut alors être calculé en effectuant une moyenne sur chaque ligne de la matrice. Les résultats sont visibles dans les tables 4.12 et 4.13

Pour terminer, il nous reste à vérifier la consistance de notre pondération. Elle est indispensable car le processus de comparaison est réalisé par des humains. Or l'humain est ainsi fait qu'il peut parfois être incohérent ou inconsistant. Un simple exemple permettra de mieux comprendre ce concept. Imaginons que notre équipe d'évaluation indique que la sécurité est plus importante que la connectivité et que la connectivité l'est plus que la qualité de service. Si ensuite la même équipe lorsqu'elle compare la sécurité et la qualité de service indique que cette dernière est plus importante, une incohérence est introduite. On peut en effet déduire logiquement des deux premières comparaisons que la sécurité est plus importante que la qualité de service. Saaty propose de calculer cette incohérence à travers trois valeurs : la valeur propre principale VPP ; l'index de consistance CI ; le ratio de consistance CR. La valeur propre principale est obtenue par la somme des produits entre chaque élément du vecteur propre et la somme des colonnes de la matrice réciproque :

$$VPP = 6 \times 0,18 + 6 \times 0,18 + \frac{46}{3} \times 0,08 + \frac{34}{15} \times 0,41 + \frac{37}{3} \times 0,09 + 16 \times 0,06 = 6,4631$$

L'index de consistance est calculé d'après la valeur propre principale et le nombre de critères :

$$CI = \frac{VPP - n}{n - 1} = \frac{6,4631 - 6}{5} = 0,0926$$

Enfin le ratio de cohérence peut être calculé à partir du CI et d'un index aléatoire de consistance, dépendant de la taille de la matrice et déterminé par Saaty. Il est donné par la table 4.14 en fonction du nombre de critère n :

$$CR = \frac{CI}{RI} = \frac{0,0926}{1,24} = 0,0747 \backsimeq 7\%$$

Saaty a suggéré un seuil de 10% pour les valeurs de RC. Il précise que ce seuil est analogue à un niveau significatif d'analyse statistique se rapportant au niveau de confiance de l'analyse plutôt qu'à une valeur fixe immuable [68]. Simplement dit, il s'agit donc avant tout d'une valeur indicative. Si une valeur trop élevée est observée, il sera cependant de bon ton de reconsidérer les comparaisons. Notre ratio de cohérence étant aux alentours de 7%, nous sommes largement sous le seuil des 10% recommandé par Saaty. Notre pondération est donc cohérente.

Table 4.12 – matrice réciproque de comparaison.

	Con	IdS	TdM	Sec	Ges	QoS
Con	1	1	3	1/3	3	3
IdS	1	1	3	1/3	3	3
TdM	1/3	1/3	1	1/3	1/3	3
Sec	3	3	5	1	5	5
Ges	1/3	1/3	3	1/5	1	1
QoS	1/3	1/3	1/3	1/5	1	1
	6	6	46/3	34/15	37/3	16

TABLE 4.13 – matrice normalisée et poids correspondants.

	Con	IdS	TdM	Sec	Ges	QoS	Pd	%
Con	1/6	1/6	9/46	5/34	9/40	3/16	0,18	18%
IdS	1/6	1/6	9/46	5/34	9/40	3/16	0,18	18%
TdM	1/18	1/18	3/46	3/34	1/40	3/16	0,08	8%
Sec	1/2	1/2	15/46	15/34	3/8	5/16	0,41	41%
Ges	1/18	1/18	9/46	3/34	3/40	1/16	0,09	9%
QoS	1/18	1/18	1/46	3/34	3/40	1/16	0,06	6%
	1	1	1	1	1	1	1	100%

TABLE 4.14 – Table des index aléatoires de consistance.

n	1	2	3	4	5	6	7	8	9	10
RI	0	0	0,58	0,9	1,12	1,24	1,32	1,41	1,45	1,49

4.7.7 Collecte d'informations

Parallèlement à la définition de la pondération des critères, la collecte d'informations sur les produits de la short liste peut s'effectuer. Dans notre mise en application, cette collecte a été réalisée en suivant une méthodologie en 2 phases. En premier la documentation de chaque produit de la short liste a été consultée. Si aucune information utile concernant un critère n'a pu être identifiée dans la documentation du produit, une recherche via Google a été conduite dans un second temps. Dans ce cas de figure, un critère ne pourra recevoir qu'une cote de 0 ou 2 selon l'échelle définie dans la section 4.7.4. Les documentations consultées pour chaque produit sont présentées dans la table 4.15

4.7.8 Évaluation

A ce stade, les catégories principales de critères ont été pondérées selon deux méthodes et les questionnaires d'évaluations ont été complétés grâce à la collecte d'informations. Le score de chaque produit est visible dans les tables 4.16, 4.17, 4.18, 4.19, 4.20 et 4.21 respectivement pour les catégories de la connectivité, des interactions de services, de traitement des messages, de la sécurité, de la gestion et enfin de la qualité de service. Chaque table présente pour chaque produit son score dans la catégorie concernée. Le résultat final de l'évaluation est présenté dans la table 4.22. Il comporte deux parties. La partie supérieure résume pour chaque produit son score dans chaque catégorie. La colonne S_{norm} présentent la normalisation sur 100 de ces scores. La normalisation du score de chaque catégorie de critères

TABLE 4.15 – Sources des documentations officielles des ESB évalués.

Produit		Documenation
A	Apache ServiceMix	`http://servicemix.apache.org/docs/7.x/index.html`
B	Nextgen Connect	`https://www.nextgen.com/-/media/Files/nextgen-connect/nextgen-connect-38-user-guide.pdf`
C	WSO2	`https://pdf-docs.wso2.com/EI650-070819-0256-2910.pdf`

Table 4.16 – Questionnaire d'évaluation de la catégorie connectivité.

Connectivité				**ESB**		
				A	**B**	**C**
Web services	soap			6	6	6
	rest			6	2	6
protocols	http			6	6	6
	https			6	6	6
	sftp			6	6	6
	smb			6	6	6
	mllp			4	6	6
adapters	sdk			0	0	6
	technologies	db	relationnelle	6	6	6
		messaging	MSMQ	2	0	6
			tot	52	50	60

Table 4.17 – Questionnaire d'évaluation de la catégorie interactions de services.

Interactions de services			**ESB**		
			A	**B**	**C**
Interface definition	wsdl		6	6	6
MEPs	Request/Response		6	6	6
	One-Way		6	6	6
	Publish-Subscribe		6	2	6
Data	Formats	XML	6	6	6
		JSON	6	6	6
		HL7	4	6	6
		DICOM	0	6	0
	Transformation	Xslt	6	6	6
		XQuery	6	0	6
		XPath	6	2	6
		Script based	2	6	6
Location transp.	Static		2	6	6
		tot	62	64	72

permet de s'assurer que le résultat final dépendra uniquement de la pondération définie. La partie inférieure indique pour chaque produit dans les colonnes S_J et S_{AHP} les scores pondérés de chaque catégorie selon les poids déterminés respectivement par la méthode des jetons et la méthode AHP. Les deux dernières lignes indiquent le score total de chaque produit selon chaque méthode de pondération ainsi que le rang obtenu.

4.7.9 Décision

Le rapport d'évaluation final présenté dans la table 4.22 laisse clairement apparaître la hiérarchie des produits selon les critères et la pondération établis lors des étapes 4.7.4 et 4.7.6. Le produit C (WSO2 ESB) se détache clairement de ses deux concurrents. Cependant comme nous l'avons mentionné en conclusion de l'étape de screening (4.7.5), ce produit ne prend pas en charge le protocole DICOM qui était un critère critique. Seul le produit B (Nextgen Connect) en est capable. Une décision

Table 4.18 – Questionnaire d'évaluation de la catégorie traitement des messages.

Traitement des messages		**ESB**		
		A	**B**	**C**
Routing	Fixed	6	6	6
	Content-based	6	6	6
	Filtering	6	6	6
Transformation	Enrichment	6	2	6
	Splitting	6	6	6
	Aggregation	6	2	6
	Filtering	6	6	6
	Resequencing	6	2	6
Redilevery		6	6	6
	tot	54	42	54

Table 4.19 – Questionnaire d'évaluation de la catégorie sécurité.

Sécurité		**ESB**		
		A	**B**	**C**
Authentication & authorization	LDAP	4	6	6
Message confidentiality	AES	2	6	6
	tot	6	12	12

Table 4.20 – Questionnaire d'évaluation de la catégorie gestion.

Gestion		**ESB**		
		A	**B**	**C**
Monitoring		2	6	6
Logging		2	6	6
Alerting		2	4	2
Tools	Cli	6	4	6
	GUI	0	6	6
	tot	12	26	26

Table 4.21 – Questionnaire d'évaluation de la catégorie qualité de service.

Qualité de service		**ESB**		
		A	**B**	**C**
Reliability	WS-ReliableMessaging	6	2	6
	tot	6	2	6

TABLE 4.22 – Rapport final de l'évaluation de la short liste des ESB.

	A		B		C		
	S	S_{norm}	S	S_{norm}	S	S_{norm}	P_J
Con	48	80,00	44	73,33	66	100,00	16,67%
IdS	62	79,49	56	71,79	72	92,31	12,50%
TdM	54	100,00	42	77,78	54	100,00	12,50%
Sec	6	50,00	12	100,00	12	100,00	29,16%
Ges	12	40,00	26	86,67	26	86,67	16,67%
QoS	6	100,00	2	33,33	6	100,00	15,50%
	S_J	S_{AHP}	S_J	S_{AHP}	S_J	S_{AHP}	P_{AHP}
Con	13,34	14,40	12,22	13,20	16,67	18,00	18,00%
IdS	9,94	14,31	8,97	12,92	11,54	16,62	18,00%
TdM	12,50	8,00	9,72	6,22	12,50	8,00	8,00%
Sec	14,58	20,50	29,16	41,00	29,16	41,00	41,00%
Ges	6,67	3,60	14,45	7,80	14,45	7,80	9,00%
QoS	15,50	6,00	5,17	2,00	15,50	6,00	6,00%
Total	72,52	66,81	79,69	83,14	99,82	97,42	
Rang	3	3	2	2	1	1	

d'adopter WSO2 ESB et Nextgen Connect respectivement dans un contexte général et dans un contexte spécifique à DICOM devrait être prise. Il est également à noter que les deux méthodes de pondération des critères qui ont été expérimentées débouchent sur une même hiérarchie de produit.

4.8 Résumé

Dans ce chapitre nous avons exploré les méthodes d'évaluation et de sélection spécifiquement destinées aux ESB. Nous avons vu qu'elles étaient très peu nombreuses et avons donc étendu nos recherches aux méthodes générales d'évaluation et de sélection de logiciels. Nous avons ensuite proposé une méthode ASD d'évaluation et de sélection des ESB composées de 9 étapes réparties en 3 phases :

Phase A : analyse A.1 Initialisation : constitution des équipes.

A.2 Analyse de l'intégration : déduction des besoins.

A.3 Définition des critères d'évaluation : création des questionnaires de screening et d'évaluation.

Phase S : screening S.1 Étude du marché : répertorier les ESB existants.

S.2 Screening : trier les ESB sur base de critères critiques et créer une short liste des ESB.

Phase D : décision D.1 Pondération des critères : définir l'importance des 6 catégories de critères.

D.2 Collecte d'informations : Répondre au questionnaire d'évaluation.

D.3 Évaluation : Appliquer la pondération au questionnaire d'évaluation.

D.4 Décision : choisir un ESB à partir des résultats de l'étape précédente.

Enfin nous avons mis en application notre méthode à notre cas d'accompagnement afin de choisir l'ESB le plus adéquat à son contexte. Nous sommes arrivés à la conclusion que deux ESB seront nécessaires pour couvrir ses besoins, Nextgen Connect et WSO2 ESB.

Chapitre 5

Étude de cas

Dans ce travail, nous avons débuté par brosser un paysage de l'intégration en mettant l'accent sur l'intégration d'applications. Nous l'avons décrite et avons exploré les styles et architectures qui s'y appliquent. Ensuite, nous avons approfondi l'architecture ESB en détail et nous avons élaboré un modèle de ses capacités sous forme d'un diagramme de fonctionnalités. Ce modèle constitue un support important à la méthodologie d'évaluation et de sélection d'ESB que nous proposons dans le chapitre précédent. Pour terminer, nous allons maintenant illustrer l'usage d'un ESB sur quelques use cases inspirés par notre cas d'accompagnement. Nous utiliserons pour cela l'ESB sélectionné lors de la mise en œuvre de notre méthodologie ASD dans le précédent chapitre.

5.1 WSO2 ESB

WSO2 ESB est un ESB open-source créé est maintenu par la société WSO2. Cette dernière, dont le siège social est situé à Moutain View en Californie, a été fondée en 2006. Elle est spécialisée dans le domaine du middleware et offre une plateforme d'intégration nommée « Enterprise Integration » dont WSO2 ESB est un des composants. Contrairement à beaucoup d'autres produits construits à partir d'EAI reconditionnés, WSO2 ESB a été conçu et développé à partir de zéro et offre une large gamme de capacités d'intégration et un support de routage de messages performant en utilisant un moteur de médiation de messages amélioré et optimisé, basé sur Apache Synapse [81].

5.1.1 Architecture de WSO2 ESB

L'architecture de WSO2 ESB est une architecture basée sur les composants. Ceux-ci sont représentés sur la figure 5.1 qui explique le fonctionnement de cet ESB.

La couche de transport est chargée de l'envoi et du rapport des messages dans plusieurs formats. De nouveaux types de transports peuvent être ajoutés en tant que plugins. Chaque transport utilise un service pour écouter les messages entrants et un autre pour envoyer les réponses.

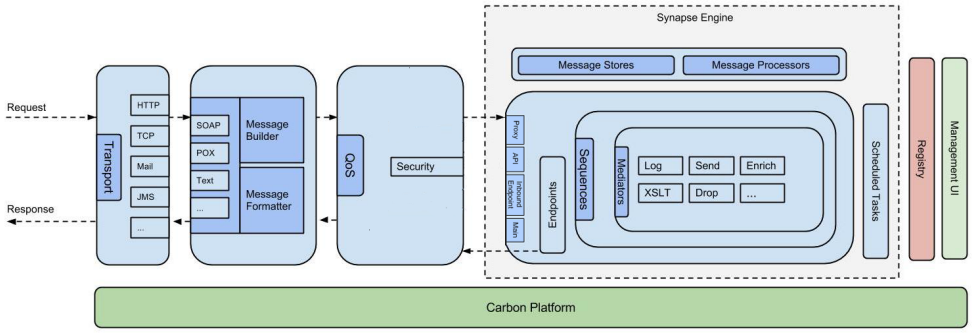


FIGURE 5.1 – Architecture de WSO2 ESB [81].

La couche de formatage et de construction est responsable en fonction de la valeur du 'content type' contenu dans le header :

— de la transformation, par un *message builder*, d'un message entrant en XML. XML est le seul langage que le moteur de médiation comprend.

— de la transformation, par un *message formatter*, d'un message sortant dans le format attendu par le client.

La couche QoS (Quality of Service) ajoute la sécurité au service proxy. Pour ce faire, un fichier de stratégie de sécurité XML est appliqué au service Web.

Le moteur Synapse ESB qui comprend les composants suivants :

— Proxy : il s'agit d'un service virtuel dans ESB qui reçoit et traite les messages. Il les distribue ensuite à un point de sortie en dehors de l'ESB.

— API : Un API dans WSO2 ESB est analogue à une application Web déployée dans le runtime ESB. Chaque API est associé à un contexte d'URL défini par l'utilisateur et ne traitera que les demandes relevant de son contexte URL. Cette approche permet d'envoyer des messages directement dans ESB à l'aide de requêtes REST.

— Points d'entrée : ils peuvent être configurés de manière dynamique sans redémarrer le serveur. Les messages passent de la couche de transport à la couche de médiation sans passer par le moteur Axis2.

— Séquences : une séquence est un ensemble de médiateurs où les messages sont traités.

— Médiateur : c'est l'unité de traitement où une action est effectuée sur un message.

— Tâches planifiées : Il s'agit d'un code à exécuter à un moment donné.

— Points de sortie : Ce sont des destinations de message. Elles sont souvent externes à l'ESB. Il pourra s'agir d'un service représenté par une URL, une boîte mail, une file d'attente JMS ou encore un socket TCP.

— Magasin de messages & Processeurs de messages : Ils sont utilisés lors du traitement asynchrone des messages, autrement dit lorsque le client n'attend

pas la réponse. Le message est dans ce cas stocké par le magasin de messages. Le processeur de messages en extrait une file d'attente et envoie les messages à un point de sortie. En utilisant ce modèle, la livraison d'un message à un point final peut être garantie, car il n'est supprimé du magasin que lorsque le point final reçoit correctement le message.

Le registre est un conteneur de méta-données dans lequel des ressources telles que des fichiers WSDL, des schémas de message pour la validation, des scripts, des fichiers XSLT, des transformations Xquery, sont stockées. Ces informations peuvent être utilisées en les référençant dans les médiateurs. C'est un moyen de mieux organiser l'information au sein de l'ESB.

L'interface utilisateur de gestion est une interface utilisateur graphique dans laquelle tous les composants peuvent être configurés via des menus et des options.

La plateforme Carbon est un middleware open-source sur lequel tous les autres composants WSO2 sont construits.

Dans la suite de cette section, nous allons nous intéresser à quelques concepts clés permettant de comprendre le fonctionnement d'un scénario d'intégration dans WSO2 ESB. Nous présenterons également brièvement l'outil graphique de développement de WSO2 ESB : « Integration Studio ».

5.1.2 Concepts clés

Afin de comprendre le cheminement d'un flux de médiation, nous allons nous intéresser à quelques concepts clés de l'architecture que nous venons de décrire. Pour illustrer ces concepts clés, nous allons nous aider d'un exemple : un hôpital dispose d'une application de gestion d'agenda médical qui permet en autre la prise de rendez-vous et expose ses fonctionnalités sous forme de web services SOAP. L'hôpital souhaite permettre à ses patients de prendre rendez-vous grâce à une application mobile mais celle-ci utilise des requêtes REST/Json. Le problème est donc d'établir une communication entre une application mobile utilisant JSon et REST à un web service SOAP sans intervenir ni sur l'un, ni sur l'autre. Une vue conceptuelle du problème est illustrée en figure 5.2. On y constate que WSO2 ESB accepte les requêtes HTTP/JSON de l'application mobile, transforme le message JSON en message SOAP, envoie ce message au web service de l'agenda et effectue la conversion inverse pour envoyer le réponse à l'application mobile. Voyons à présent plus en détails quels sont les composants principaux intervenant dans ce scénario. Ceux-ci sont illustrés sur la figure 5.3.

Point d'entrée Un point d'entrée est le composant responsable de la gestion du transfert d'un message du système externe (l'application mobile) vers l'ESB. Il existe divers types de point d'entrée dans WSO2 ESB :

— Service proxy : Service virtuel permettant d'exposer un service d'arrière plan sous la forme d'un web service. Il s'agira souvent d'un web service SOAP/HTTP(s).

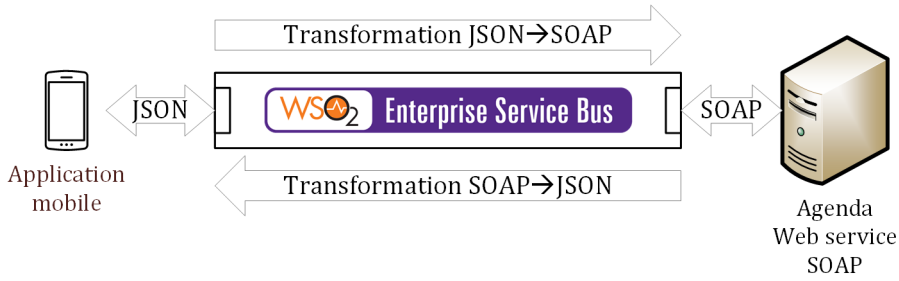


Figure 5.2 – Concept d'un scénario d'intégration dans WSO2 ESB.

— REST API : Service virtuel permettant d'exposer un service d'arrière plan sous la forme d'un interface HTTP conforme aux web services REST.

— Terminal entrant : Source de messages avec des capacité d'écoute ou de polling.

Un message arrivant sur un point d'entrée est dirigé vers une séquence.

Séquence et mediateurs Le traitement des messages arrivant dans l'ESB est réalisé dans un composant appelé séquence. Une séquence est elle même composée de médiateurs. Un médiateur est l'unité de traitement des messages de base et un élément fondamental des ESB. Un médiateur reçoit un message, effectue des actions sur ce message et le passe ensuite au médiateur suivant. Il existe de nombreux types de médiateur. Un premier type apparaît dans l'illustration 5.3, il s'agit du « call ». Il permet d'envoyer un message à un point de sortie.

WSO2 ESB contient une séquence particulière dite séquence de faute. Elle permet de gérer les erreurs qui pourraient intervenir dans le flux de traitement d'un message transitant sur le bus.

Point de sortie Le point de sortie est une représentation logique d'un service externe. Dans notre exemple, l'adresse du web service agenda est configurée comme un point de sortie par lequel un message est acheminé vers ce service via un médiateur d'appel.

5.1.3 Integration Studio

Nous connaissons à présent les concepts qui vont nous permettre de mettre en place nos flux de médiation. WSO2 ESB offre un environnement de développement graphique permettant de concevoir et développer les scénarios d'intégration. Cette environnement est basé sur Eclipse et se nomme « Integration Studio ». Il peut être utilisé aussi bien pour développer des services, des fonctionnalités ou des composants que pour gérer leurs liens et dépendances. La fenêtre principale d'Integration Studio est fortement similaire à celle d'Eclipse et peut être divisée en 4 zones distinctes visibles sur la figure 5.4 :

Explorateur de solution L'explorateur de solution, marqué 1, présente la structure des répertoires et fichiers du projet d'intégration. On peut déjà y retrouver

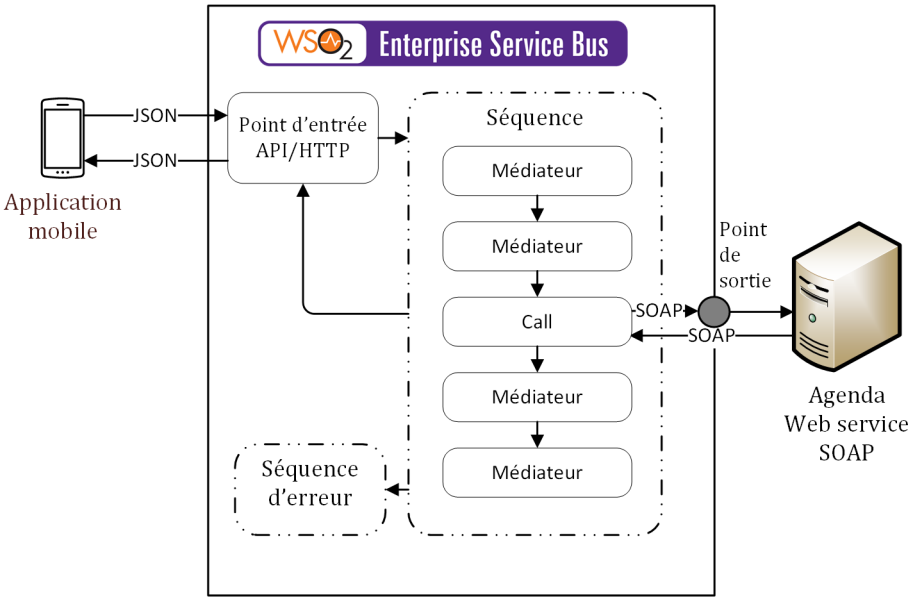


Figure 5.3 – Concept d'un scénario d'intégration dans WSO2 ESB.

quelques répertoires faisant référence à des notions qui nous sont maintenant familières comme : api ; endpoints ; proxy-services ; sequences.

Editeur ESB La zone 2 montre la fenêtre d'édition de l'ESB. Elle permet de configurer le scénario d'intégration par simple glisser coller des médiateurs qui sont présentés sur la gauche de la zone.

La fenêtre structure Elle affiche la structure de l'élément actuellement ouvert dans la zone d'édition et répertorie les éléments structurels. Elle permet de masquer certains champs, méthodes et types, ainsi que de trier et filtrer afin de trouver une information rapidement. Elle correspond à la zone 3.

Les propriétés et la console Cette fenêtre, visible en zone 4, comprend deux onglets. L'onglet propriétés permet de visualiser et modifier les propriétés de l'élément sélectionné dans la zone d'édition. L'onglet console permet l'affichage de trois types de console dans Integration Studio :

- La console de processus affiche les sorties normales et d'erreurs.
- La pile d'appels montre l'état de la pile d'appels Java avec des liens vers des emplacements spécifiques de code source.
- La console CVS [1] affiche le résultat des opérations CVS.

La création d'un projet dans Integration Studio est similaire à celle de tout projet de développement. Il suffit de passer par le menu File -> New -> ESB Solution Project. Il est naturellement nécessaire de choisir un nom pour la solution d'intégration et ensuite de sélectionner le(s) type(s) de projet que la solution devra contenir. Ces types sont les suivants :

Application composite ce type de projet permet de packager tous les éléments de la solution dans une application composite déployable sur un serveur ESB.

1. Concurrent Versions System

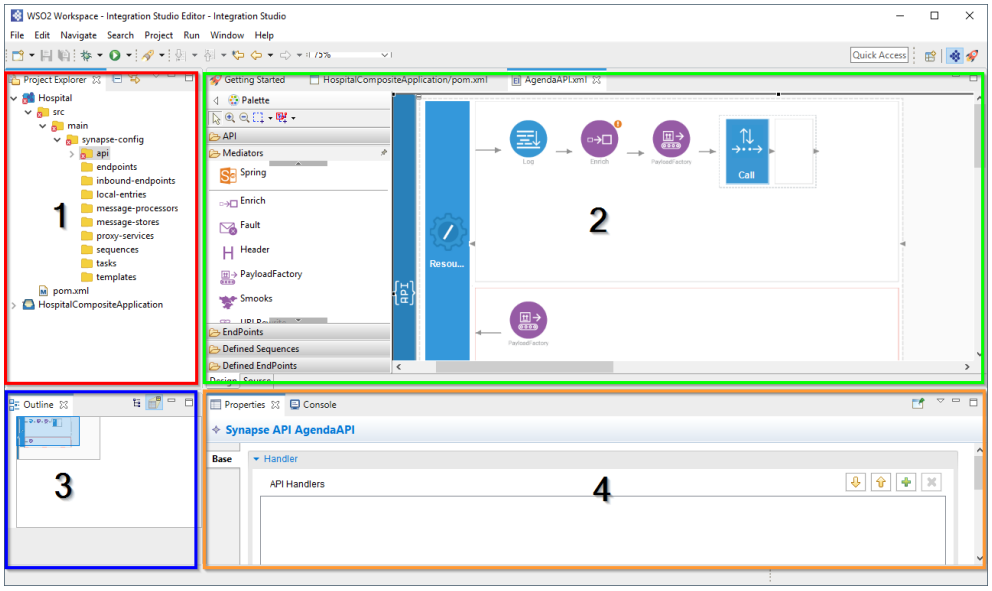


Figure 5.4 – Environnement de développement « WSO2 Integration Studio ».

Registre de ressources Il permet de créer des ressources disponibles dans un registre et qui seront utilisables dans les flux de médiations.

Connecteurs Ce type de projet est nécessaire si le flux de médiation utilise des connecteurs. Chacun d'eux doit être stocké dans le projet pour être packagé dans l'application composite.

La fenêtre de création d'une solution d'intégration est visible sur la figure 5.5. Nous avons nommé notre solution « healthBus ». Nous n'utiliserons pas de connecteurs dans les cas que nous allons développer, ce type de projet n'a donc pas été sélectionné. Si nécessaire, il peut toutefois être ajouté à posteriori. Afin d'illustrer la construction de flux de médiation dans WSO2 ESB, nous développerons 2 cas d'utilisation :

Agenda médical Il s'agit du cas que nous avons décrit pour introduire les concepts clés des flux de médiation dans WSO2 ESB. Pour rappel, il est composé d'un service d'agenda offrant une interface de type web service SOAP. Nous y intégrerons une application mobile cliente faisant usage de requête REST.

Flux patients Ce cas, dans lequel nous illustrerons la transmission des informations des flux de patients, nous permettra de démontrer l'usage d'un point d'entrée de type terminal, le routage et à nouveau la transformation de protocoles et de messages.

5.2 Cas 1 : L'agenda médical

Ce premier cas est celui que nous avons déjà expliqué dans la section 5.1.2 de ce chapitre. Pour rappel, un web service SOAP expose des fonctionnalités de gestion d'un agenda médical. Le problème qui se pose est d'y intégrer une application mobile

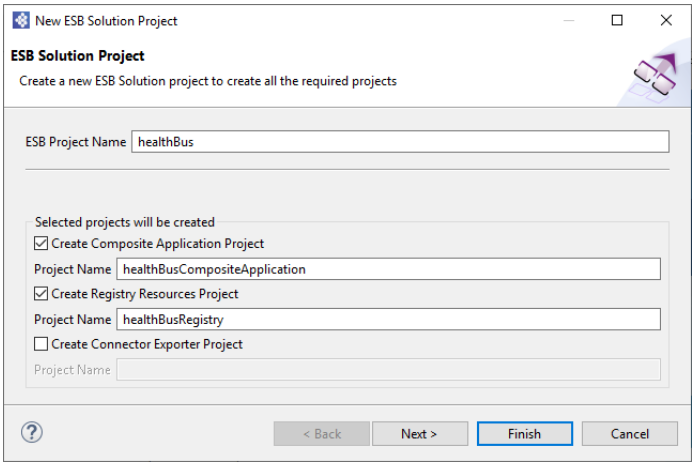


Figure 5.5 – Fenêtre de création d'une nouvelle solution d'intégration.

de réservation de rendez-vous utilisant une interface REST. Nous illustrerons cette intégration à travers trois opérations :

1. L'application mobile demande une liste des spécialités médicales disponibles.
2. L'application mobile demande les prochains rendez-vous disponibles pour une spécialité.
3. L'application mobile réserve un rendez-vous.

Nous avons déjà créer notre projet dans la section 5.1.3. La figure 5.6 nous montre le contenu de notre projet dans l'explorateur de solution. Plusieurs dossiers y sont présentés. Dans ce chapitre nous travaillerons avec les suivants :

— api : Contient les points d'entrée de type REST.

— endpoints : Contient les points de sortie qui pour rappel sont des représentations logiques des services externes qui participent aux flux de médiation.

— inbound-endpoints : Contient les points d'entrée de type terminal entrant. Ils possèdent des capacités d'écoute ou de polling.

— proxy-services : Contient les points d'entrée des web services SOAP.

— sequences : Contient les séquences de médiation qui seront utilisées dans les flux de médiation. Ces séquences peuvent être créés dans ce dossier ou directement dans les flux de médiation.

5.2.1 Création du point de sortie

Avant de débuter la mise en place de la médiation de nos trois opérations, la première chose à faire est de déclarer notre service externe d'agenda. Pour cela, nous ouvrons le menu contextuel du dossier *endpoints* et choisissons de créer un nouveau endpoint. Nous baptisons notre endpoint « wsAgenda », choisissons un endpoint de type *WSDL Endpoint* et configurons celui-ci comme indiqué sur la figure 5.7. Notre service agenda est dès lors connu de l'ESB. Pour les besoins de ce travail, un web service SOAP a été créé pour simuler le service Agenda. Il a été hébergé sur

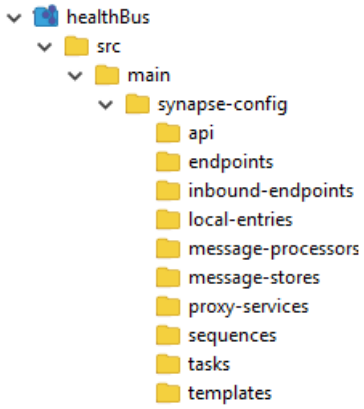


Figure 5.6 – Explorateur de solution de WSO2 Integration Studio.

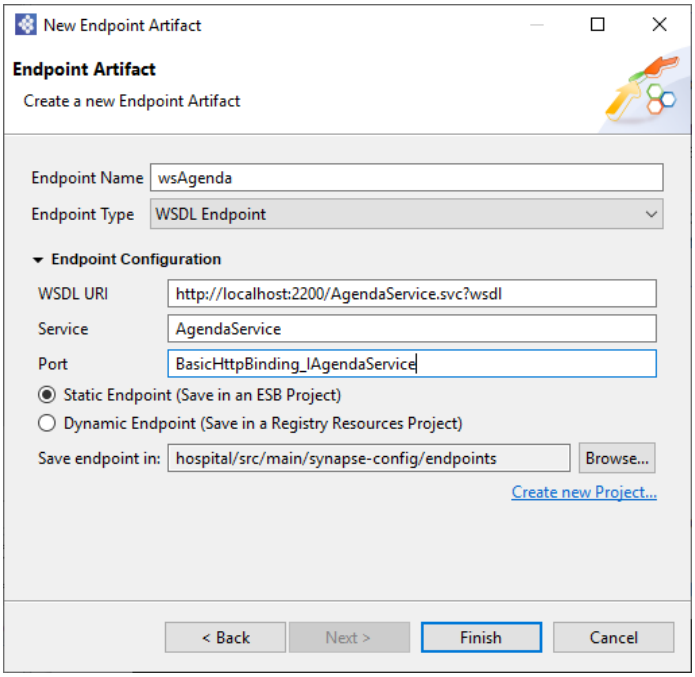


Figure 5.7 – Création du point de sortie correspondant au web service agenda.

la machine de développement ce qui explique l'adresse `http://localhost:2200/AgendaService.svc?wsdl`.

5.2.2 Création du point d'entrée

Nous disposons à présent d'un point de sortie vers notre service Agenda. Il nous reste dès lors à créer le point d'entrée de type *REST API* qui permettra à l'application mobile d'envoyer ses requêtes REST. Ceci est réalisé via le menu contextuel du dossier api de l'explorateur de solution dans lequel nous choisissons de créer

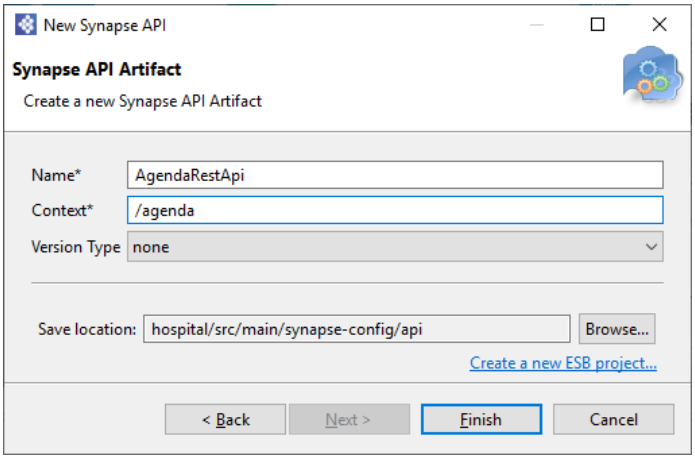


Figure 5.8 – Création du point d'entrée AgendaRestApi.

un nouveau *REST API*. Il consiste en un contexte et un ensemble de ressources identifiées par des URI et accessibles par des verbes HTTP. Pour notre application mobile, nous aurons besoin de deux ressources et trois opérations au total :

- GET /agenda/specialties : qui fournira la liste des spécialités disponibles pour un rendez-vous ;
- GET /agenda/specialties/{specialty}/appointments : qui fournira la liste des prochains rendez-vous pour une spécialité ;
- PUT /agenda/specialties/{specialty}/appointments : qui effectuera une réservation pour un rendez-vous. La requête sera accompagnée d'un payload JSon indiquant l'id du rendez-vous ainsi que l'id du patient qui réserve ce rendez-vous.

La création de notre point d'entrée se réalise via le menu contextuel du dossier *api*. La fenêtre permettant cette création est visible sur la figure 5.8. Nous y constatons que la création est aisée puisqu'il suffit de choisir un nom à notre API ainsi qu'un contexte. Le contexte détermine l'adresse de base à laquelle l'API sera accessible. Dans notre cas nous avons choisi `/agenda`.

5.2.3 Création du flux de médiation

Nous disposons à présent de tous les éléments nécessaires à la réalisation de notre flux de médiation entre notre application mobile REST/JSon et notre web service SOAP agenda. Un flux de médiation dans WSO2 ESB se configure plus qu'il ne se programme. Integration Studio permet pour cela la conception graphique du flux de médiation comme illustré sur la figure 5.9. La conception du flux de médiation est répartie sur 3 zones :

Répertoire des médiateurs Numérotée 1 et située en haut à gauche, cette zone contient les médiateurs qui serviront lors de la conception du flux. Ils seront simplement glissés dans l'espace de conception.

Espace de conception Zone de conception du flux de médiation, elle fournit une vue graphique complète de celui-ci. Elle permet également la visualisation du

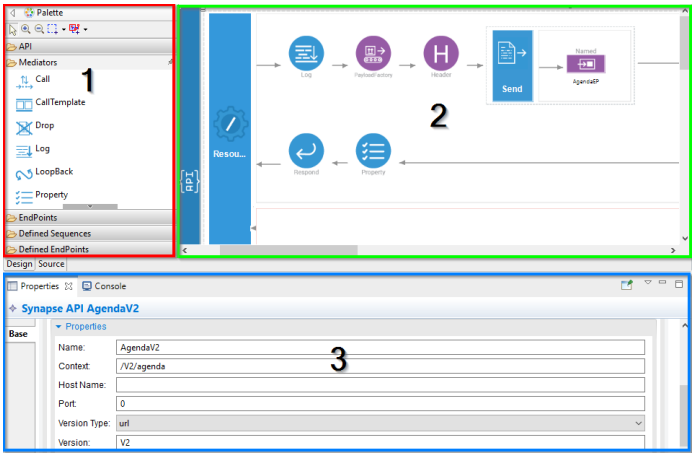


FIGURE 5.9 – Concepteur graphique de médiation de WSO2 ESB.

code source XML du flux. Il est donc également possible d'écrire le flux plutôt que de le configurer. Elle porte le numéro 2 sur notre figure.

Zone des propriétés Elle permet de visualiser et définir les propriétés du médiateur sélectionné dans l'espace de conception. Elle est numérotée 3.

La figure 5.10 présente le résultat final du flux de médiation de l'agenda médical. L'entièreté du flux a été réalisée par de simple glisser/déposer des médiateurs de la zone 1 vers la zone 2 et par la configuration de leurs propriétés dans la zone 3. Tous les artefacts créés dans Integration Studio sont représentés au format XML. Nous décrirons les différents flux de médiation créés à partir du listing de leurs représentations XML. Ceux-ci permettent une meilleure compréhension. Pour ce cas pratique, l'ensemble des flux est configuré dans un seul et même fichier XML. Son aspect dans l'éditeur ESB est visible sur la figure 5.10. Nous avons séparé ce listing en plusieurs sections afin de fournir au lecteur une immersion progressive dans celui-ci. Dans chacune de ces sections nous avons indiqué des points d'attention numérotés sous forme de commentaires XML. Une description les accompagnera.

Création de l'API REST et de ses ressources La création de notre point d'entrée fourni le squelette de nos flux de médiation. Il est présenté sur le listing 5.1 dont la description suit.

Listing 5.1 – Déclaration des ressources de l'API REST de l'application mobile.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- [1] -->
<api xmlns="http://ws.apache.org/ns/synapse" context="/agenda" name="
    AgendaRestApi">
  <!-- [2] -->
  <resource methods="GET" url-mapping="/specialties">
    <!-- sequence -->
  </resource>

  <!-- [3] -->
  <resource methods="PUT GET" uri-template="/specialties/{specialty}/
      appointments">
```

```
    <!-- sequence -->
  </resource>
</api>
```

[1] L'API est déclarée et accessible via le contexte `/agenda`.

[2] Une première ressource accessible via un GET sur l'url `/agenda/specialties` est déclarée.

[3] Une seconde ressource accessible via un GET ou un PUT sur l'url `/specialties/{specialty}/appointments` est déclarée. On notera qu'elle contient un paramètre `{specialty}`.

Obtention de la liste des spécialités La première opération que l'application cliente souhaite réaliser est d'obtenir la liste des spécialités médicales dispensées par l'institution. Elle effectue pour cela une requête GET sur la ressource identifiée par l'url `/agenda/specialties`. L'ESB se charge de convertir cette requête REST dans la requête SOAP requise par le web service agenda, et réalise l'opération inverse pour fournir le résultat à l'application cliente. Le listing 5.2 détaille le flux établi.

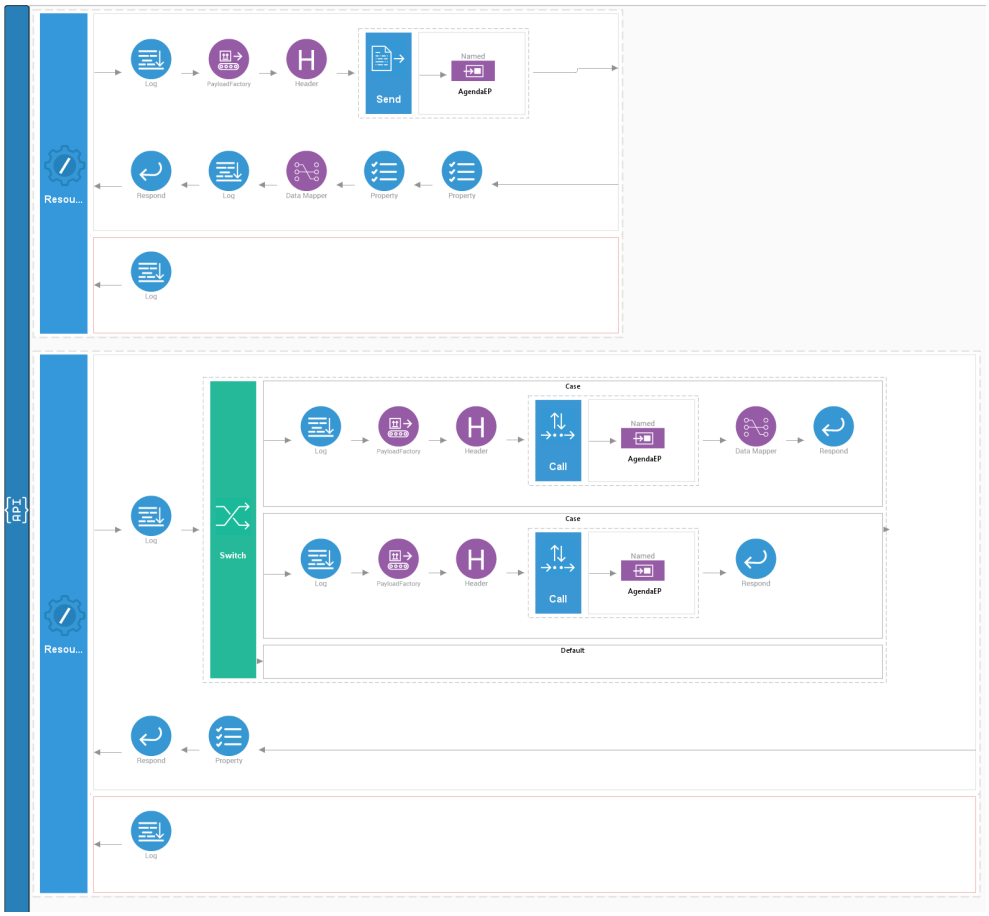


Figure 5.10 – Aspect graphique du flux de médiation entre l'application mobile et le web service agenda.

Listing 5.2 – Flux de médiation pour l'obtention de la liste des spécialités

```
<!-- [2.1] -->
<inSequence>
  <!-- [2.2] -->
  <log />
  <!-- [2.3] -->
  <payloadFactory media-type="xml">
    <format>
      <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
          envelope/" xmlns:tem="http://tempuri.org/">
        <soapenv:Header />
        <soapenv:Body>
          <tem:GetSpecialties />
        </soapenv:Body>
      </soapenv:Envelope>
    </format>
    <args />
  </payloadFactory>
  <!-- [2.4] -->
  <header name="soapAction" scope="transport" value="http://tempuri.org/
      IAgendaService/GetSpecialties" />
  <!-- [2.5] -->
  <send>
    <endpoint key="AgendaEP" />
  </send>
</inSequence>
<!-- [2.6] -->
<outSequence>
  <!-- [2.7] -->
  <datamapper config="gov:datamapper/getSpecialtiesMapping.dmc" inputSchema="
      gov:datamapper/getSpecialtiesMapping_inputSchema.json" inputType="XML"
       outputSchema="gov:datamapper/getSpecialtiesMapping_outputSchema.json"
       outputType="JSON" xsltStyleSheet="gov:datamapper/
      getSpecialtiesMapping_xsltStyleSheet.xml" />
  <!-- [2.8] -->
  <respond />
</outSequence>
<!-- [2.9] -->
<faultSequence>
  <!-- [2.10] -->
  <log category="ERROR" level="full" />
</faultSequence>
```

[2.1] La séquence de médiation de la requête débute. Elle contiendra les divers médiateurs qui interviennent dans le flux de médiation.

[2.2] Nous utilisons en premier lieu un médiateur *log* afin d'enregistrer la réception de la requête. Le logging est l'un des outils de surveillance les plus importants d'un serveur de production. Un système de logging correctement configuré est essentiel pour identifier les erreurs, les menaces de sécurité et les modèles d'utilisation.

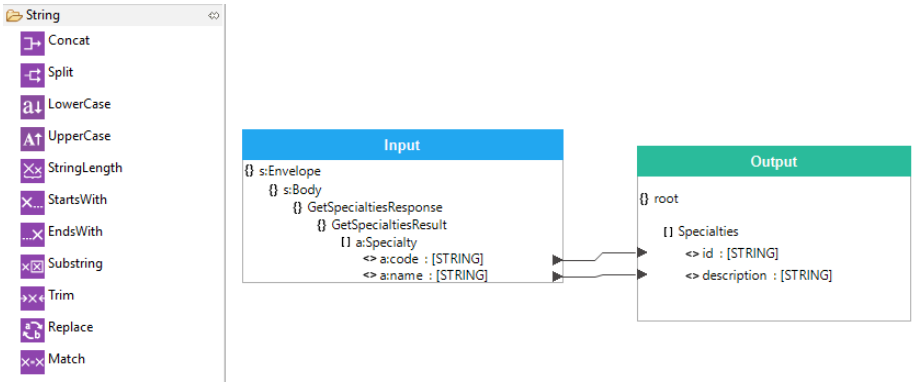
[2.3] Le médiateur *payloadFactory* est un des plus couramment utilisé. Il permet de transformer ou remplacer le contenu d'un message. Dans ce cas, il est utilisé pour spécifier le message SOAP qui doit être envoyé au web service agenda.

[2.4] Le médiateur *header* permet de manipuler les header SOAP et HTTP. Dans notre cas, il est nécessaire de définir la propriété *soapAction* afin que le web service agenda accepte la requête. Elle indique l'action du web service agenda qui est appelée. Dans ce cas il s'agit de l'action *GetSpecialties*.

[2.5] Le médiateur *send* permet d'envoyer la requête au web service agenda que nous avons défini dans un point de sortie nommé « AgendaEP » dans la section 5.2.1. Ce médiateur implique l'arrêt du traitement du flux du message et, une fois la réponse reçue, le début du traitement de celle-ci dans le séquence de sortie. Nous verrons plus tard le médiateur *call* qui permet également l'envoi d'une requête vers un point de sortie mais qui adopte un comportement différent.

[2.6] La séquence de médiation de la réponse débute.

[2.7] Afin de transformer notre réponse SOAP en JSon, nous utilisons un *datamapper*. Il permet le passage entre formats XML, JSon et CSV. Il faut pour cela lui fournir les formats d'entrée et de sortie. Une fois ceci fait, il permet de configurer la transformation des données via l'interface graphique représentée ci-dessous :



Les listings 5.3 et5.4 présentent respectivement la réponse SOAP fournie par le web service agenda et la réponse JSon envoyée au client après le passage par le *datamapper*.

[2.8] Le médiateur *respond* envoie la réponse au client ce qui marque la fin du flux de médiation pour cette ressource.

[2.9] La séquence de faute est déclenchée si une erreur survient durant la médiation.

[2.10] Nous insérons dans ce cas une ligne d'erreur contenant toutes les informations disponibles.

Listing 5.3 – Réponse SOAP du web service agenda à la demande de la liste des spécialités

```
<?xml version="1.0" encoding="UTF-8"
<s:Envelope xmlns:s="http://schemas.
<s:Body>
<GetSpecialtiesResponse xmlns="http:
<GetSpecialtiesResult xmlns:a="http:
<a:Specialty>
<a:code>cardiology</a:code>
<a:name>Cardiologie</a:name>
</a:Specialty>
<a:Specialty>
<a:code>neurology</a:code>
<a:name>Neurologie</a:name>
</a:Specialty>
<a:Specialty>
<a:code>urology</a:code>
<a:name>Urologie</a:name>
</a:Specialty>
<a:Specialty>
<a:code>physiotherapy</a:code>
<a:name>Physiotherapie</a:name>
</a:Specialty>
</GetSpecialtiesResult>
</GetSpecialtiesResponse>
</s:Body>
</s:Envelope>
```

Listing 5.4 – Réponse JSon produite par le datamapper

```
{"Specialties": [
{
"id": "cardiology",
"description": "Cardiologie"
},
{
"id": "neurology",
"description": "Neurologie"
},
{
"id": "urology",
"description": "Urologie"
},
{
"id": "physiotherapy",
"description": "Physiotherapie"
}
]}
```

Obtention de la liste des prochains rendez-vous et réservation d'un rendez-vous L'obtention de la liste des prochaines rendez-vous pour une spécialité et la réservation d'un rendez-vous se réalise en accédant à une même ressource identifiée par l'url `/agenda/specialties/\{specialty\}/appointments` respectivement par une requête GET et une requête PUT. L'ESB doit cependant pouvoir différencier s'il est dans un cas ou dans l'autre. Cette différenciation est expliquée dans le listing 5.5.

Listing 5.5 – Flux de médiation pour l'obtention de la liste des spécialités

```
<inSequence>
  <log />
  <!-- [3.1] -->
  <switch source="$axis2:HTTP_METHOD">
    <!-- [3.2] -->
    <case regex="GET">
      <!-- traitement du GET -->
    </case>
    <!-- [3.3] -->
    <case regex="PUT">
      <!-- traitement du PUT -->
    </case>
    <default />
  </switch>
</inSequence>
<outSequence />
<faultSequence>
  <log level="full" />
</faultSequence>
```

[3.1] Afin d'envoyer la requête à la bonne méthode du web service agenda, nous devons effectuer un routage basé sur le header du message. Pour cela, nous utilisons un médiateur *switch* qui fonctionne de manière similaire à un *switch* en Java. Dans notre cas, il évalue l'expression $axis2 :HTTP_METHOD qui retourne sous forme de chaîne de caractères le verbe http utilisé pour accéder à la ressource. La chaîne est ensuite comparée à l'expression régulière des différents cas du *switch* jusqu'à trouver un cas correspondant. La séquence de médiateurs du cas correspondant est alors exécutée. Il est important de noter que les cas d'un switch s'exécutent dans une seule et même séquence, ce qui implique qu'il ne pourra y avoir qu'une seule séquence de sortie pour l'ensemble des cas de ce *switch*. Nous y reviendrons plus tard.

[3.2] Le verbe http GET correspond à une demande des prochains rendez-vous libres pour une spécialité.

[3.3] Le verbe http PUT correspond à la réservation d'un rendez-vous.

Obtention de la liste des prochains rendez-vous Pour obtenir une liste des prochains rendez-vous libres pour une spécialité donnée, l'application mobile effectue une requête GET sur l'url `/agenda/specialties/\{specialty\}/appointments`. Le paramètre {specialty} désigne la spécialité concernée et sera accessible dans le flux de médiation sous forme d'une variable. Ce dernier est détaillé dans le listing 5.6.

Listing 5.6 – Flux de médiation pour l'obtention de la liste des spécialités

```
<log />
<!-- [3.2.1] -->
<payloadFactory media-type="xml">
  <format>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope
        /" xmlns:tem="http://tempuri.org/">
      <soapenv:Header />
      <soapenv:Body>
        <tem:GetFreeAppointments>
          <!-- [3.2.2] -->
          <tem:specialty>$1</tem:specialty>
        </tem:GetFreeAppointments>
      </soapenv:Body>
    </soapenv:Envelope>
  </format>
  <args>
    <!-- [3.2.3] -->
    <arg evaluator="xml" expression="$ctx:uri.var.specialty" />
  </args>
</payloadFactory>
<header name="soapAction" scope="transport" value="http://tempuri.org/
    IAgendaService/GetFreeAppointments" />
<!-- [3.2.4] -->
<call>
  <endpoint key="AgendaEP" />
</call>
<!-- [3.2.5] -->
<datamapper config="gov:datamapper/getSpecialtiesResponseMapping.dmc"
    inputSchema="gov:datamapper/getSpecialtiesResponseMapping_inputSchema.
```

```
    json" inputType="XML" outputSchema="gov:datamapper/
    getSpecialtiesResponseMapping_outputSchema.json" outputType="XML"
    xsltStyleSheet="gov:datamapper/
    getSpecialtiesResponseMapping_xsltStyleSheet.xml" />
<!-- [3.2.6] -->
<respond />
```

[3.2.1] Nous commençons par créer l'enveloppe SOAP requise par le service agenda grâce à un *payloadFactory*.

[3.2.2] Un argument $1 est ici introduit. Il désigne l'emplacement ou la spécialité concernée sera indiquée. La déclaration d'un argument entraîne la nécessité de définir cet argument dans le *payloadFactory*.

[3.2.3] Nous déclarons cet argument et indiquons à quelle valeur il correspond. Il retourne la variable {specialty} via l'expression $ctx :uri.var.specialty.

[3.2.4] Nous faisons appel au web service agenda grâce au médiateur *call*. Contrairement au médiateur *send*, il n'entraîne pas la fin de la séquence d'entrée. La réponse fournie est simplement envoyée au médiateur qui suit dans la séquence d'entrée. Ceci va nous permettre d'envoyer une réponse différente pour chaque cas du switch.

[3.2.5] Un *datamapper* est ensuite utilisé pour convertir la réponse SOAP au format JSon requis par l'application cliente.

[3.2.6] Pour terminer, la réponse est envoyée au client grâce au médiateur *respond*.

Réservation d'un rendez-vous Pour effectuer une réservation d'un rendez-vous, l'application mobile effectue une requête PUT sur l'url `/agenda/specialties/\{specialty\}/appointments`. Le paramètre {specialty} désigne la spécialité concernée. La requête est accompagnée d'un payload JSon similaire à celui du listing 5.7. Le flux de médiation correspondant est visible dans le listing 5.6.

Listing 5.7 – Payload de la requête REST de réservation d'un rendez-vous.

```
{
  "appointmentId": "e6adf2b1-75ef-441e-aee5-dc124f44b44b",
  "patientId": 350212
}
```

Listing 5.8 – Flux de médiation pour l'obtention de la liste des spécialités

```
<log />
<!-- [3.3.1] -->
<payloadFactory media-type="xml">
  <format>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope
        /" xmlns:tem="http://tempuri.org/">
      <soapenv:Header />
      <soapenv:Body>
        <tem:Book>
          <!-- [3.3.2] -->
          <tem:appointment>$1</tem:appointment>
          <tem:patientId>$2</tem:patientId>
        </tem:Book>
      </soapenv:Body>
    </soapenv:Envelope>
```

```
  </format>
  <args>
    <!-- [3.3.3] -->
    <arg evaluator="json" expression="$.appointmentId" />
    <arg evaluator="json" expression="$.patientId" />
  </args>
</payloadFactory>
<header name="soapAction" scope="transport" value="http://tempuri.org/
    IAgendaService/Book" />
<call>
  <endpoint key="AgendaEP" />
</call>
<!-- [3.3.4] -->
<respond />
```

[3.3.1] L'enveloppe SOAP est construite grâce à un médiateur *payloadFactory*.

[3.3.2] Elle sera complétée par deux arguments.

[3.3.3] Ces deux arguments sont ensuite déclarés. Ils correspondent aux champs du payload JSon accompagnant la requête REST.

[3.3.4] La requête a été envoyée au web service agenda. Aucun formatage n'est nécessaire sur la réponse. Un code HTTP 200 est suffisant.

Dans ce premier cas, nous avons mis en pratique les concepts clés de la section 5.1.2 que sont le point d'entrée, la séquence, le point de sortie. Nous avons également utilisé quelques médiateurs.

- call : il permet d'envoyer un message à un point de sortie et de continuer le déroulement normal de la séquence.
- datamapper : Il permet le passage d'un format à un autre ainsi que la modification de la structure des données dans un message.
- header : il permet de manipuler un header SOAP ou HTTP en modifiant, supprimant ou en ajoutant des propriétés à celui-ci.
- log : il permet d'insérer des informations dans les logs du serveur ESB.
- payloadFactory : il permet de manipuler le contenu des message.
- property : Il permet de définir ou modifier des propriétés qui seront accessibles tout au long du flux de médiation.
- send : il permet d'envoyer un message à un point de sortie. Il implique la fin de la séquence et le début de la séquence de sortie.
- respond : Il est utilisé pour envoyer une réponse au client.
- switch : il permet d'effectuer un routage d'une requête sur base d'expression XPath, JSonPath ou définie dans WSO2ESB.

Grâce à ces concepts, nous avons pu illustrer la transformation des requêtes REST envoyées par une application cliente en requêtes SOAP acceptées par un web service d'arrière plan. Nous avons également vu comment effectuer un routage.

5.3 Cas 2 : Flux de patients

Pour ce second et dernier cas, nous allons traiter les informations envoyées par un système de gestion des patients dans un hôpital. Ce système gère entre autres choses les admissions, sorties et mouvements de patients. Assurer la bonne circulation des ces informations est essentiel. La figure 5.11 illustre les systèmes qui interviennent dans le cas que nous allons concevoir. Le système PMS est le gestionnaire de patients. Chaque évènement qui est encodé est aussitôt communiqué à l'ESB sous la forme de messages HL7 par l'intermédiaire du protocole MLLP. Nous sommes ici en présence d'une interaction de type one-way. En effet, le PMS envoie son message à l'ESB et continue son exécution. Son travail se limite à cela, il ne se soucie pas de savoir à qui il sera communiqué ni si il sera bien acheminé. L'ESB prend donc le relais et doit communiquer l'information à trois systèmes :

1. Un équipement médical au moyen d'une communication semblable à celle utilisée par le PMS. Il sera donc envoyé en TCP/IP via la protocole MLLP.

2. Un système de gestion de laboratoire. Les messages HL7 seront envoyés vers un répertoire sous la forme de fichiers.

3. Un dossier médical électronique. Celui-ci dispose de web services SOAP permettant de communiquer les messages HL7 afin qu'ils soient intégrés dans l'application.

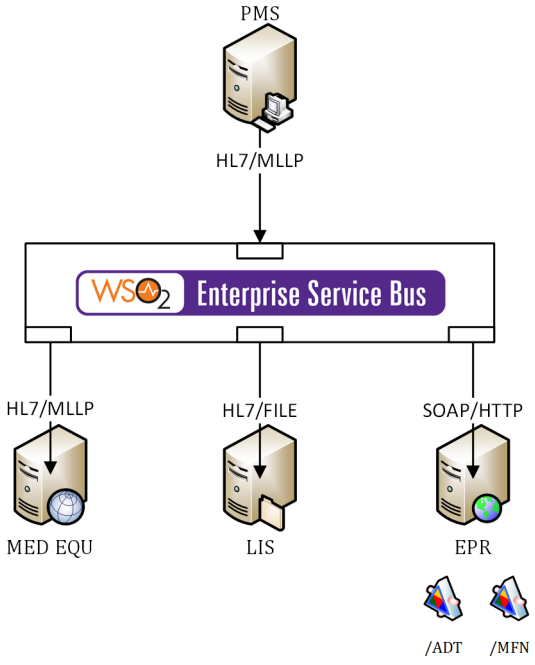


Figure 5.11 – Vue conceptuelle du cas 2 : Flux de patients.

5.3.1 Les messages HL7

HL7 est une norme internationale désignant un format de message dédié à la communication d'informations de santé. Un exemple de message HL7 est fourni dans le listing 5.9.

Listing 5.9 – Un message HL7 de type ADT-A31

```
MSH|^~\&|HL7Soup|Instance1|HL7Soup|Instance2|20180717090613||ADT^A31
     |0000098313|P|2.5|||||BEL|8859/1
EVN|A31|201807170906|||encoder|201807170906|11
PID||57279|57279||EASTWOOD^CLINT||19580313|M|||Hollywood Bvd ^^NAMUR 1^^5000^
     BE||""~""||fr|U|||58031308335|""|""||NAMUR|||BE|||""|N|||20191031110613
NTE|1
ZPA|""||""|""|||""|""|""|""
ROL||DE|FHCP|""
NK1|""|""^""|""|""^^""^^""^""|""
PV1|1|N
```

Décrire précisément le fonctionnement des messages HL7 n'aurait que peu d'intérêt dans le contexte de ce travail. Nous nous limiterons donc à une explication succincte. Un message HL7 est émis lors de l'occurrence d'un évènement particulier comme une admission de patient. Sa structure générale est la suivante :

- Segments : Ils commencent par une séquence de 3 lettres et se terminent par un retour à la ligne
  - Éléments : Ils sont délimités par un « | ». Certains sont répétables.
    - Composants : Ils sont délimités par un « ^ ».
      - Sous-composants : Ils sont délimités par un « & ».

Un message HL7 est composé de segments qui contiennent des informations corrélées. Ils démarrent par un identifiant en 3 lettres et se terminent par un retour à la ligne. Le premier segment de tout message HL7 est toujours le segment *MSH* qui contient des méta données sur le message. Il est unique et obligatoire. Suivent d'autres segments qui dépendent du type et du sous type de message. Ces derniers sont indiqués dans le segment *MSH*. Il existe plusieurs dizaines de types de messages HL7. Nous en utiliserons deux :

ADT Il correspond aux messages relatifs aux informations démographiques des patients.

MFN Il correspond à ce que la norme appelle des fichiers communs de références autrement dit des références utilisées par plusieurs applications.

A titre d'exemple, l'ajout d'un patient produira un message de type ADT-A28. Si ses données sont mises à jour, le message émis sera alors de type ADT-A31 comme dans le listing 5.9. L'apparition d'un nouveau médecin aura pour effet de produire un message de type MFN-M02.

5.3.2 Création du point d'entrée

Avant de construire le flux de médiation, nous devons créer nos points d'entrée et de sortie. Nous commençons donc par créer un point d'entrée de manière similaire à celui créé dans la section 5.2.2 mais nous choisissons cette fois le type HL7. Un point d'entrée de ce type contient systématiquement une séquence de traitement normal

et une séquence de traitement d'erreur. Nous les créerons plus tard. Le listing 5.10 montre le code XML de notre point d'entrée. On y constate que ce point d'entrée sera à l'écoute sur le port TCP 2000. A noter également qu'il enverra, dès réception d'un message, un acquittement au client. Celui-ci peut dès lors continuer son exécution sans attendre la fin de l'exécution du flux de traitement du message par l'ESB.

Listing 5.10 – Point d'entrée de type HL7

```
<?xml version="1.0" encoding="UTF-8"?>
<inboundEndpoint name="HL7GW"
                 protocol="hl7"
                 suspend="false"
                 xmlns="http://ws.apache.org/ns/synapse">
    <parameters>
        <parameter name="inbound.hl7.Port">2000</parameter>
        <parameter name="inbound.hl7.AutoAck">true</parameter>
        <parameter name="inbound.hl7.TimeOut">10000</parameter>
        <parameter name="inbound.hl7.CharSet">UTF-8</parameter>
        <parameter name="inbound.hl7.ValidateMessage">true</parameter>
        <parameter name="inbound.hl7.BuildInvalidMessages">true</parameter>
        <parameter name="inbound.hl7.PassThroughInvalidMessages">true</
            parameter>
    </parameters>
</inboundEndpoint>
```

5.3.3 Création des points de sortie

Nous disposons de notre point d'entrée. Il nous faut maintenant définir les points de sortie à qui seront communiquer les messages HL7. Ils seront au nombre de 4 :

1. Un équipement médical comme par exemple un électrocardiogramme. Il recevra les messages de la même manière que notre point d'entrée c'est à dire sur un port TCP/IP dédié à cet effet.
2. Les messages seront également communiqués à un système de gestion du laboratoire d'analyse. Celui-ci désire recevoir les messages sous forme de fichiers dans un répertoire défini.
3. L'EPR, qui est pour rappel un dossier médical électronique, doit également être informé et souhaite l'être par l'intermédiaire de ses web services SOAP. Un pour les messages de type ADT, un autre pour ceux de type MFN.

L'équipement médical De façon similaire à la section 5.2.1, nous créons un point de sortie mais nous choisissons cette fois le type « Address Endpoint ». Nous lui donnons un nom et renseignons l'adresse `hl7://localhost:2001`. Elle indique que nous souhaitons envoyer des messages hl7 à l'adresse localhost sur le port 2001. Le résultat de l'opération est visible dans le listing 5.13.

Listing 5.11 – Point de sortie de type HL7 de l'appareil médical

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="ECG_MLLP_EP" xmlns="http://ws.apache.org/ns/synapse">
    <address uri="hl7://localhost:2001"/>
</endpoint>
```

Le laboratoire d'analyse Nous ajoutons un point de sortie à nouveau de type « Address Endpoint ». Nous utilisons cette fois l'adresse `vfs:file:///d:/hl7/in/` qui indique à l'ESB d'enregistrer les fichiers qu'il recevra dans le répertoire d :/hl7/in de la machine courante. Il est également possible d'utiliser un répertoire partagé via une adresse du type `vfs:smb://remotehost/hl7in`. Le fichier XML correspondant est visible dans le listing 5.13.

Listing 5.12 – Point de sortie de type fichier du laboratoire d'analyse.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="LAB_HL7FILE_EP" xmlns="http://ws.apache.org/ns/synapse">
    <address uri="vfs:file:///d:/temp/hl7/in/"/>
</endpoint>
```

Le dossier médical électronique Il nous reste à créer les points de sortie de l'EPR qui expose des web services SOAP pour la réception des messages HL7. Plus exactement, il en expose un par type de message HL7. Nous en créerons deux, un pour les messages ADT et un pour les messages MFN. La procédure est similaire à celle utilisée dans la section 5.2.1. Le listing 5.13 montre le résultat de la création de ces deux points de sortie.

Listing 5.13 – Point de sortie de type web service SOAP du dossier médical électronique.

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="EPR_WS_ADT_EP" xmlns="http://ws.apache.org/ns/synapse">
    <wsdl port="BasicHttpBinding_adt"
          service="Adt"
          uri="http://localhost:2300/Adt.svc?singleWsdl"/>
</endpoint>

<?xml version="1.0" encoding="UTF-8"?>
<endpoint name="EPR_WS_MFN_EP" xmlns="http://ws.apache.org/ns/synapse">
    <wsdl port="BasicHttpBinding_mfn"
          service="Mfn"
          uri="http://localhost:2300/Mfn.svc?singleWsdl"/>
</endpoint>
```

5.3.4 Création du flux de médiation

Un point d'entrée de type HL7 impose la création de deux séquences dans des fichiers séparés. Nous procéderons également de la sorte pour le reste du flux de médiation. Une vue globale de celui-ci est présenté sur le figure 5.12.

On y constate que 5 séquences seront nécessaires :

— La séquence normale correspond à celle qui sera exécutée lors de la réception d'un message HL7 par notre point d'entrée.

— La séquence d'erreur correspond à celle qui sera exécutée si une erreur est rencontrée lors de la réception d'un message

— La séquence EPR correspond à l'envoi des message HL7 vers le dossier médical électronique.

— La séquence équipement médical correspond à l'envoi des messages HL7 vers un équipement médical à l'écoute sur un port tcp.

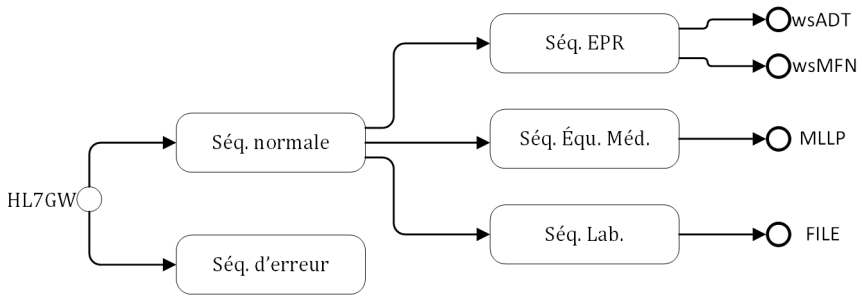


FIGURE 5.12 – Séquences du flux de médiation des patients.

— La séquence laboratoire correspond à l'envoi des message HL7 vers un laboratoire d'analyse.

Nous commençons par créer ces séquences. Une séquence est simplement créée via le menu contextuel du dossier *sequences* de la figure 5.6. Nos procédons donc de la sorte pour créer nos 5 séquences que nous allons ensuite configurer.

Séquence d'erreur La séquence d'erreur est très simple. Nous nous contentons d'y ajouter un médiateur *log* et le paramétrons pour enregistrer l'entièreté d'une éventuelle erreur. Le XML résultant est visible dans le listing 5.14

Listing 5.14 – Séquence d'erreur du flux des patients.

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence name="FaultSeq" trace="disable"
          xmlns="http://ws.apache.org/ns/synapse">
    <log category="ERROR"
         level="full"/>
</sequence>
```

Séquence normale Nous créons ensuite notre séquence normale dont le fonctionnement sera assez simple. Il consistera à enregistrer le type et le sous type de message dans deux propriétés et à envoyer le message vers nos trois systèmes. Le résultat de cette configuration est visible dans le listing 5.15 suivi des explications nécessaires.

Listing 5.15 – Séquence normale du flux des patients.

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence name="NormalSeq" trace="disable" xmlns="http://ws.apache.org/ns/
    synapse">
    <!-- [1] -->
    <propertyGroup>
        <!-- [2] -->
        <property expression="//hl7:MSG.1[1]/text()"
                  name="HL7_TYPE"
                  scope="default"
                  type="STRING" xmlns:hl7="urn:hl7-org:v2xml"/>
        <!-- [3] -->
        <property expression="//hl7:MSG.2[1]/text()"
                  name="HL7_SUBTYPE"
                  scope="default"
```

```
                    type="STRING" xmlns:hl7="urn:hl7-org:v2xml"/>
    </propertyGroup>
    <!-- [4] -->
    <clone>
        <!-- [5] -->
        <target>
            <sequence>
                <!-- [6] -->
                <sequence key="EprSeq"/>
            </sequence>
        </target>
        <target>
            <sequence>
                <sequence key="LabSeq"/>
            </sequence>
        </target>
        <target>
            <sequence>
                <sequence key="MedEquSeq"/>
            </sequence>
        </target>
    </clone>
</sequence>
```

[1] Nous ajoutons un médiateur *propertyGroup*. Comme son nom l'indique, il permet de grouper la définition de plusieurs propriétés.

[2] Nous ajoutons un médiateur *property* avec le nom HL7_TYPE, de type STRING et dont la valeur sera donnée par l'expression XPath `//hl7:MSG.1[1]/text()`. Ceci nous permet d'obtenir le type de message HL7. Nous pourrons dès lors utiliser cette propriété dans le reste du flux de médiation.

[3] Nous procédons de la même manière pour le sous-type du message.

[4] Nous ajoutons ensuite un médiateur *clone*. Il permet de copier un message autant de fois que nécessaire.

[5] Nous ajoutons 3 cibles (target) dans notre médiateur *clone*. Une pour l'EPR, une pour le laboratoire et enfin la dernière pour l'équipement médical.

[6] Pour terminer, nous ajoutons à chaque cible la séquence qui lui correspond.

Séquence EPR La séquence EPR a pour objectif de transmettre les messages HL7 vers les web services du dossier médical électronique. Celui-ci n'est intéressé que par les messages ADT et MFN, et présente un web service pour chacun de ces types. Le résultat de cette configuration est visible dans le listing 5.16 suivi des explications nécessaires.

Listing 5.16 – Séquence EPR du flux des patients.

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence name="EprSeq" trace="disable" xmlns="http://ws.apache.org/ns/
    synapse">
    <log>
        <property name="ROUTE" value="EPR"/>
    </log>
    <!-- [1] -->
    <propertyGroup>
        <property name="ROUTE" scope="default" type="STRING" value="EPR"/>
```

```
        <!-- [2] -->
        <property name="messageType" scope="axis2"
                  type="STRING" value="application/edi-hl7"/>
        <property name="ContentType" scope="axis2"
                  type="STRING" value="application/edi-hl7"/>
    </propertyGroup>
    <!-- [3] -->
    <payloadFactory media-type="xml">
        <format>
            <soapenv:Envelope
              xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
              xmlns:epr="http://epr/"
              xmlns:wsep="http://schemas.datacontract.org/2004/07/WsEpr.
                   DataContract">
               <soapenv:Header/>
               <soapenv:Body>
                  <epr:Send>
                     <epr:message>
                        <!-- [4] -->
                        <wsep:content><![CDATA[$1]]></wsep:content>
                     </epr:message>
                  </epr:Send>
               </soapenv:Body>
            </soapenv:Envelope>
        </format>
        <args>
            <!-- [5] -->
            <arg evaluator="xml" expression="$body/*[1]"/>
        </args>
    </payloadFactory>
    <!-- [6] -->
    <switch source="get-property('HL7_TYPE')">
        <!-- [7] -->
        <case regex="ADT">
            <log>
                <property name="ROUTE" value="EPR - ADT"/>
            </log>
            <!-- [8] -->
            <header name="soapAction" scope="transport"
                    value="http://epr/adt/Send"/>
            <!-- [9] -->
            <call>
                <endpoint key="EPR_WS_ADT_EP"/>
            </call>
        </case>
        <!-- [10] -->
        <case regex="MFN">
            <log>
                <property name="ROUTE" value="EPR - MFN"/>
            </log>
            <!-- [11] -->
            <header name="soapAction" scope="transport"
                    value="http://epr/mfn/Send"/>
            <!-- [12] -->
            <call>
                <endpoint key="EPR_WS_MFN_EP"/>
            </call>
        </case>
        <default>
            <log>
                <property name="ROUTE" value="EPR - OTHER - DROPPING"/>
            </log>
```

```
            <!-- [13] -->
            <drop/>
        </default>
    </switch>
</sequence>
```

[1] Nous ajoutons un *propertyGroup*

[2] Nous y définissons deux *property* dont l'objectif est d'indiquer à l'ESB que nous souhaitons travailler avec le format HL7. Sans ces deux propriétés, le message HL7 serait envoyé au format XML dans le contenu de l'enveloppe SOAP (voir point 5).

[3] Nous créons l'enveloppe SOAP qui sera envoyée au web service de l'EPR à l'aide d'un *payloadFactory*.

[4] Nous y déclarons un argument $1.

[5] Celui-ci prendra la valeur de l'expression `\$body/*[1]`. Elle correspond au contenu du message reçu par l'ESB autrement dit à notre message HL7.

[6] Nous utilisons ensuite un médiateur *switch*. Celui-ci va effectuer le routage de notre message en fonction de son type. Pour cela, il fait appel à la propriété HL7_TYPE que nous avions déclaré dans la séquence normale.

[7] Si le message est du type ADT

[8] Grâce au médiateur *header*, nous créons une propriété *soapAction* qui sera insérée dans le header de la requête HTTP qui sera envoyé au web service EPR. Nous y indiquons la méthode du web service à laquelle nous souhaitons faire appel.

[9] Nous appelons le web service que nous avions défini lors de la création des points de sortie.

[10] Sinon si le message est du type MFN

[11] Nous renseignons la méthode que nous souhaitons appeler via l'ajout d'une propriété dans le header de la requete HTTP qui sera envoyée au web service.

[12] Nous appelons le web service que nous avions défini lors de la création des points de sortie.

[13] Sinon il s'agit d'un autre type de message et nous l'ignorons grâce à l'ajout d'une médiateur *drop*.

Séquence laboratoire La séquence laboratoire a pour objectif de transmettre les message HL7 au système de gestion du laboratoire d'analyse. Celui-ci désire recevoir tous les messages sous forme de fichiers. Le résultat de cette configuration est visible dans le listing 5.17 suivi des explications nécessaires.

Listing 5.17 – Séquence laboratoire du flux des patients.

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence name="LabSeq" trace="disable" xmlns="http://ws.apache.org/ns/
    synapse">
    <!-- [1] -->
    <propertyGroup>
        <property name="ROUTE" scope="default" type="STRING" value="LAB"/>
        <!-- [2] -->
```

```
        <property name="OUT_ONLY" scope="default"
                  type="STRING" value="true"/>
        <!-- [3] -->
        <property name="messageType" scope="axis2"
                  type="STRING" value="application/edi-hl7"/>
        <property name="ContentType" scope="axis2"
                  type="STRING" value="application/edi-hl7"/>
        <!-- [4] -->
        <property name="transport.vfs.ReplyFileName" scope="transport"
                  type="STRING"
                  expression="fn:concat(get-property('SYSTEM_DATE', 'yyyyMMdd
                      .HHmmssSSS'), '.hl7')"/>
        <!-- [5] -->
        <property name="transport.vfs.ContentType" scope="default"
                  type="STRING" value="application/edi-hl7"/>
    </propertyGroup>
    <!-- [6] -->
    <send>
        <endpoint key="LAB_HL7FILE_EP"/>
    </send>
</sequence>
```

[1] Nous utilisons un groupe de propriétés grâce à un médiateur *propertyGroup*.

[2] Nous utilisons une propriété générique de WSO2 ESB. Elle se nomme OUT_ONLY. Nous lui donnons la valeur true qui indique à l'ESB qu'il ne doit attendre aucune réponse suite à la transmission du message.

[3] Nous indiquons via deux propriétés que nous souhaitons travailler au format HL7.

[4] Nous construisons le nom du fichier. Nous utilisons une nomenclature qui permet de conserver la séquence de réception des messages HL7.

[5] Nous indiquons à l'ESB que les fichiers seront de type HL7.

[6] Nous appelons le point de sortie de type fichier que nous avions créé dans la section 5.3.3.

Séquence équipement médical La séquence équipement médical a pour objectif de transmettre les messages HL7 à un appareillage médical. Celui-ci souhaite les recevoir sur un port TCP mais n'est intéressé que par les messages de type ADT et de sous type A04. Ils correspondent à l'enregistrement de la visite d'un patient en consultation par exemple. Le résultat de cette configuration est visible dans le listing 5.17 suivi des explications nécessaires.

Listing 5.18 – Séquence équipement médical du flux des patients.

```
<?xml version="1.0" encoding="UTF-8"?>
<sequence name="MedEquSeq" trace="disable" xmlns="http://ws.apache.org/ns/
    synapse">
    <log>
        <property name="ROUTE" value="MED_EQUIPMENT"/>
    </log>
    <log level="full"/>
    <!-- [1] -->
    <filter regex="true"
            source="get-property('HL7_TYPE') = 'ADT' and get-property('
                HL7_SUBTYPE') = 'A04'">
```

```
        <then>
            <log>
                <property name="ROUTE" value="MED_EQU - ADT-A04"/>
            </log>
            <!-- [2] -->
            <call>
                <endpoint key="HL7MEDEQU"/>
            </call>
        </then>
        <else>
            <log>
                <property name="ROUTE" value="MED_EQU - OTHER - DROPPING"/>
            </log>
            <!-- [3] -->
            <drop/>
        </else>
    </filter>
</sequence>
```

[1] Nous commençons par utiliser un médiateur *filter*. Il nous permettra de ne traiter que les messages des types et sous types désirés. Il évalue pour cela l'expression renseignée comme source. Il est intéressant de noter que des opérateurs peuvent être utilisés évitant de ce fait de devoir cascader plusieurs médiateur *filter*.

[2] Si nous sommes en présence d'un message ADT A04, nous l'envoyons simplement au point de sortie créé dans la section 5.3.3.

[3] Sinon nous l'ignorons par l'ajout d'un médiateur *drop*.

5.4 Résumé

Ce chapitre a été consacré à la mise en pratique d'un ESB. Nous avons pour cela utilisé l'ESB WSO2 ESB qui a été identifié comme 1er choix dans la mise en pratique de notre méthodologie d'évaluation et de sélection d'ESB. Nous avons commencé par décrire l'architecture du produit et nous sommes plus particulièrement intéressés aux composants clés permettant la mise en place d'un flux de médiation. Ceux-ci sont :

— Les points d'entrée, qui peuvent prendre plusieurs formes et qui permettent la communication de messages à l'ESB.

— Les séquences et les médiateurs qui permettent le traitement des messages.

— Les points de sortie qui sont la représentation virtuelle dans l'ESB de services externes.

Nous avons ensuite mis en pratique 2 cas concrets. Dans le premier cas, nous avons établi une communication entre un agenda médical exposant un web service SOAP et une application mobile de prise de rendez-vous effectuant des requêtes REST/JSon. Dans le second, nous avons mis en place un flux de communication de messages HL7 entre un système de gestion de patients et un dossier médical électronique, un système de gestion de laboratoire d'analyse et enfin un appareillage médical. Nous avons pour cela utilisé les médiateurs suivants :

— call : il permet d'envoyer un message à un point de sortie et de continuer le déroulement normal de la séquence.

— clone : Il permet d'effectuer un nombre spécifié de copies d'un message. Chaque copie peut alors être traitée selon une séquence qui lui est propre.

— datamapper : Il permet le passage d'un format à un autre ainsi que la modification de la structure des données dans un message.

— drop : Il permet de stopper le traitement du message courant. Il implique l'arrêt de la séquence courante. Tout médiateur situé en aval de ce médiateur sera donc ignoré.

— filter : Il permet de filtrer les messages sur base de leur header ou de leur contenu.

— header : il permet de manipuler un header SOAP ou HTTP en modifiant, supprimant ou en ajoutant des propriétés à celui-ci.

— log : il permet d'insérer des informations dans les logs du serveur ESB.

— payloadFactory : il permet de manipuler le contenu des messages.

— property : Il permet de définir ou modifier des propriétés qui seront accessibles tout au long du flux de médiation.

— send : il permet d'envoyer un message à un point de sortie. Il implique la fin de la séquence et le début de la séquence de sortie.

— respond : Il est utilisé pour envoyer une réponse au client.

— switch : il permet d'effectuer un routage d'une requête sur base d'expression XPath, JSonPath ou définie dans WSO2ESB.

Conclusion

Ce mémoire avait pour ambition de répondre à deux questions de recherche. « Quelle pourrait être un modèle des capacités fonctionnelles des ESB ? » et « Quelle méthodologie pourrait être suivie pour évaluer et sélectionner un ESB sur base de ce modèle ? ». Un ESB est un modèle architectural d'applications dont l'objectif est de permettre d'établir une communication entre systèmes hétérogènes. C'est donc un outil d'intégration d'applications. Nous avons donc débuté en posant un cadre général de la notion d'intégration dans une entreprise. Nous avons vu que l'intégration, d'un point de vue global, intervient à deux niveaux. Le premier est le niveau organisationnel dans lequel on fera davantage usage du terme coordination. Le second niveau concerne l'intégration des systèmes. Il vient en support du niveau organisationnel et est composé de 3 couches ; la couche données ; la couche applications ; la couche processus métiers.

Dans un second temps, et afin de bien comprendre et cerner le domaine, nous avons proposé un parcours exhaustif des différents styles, méthodes et architectures d'intégration d'applications, accompagnés d'exemples didactiques. Dans le style file transfer, les données entre systèmes circulent sous forme de fichiers. Le style shared database propose que les applications utilisent une base de données commune. Les RPC permettent à une application de faire appel aux méthodes d'une autre application à distance selon diverses techniques comme RMI ou encore les web services SOAP. L'entreposage de méthodes repose sur la constitution d'un ensemble unique de méthodes devant être utilisé par les applications d'une organisation. Un ERP regroupe l'ensemble ou une partie des métiers d'une entreprise dans une seule et même application. ETL, qui travaille au niveau de la couche des données, est particulièrement adapté à la constitution d'un datawarehouse mais peut également constituer une solution d'intégration d'applications. Le style messaging repose sur une communication asynchrone de messages entre applications. 4 architectures sont ensuite répertoriées et expliquées. La première est l'architecture point à point dans laquelle chaque besoin d'une intégration entre deux applications est envisagé individuellement. L'architecture hub'n spoke a contrario, propose de rassembler toute l'intégration en un point central (le hub) avec lequel toutes les applications communiquent. Vient ensuite l'architecture SOA qui constitue le standard actuel en matière d'architecture d'intégration. Elle repose sur les notions de services, de consommateurs de service et d'annuaire. Ces services peuvent cependant être implémentés à l'aide de diverses technologies imposant aux applications de s'adapter l'une à l'autre. C'est la que l'architecture ESB intervient en se positionnant comme médiatrice entre les applications d'un parc informatique. Elle propose de prendre en charge les diverses tâches de transformation de protocoles et de formats de données. Nous

consacrons un chapitre à l'étude détaillée de cette architecture en décrivant ses divers composants et caractéristiques. Nous proposons également des pistes de réflexion permettant de répondre à la question « Quand utliser un ESB ? » à travers des signaux d'usage, un modèle des coûts et les bénéfices qu'il peut apporter.

Nous sommes ensuite entrés dans le vif du sujet avec le chapitre Modèle des capacités des ESB. Il apporte une réponse à notre première question de recherche : « Quelle pourrait être un modèle des capacités des ESB ? ». A partir d'une étude méthodologique de la littérature existante sur le sujet, nous mettons en évidence l'absence d'un modèle global des capacités des ESB. Une analyse est donc menée sur base de 18 références bibliographiques afin de construire un modèle des capacités des ESB. Ce modèle est formalisé grâce à un diagramme de fonctionnalités dans lequel 6 catégories principales de fonctionnalités sont identifiées. La connectivité fournit les mécanismes nécessaires pour connecter une application à l'ESB. Les capacités permettant l'interactions entre client et fournisseur de services sont regroupées dans les interactions de services. La catégorie traitement des messages contient les possibilités de traiter ceux-ci. La catégorie de gestion indique les capacités de gestion au quotidien de l'ESB. Les catégories sécurité et qualité de service parlent d'elles mêmes

A partir du modèle des capacités proposé, nous avons ensuite développé notre seconde question de recherche : « Quelle méthodologie pourrait être suivie pour évaluer et sélectionner un ESB sur base de ce modèle ? ». Pour cela nous avons, dans le chapitre Évaluation et sélection d'un ESB, proposé une méthodologie de sélection et d'évaluation des ESB baptisée des initiales des 3 phases qui la composent : **A**nalyse ; **S**creening ; **D**écision. Cette méthode est adaptable à tous les contextes et à toutes les tailles d'entreprises. En premier lieu, une analyse dont l'objectif est d'établir une cartographie des besoins est menée. Elle se conclu par l'élaboration des critères qui serviront à l'évaluation des ESB. En second lieu, le screening établit une liste des ESB du marché et réalise un premier tri selon des critères critiques. En troisième lieu, la phase de décision comprend la pondération des catégories de critères, la collecte des données et enfin l'évaluation et la décision. La pondération fournit l'importance de chaque catégorie de critères. La collecte des données consiste à évaluer chaque produit sur chacun des critères établis durant l'analyse. Pour terminer, l'évaluation permet de calculer la cotation finale d'un produit à partir de la collecte des données et de la pondération. Elle fournit les informations nécessaires aux choix d'un ESB. Nous avons terminé le chapitre en appliquant la méthodologie proposée à notre cas d'accompagnement. Nous y proposons d'établir la pondération selon deux méthodes. La méthode des jetons et la méthode AHP qui débouchent au final sur la même hiérarchie de produit avec en tête WSO2 ESB.

En guise de fin, nous avons mis en application WSO2 ESB sur 2 cas concrets de notre cas d'accompagnement. Le premier consiste à établir une communication entre une application mobile de réservation de rendez-vous utilisant REST et JSon et un web service SOAP. Le second repose sur la transmission, aux divers systèmes qui composent un système d'information hospitalier, de message HL7. Réceptionnés sur un port TCP de l'ESB, ils sont ensuite transmis par ce dernier sous forme de fichiers à un système de gestion d'un laboratoire d'analyse, à un port TCP exposé

par un appareillage médical et enfin par appels aux web services SOAP d'un dossier médical électronique.

5.5 Perspectives et retour d'expérience

Le modèle des capacités et la méthodologie d'évaluation et de sélection que nous avons proposé ne prennent en compte que les aspects fonctionnels des ESB. Toutefois, dans la sélection d'un ESB ou d'une application de manière générale, d'autres aspects doivent également être envisagés. Nous pensons en premier lieu au facteur financier. Celui-ci constitue en effet un critère souvent important dans le choix d'un produit. Il devrait donc prendre en compte le prix d'achat du produit, le coût des formations nécessaires et le coût du support et de sa maintenance. Ensuite, des critères tels que la disponibilité et la qualité de la documentation, la réputation du produit, ses parts générales de marchés ainsi que celles spécifiques au domaine d'application concerné, la qualité du support, ou encore l'importance de la communauté d'utilisateurs et son activité devraient également être pris en compte.

Dans notre méthodologie de sélection, nous avons notamment utilisé la méthode de décision multicritère AHP pour établir la pondération des catégories principales de critères. Celle-ci est largement utilisée pour l'évaluation de logiciels. Cette pondération pourrait néanmoins s'étendre à tous les niveaux du modèle des capacités. Elle nécessiterait cependant un grand nombre de comparaisons et calculs qu'il serait laborieux de réaliser manuellement. Une première piste serait de réaliser le tout dans une feuille de calcul. Cependant la liste des critères d'évaluation des ESB étant variables en fonction du contexte, cette solution manquerait de souplesse. Il serait donc intéressant, dans cette éventualité, de développer un outil d'évaluation dans lequel serait intégré notre modèle des capacités auquel serait appliqué globalement la méthode AHP.

Concernant la mise en pratique de WSO2 ESB, nous en avons retiré les enseignements suivants : utiliser un ESB pour faire communiquer divers systèmes nécessite une compréhension et une maîtrise de toutes les technologies utilisées. Ceci nous apparaît comme un pré-requis à toute mise en place. Cet aspect devrait donc être évalué lorsqu'on envisage d'utiliser un ESB en se posant la question : « Dispose-t-on des compétences nécessaires pour installer, configurer, maintenir et surveiller un tel outil ? ». WSO2 ESB dispose d'une documentation importante mais qui malheureusement n'aborde que des cas d'école qu'il est parfois difficile d'adapter à une situation réelle. Ceci nous a contraint à dépenser beaucoup de temps à rechercher des solutions aux problèmes rencontrés, notamment questionnant des forums d'utilisateurs. Certaines subtilités, non présentes dans la documentation officielle, y sont renseignées. Nous en tirons la conclusion que le choix d'un ESB par notre méthode de sélection et d'évaluation ne devrait pas uniquement être dicté par le résultat de l'évaluation mais également par la mise en place d'un proof of concept permettant de valider d'avantage ce choix.

Enfin et pour terminer, l'ESB est régulièrement présenté comme une solution permettant de résoudre le problème du grand nombre d'interfaces qu'impose une ar-

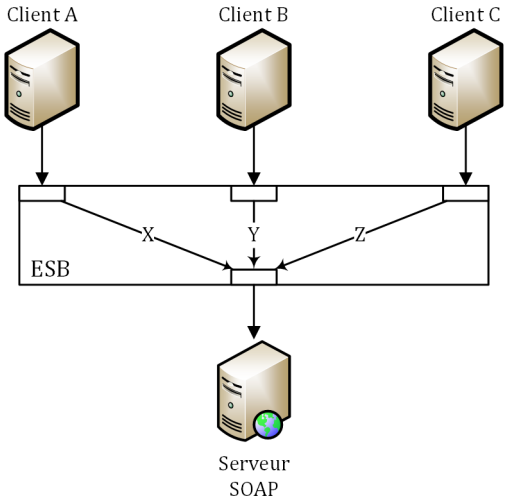


FIGURE 5.13 – Vue conceptuelle du cas 2 : Flux de patients.

chitecture point à point. Selon nous, cette affirmation est partiellement fausse. Pour étayer nos dires, prenons l'exemple simple illustré sur la figure 5.13 dans lequel 3 clients font appel par l'intermédiaire d'un ESB à un web service SOAP. Nous sommes bien en présence de 3 interfaces (X, Y et Z). Si nous remplaçons un des clients, nous ne devons effectivement remplacer qu'un interface. Cependant, si nous remplaçons le serveur, nous devons bien ré-écrire 3 interfaces. De même, dans l'hypothèse de l'ajout d'un serveur auquel ces trois clients devraient faire appel, nous serions à nouveau contraint de créer 3 nouveaux interfaces. Un ESB constitue néanmoins une solution élégante aux problèmes d'intégration pour toutes les autres raisons que nous avons évoquées dans le chapitre Enterprise Service Bus.

Glossaire

API Application Programming Interface : Ensemble des méthodes exposées par une application permettant aux autres applications de communiquer avec elle. .

CIFS Common Internet File System : protocole de système de fichiers utilisé pour fournir un accès partagé aux fichiers et aux imprimantes entre les machines d'un réseau..

CSV Comma-Separated Values : fichier texte de stockage de données sous forme tabulaire. Chaque ligne d'un fichier CSV correspond à un enregistrement et chaque élément de la ligne, séparé par un caractère pré-défini, représente un champ de cette enregistrement .

DSA Data Staging Area : Dans une processus ETL, une DSA est une zone de stockage intermédiaire de données situées entre les données sources et le datawarehouse.Les données stockées dans cette zone le sont souvent de façon temporaire pour y subir divers traitements..

ERP Enterprise Resource Planning : Système informatique intégré permettant la gestion de l'ensemble de l'information et des ressources nécessaires à l'exploitation d'une entreprise.

FTP File Transfer Protocol : Protocole standardisé de transfert de fichiers entre ordinateurs sur un réseau d'ordinateurs.

HDFS Hadoop Distributed File System : Système de fichiers distribué permettant de stocker et récupérer des fichiers à haute performance `https://hadoop.apache.org/`.

HIS Hospital Information System : Un système d'information hospitalier est un système technologique complet qui permet aux hôpitaux de gérer tous les aspects de leurs opérations, et pas seulement l'information des patients. Il peut être considéré comme un système qui aide tous les services hospitaliers à gérer leur information afin de leur permettre de faire leur travail efficacement. Les systèmes d'information hospitaliers gèrent non seulement l'information sur les patients, mais aussi l'information financière et commerciale..

HTTP Hyper Text Transfer Protocol : Protocole de niveau application pour les systèmes d'information hypermédia distribués et collaboratifs. Il s'agit d'un protocole générique, sans état, qui peut être utilisé pour de nombreuses tâches au-delà de son utilisation pour l'hypertexte, comme les serveurs de noms et

les systèmes de gestion d'objets distribués, en étendant ses méthodes de requête, ses codes d'erreur et ses en-têtes. Une caractéristique de HTTP est la saisie et la négociation de la représentation des données, permettant aux systèmes d'être construits indépendamment des données transférées https://www.w3.org/Protocols/rfc2616/rfc2616.html.

JSON JavaScript Object Notation : Format léger d'échange de données. Il est facile à lire et à écrire pour les humains. Il est facile à analyser et à générer pour les machines. Il est basé sur un sous-ensemble du langage de programmation JavaScript, Standard ECMA-262 3e édition - décembre 1999. JSON est un format texte complètement indépendant du langage mais qui utilise des conventions familières aux programmeurs de la famille des langages C, y compris C, C++, C#, Java, Java, JavaScript, Perl, Python et bien d'autres https://www.w3.org/Protocols/rfc2616/rfc2616.html.

MOM Message Oriented Middelware : Concept qui implique le transfert de données entre applications en utilisant un canal de communication qui transporte des unités d'information autonomes (messages). Dans un environnement de communication basé sur MOM, les messages sont généralement envoyés et reçus de manière asynchrone. Grâce aux communications basées sur les messages, les applications sont découplées de manière abstraite ; les expéditeurs et les destinataires ne sont jamais au courant les uns des autres. Au lieu de cela, ils envoient et reçoivent des messages à destination et en provenance du système de messagerie. Il incombe au système de messagerie (MOM) d'acheminer les messages à leur destination prévue..

Recip-e Système qui permet la délivrance de prescriptions électroniques en matière de soins de santé. Il fait partie du plan e-Santé initié en 2014 par le gouvernement fédéral belge, soucieux de se conformer aux directives européennes.
Recip-e permet à différents prescripteurs de soins (médecins, dentistes, sages-femmes) d'envoyer de manière électronique et sécurisée des prescriptions vers un serveur. Elles y sont encodées et conservées jusqu'à ce qu'elles soient utilisées par le patient auprès d'un prestataire de soins (pharmacien, kinésithérapeute, infirmier). L'envoi et le retrait s'opèrent à l'aide de logiciels spécialisés sur lesquels Recip-e est installé https://recip-e.be.

REST REpresentational State Transfer : Style architectural pour les systèmes hypermédia distribués, présenté pour la première fois par Roy Fielding en 2000 dans sa célèbre thèse. Comme tout autre style architectural, REST a aussi ses contraintes (client-serveur, sans état, avec cache, interface uniforme, système en couches et code à la demande) de guidage qui doivent être satisfaites si une interface doit être appelée RESTful https://restfulapi.net/.

RPC Remote Procedure Call : Modèle ou technique pour construire des applications distribuées, client-serveur. Il est basé sur l'extension de l'appel de procédure locale conventionnelle, de sorte que la procédure appelée n'a pas besoin d'exister dans le même espace adresse que la procédure appelante. Les deux processus peuvent se trouver sur le même système ou sur des systèmes différents avec un réseau qui les relie..

SCP Secure Copy Protocol : protocole réseau, basé sur le protocole BSD RCP, qui supporte les transferts de fichiers entre hôtes sur un réseau. SCP utilise Secure Shell (SSH) pour le transfert de données et utilise les mêmes mécanismes d'authentification, garantissant ainsi l'authenticité et la confidentialité des données en transit. Un client peut envoyer des fichiers vers un serveur, en incluant éventuellement leurs attributs de base (permissions, horodatages). Les clients peuvent également demander des fichiers ou des répertoires à partir d'un serveur.

SFTP SSH File Transfer Protocol : Protocole sécurisé de transfert de fichiers. Il fonctionne sur le protocole SSH. Il prend en charge toutes les fonctionnalités de sécurité et d'authentification de SSH. SFTP a quasiment remplacé l'ancien protocole FTP en tant que protocole de transfert de fichiers, et remplace rapidement FTP/S. Il fournit toutes les fonctionnalités offertes par ces protocoles, mais de manière plus sûre et plus fiable, avec une configuration plus facile.

SMB Server Message Block : Protocole réseau utilisé par les ordinateurs Windows qui permet aux systèmes d'un même réseau de partager des fichiers.

SOA Service Oriented Architecture : Style architectural qui soutient l'orientation services. Celle-ci est une façon de penser en termes de services et de développement basé sur le service et les résultats des services.
Un service est une représentation logique d'une activité commerciale reproductible qui a un résultat précis (p. ex. vérifier le crédit du client, fournir des données météorologiques, consolider les rapports de forage), est autonome, peut être composé d'autres services et est une "boîte noire" pour les consommateurs du service `https://www.opengroup.org/soa/source-book/soa/p1.htm`.

SOAP Simple Object Access Protocol : Protocole XML léger pour l'échange d'informations dans un environnement décentralisé et distribué `https://www.w3.org/TR/soap/`.

SPOF Single Point Of Failure : Partie d'un système qui, s'il cesse de fonctionner, entraîne l'arrêt du système entier.

TCO Total cost of ownership - Calcul financier estimant les coûts directs et indirects d'un produit tout au long de son cycle de vie.

URI Uniform Resource Identifier - Un URI est une chaîne de caractère identifiant sans ambiguité une resource..

XML Extensible Markup Language : Format texte simple et très flexible dérivé du SGML (ISO 8879). Conçu à l'origine pour relever les défis de la publication électronique à grande échelle, XML joue également un rôle de plus en plus important dans l'échange d'une grande variété de données sur le Web et ailleurs `https://www.w3.org/XML/`.

XPath XML Path Language : Langage d'adressage de parties d'un document XML, conçu pour être utilisé par XSLT et XPointer `https://www.w3.org/TR/xpath-31/`.

XQuery XML Query Language : Langage de développement d'applications et de requêtes polyvalent, capable de traiter le contenu informationnel de diverses sources de données, y compris des documents structurés et semi-structurés,

des bases de données relationnelles et arborescentes. Le langage XQuery est conçu pour prendre en charge de puissantes optimisations et pré-compilations conduisant à des recherches très efficaces sur de grandes quantités de données, y compris sur des bases de données dites XML-natives qui lisent et écrivent du XML mais ont un stockage interne efficace `https://www.w3.org/TR/xquery/all/`.

XSLT XSL Transformations : Langage de transformation de documents XML. Il décrit comment transformer (changer) la structure d'un document XML (Extensible Markup Language) en un document XML avec une structure différente. `https://www.w3.org/TR/xslt/all/`.

Bibliographie

[1] Lachlan Aldred et al. “On the Notion of Coupling in Communication Middleware”. In : *On the Move to Meaningful Internet Systems 2005 : CoopIS, DOA, and ODBASE*. Sous la dir. de David Hutchison et al. T. 3761. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, p. 1015-1033. isbn : 978-3-540-29738-3. doi : 10.1007/11575801{\textunderscore}6.

[2] Gustavo Alonso et al. *Web Services : Concepts, Architectures and Applications*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004. isbn : 978-3-642-07888-0. doi : 10.1007/978-3-662-10876-5.

[3] Yaacov Apelbaum. *To ESB or not to ESB is the question. Or is it more noble to just ETL it ?* 2012. url : https://apelbaum.wordpress.com/2012/09/30/to-esb-or-not-to-esb-is-the-question-or-is-it-more-noble-to-just-etl-it/ (visité le 12/05/2019).

[4] Nuwan Bandara. *Blogs WSO2 : Orchestration and Choreography – When To Use An ESB vs a Workflow Engine*. 2016. url : https://wso2.com/blogs/thesource/2016/03/orchestration-and-choreography-when-to-use-an-esb-vs-a-workflow-engine/.

[5] Don Batory. “Feature Models, Grammars, and Propositional Formulas”. In : *Software Product Lines*. Sous la dir. de David Hutchison et al. T. 3714. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer Berlin Heidelberg, 2005, p. 7-20. isbn : 978-3-540-28936-4. doi : 10.1007/11554844{\textunderscore}3.

[6] Ridha Mohammed Benosman. “Conception et évaluation de performance d’un Bus applicatif, massivement parallèle et orienté service”. Thèse de doct. Conservatoire national des arts et metiers, 2014.

[7] Robin Singh Bhadoria, Narendra S. Chaudhari et V. G. Tharinda Nishantha Vidanagama. “Analyzing the role of interfaces in enterprise service bus : A middleware epitome for service-oriented systems”. In : *Computer Standards & Interfaces* 55 (2018), p. 146-155. issn : 09205489. doi : 10.1016/j.csi.2017.08.001.

[8] Robin Singh Bhadoria, Narendra S. Chaudhari et Geetam Singh Tomar. “The Performance Metric for Enterprise Service Bus (ESB) in SOA system : Theoretical underpinnings and empirical illustrations for information processing”. In : *Information Systems* 65 (2017), p. 158-171. issn : 03064379. doi : 10.1016/j.is.2016.12.005.

[9] Justus Bogner et Alfred Zimmermann. “Towards Integrating Microservices with Adaptable Enterprise Architecture”. In : *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 5/09/2016 - 09/09/2016, p. 1-6. isbn : 978-1-4673-9933-3. doi : 10.1109/EDOCW.2016.7584392.

[10] Valérie Botta-Genoulaz et Pierre-Alain Millet. “An investigation into the use of ERP systems in the service sector”. In : *International Journal of Production Economics* 99.1-2 (2006), p. 202-221. issn : 09255273. doi : 10.1016/j.ijpe.2004.12.015.

[11] Martin Breest et Roy Schulte. “An Introduction to the Enterprise Service Bus”. In : (2006).

[12] Bruce Silver Associates, éd. *Enterprise Service Bus Technology for Real-World Solutions*. 2004. url : ftp://public.dhe.ibm.com/software/websphere/JavaDevTools/Resources/EnterpriseServiceBus-WebSphere-WBI-ESB-WhitePaper-20040815.pdf (visité le 21/11/2018).

[13] David A.. Chappell. *Enterprise service bus*. 1st ed. Sebastopol, Calif. : O’Reilly, 2004. isbn : 978-0596006754.

[14] Patrick Y.K. Chau. “Factors used in the selection of packaged software in small businesses : Views of owners and managers”. In : *Information & Management* 29.2 (1995), p. 71-78. issn : 03787206. doi : 10.1016/0378-7206(95)00016-P.

[15] Santiago Comella-Dorda et al. *A Process for COTS Software Product Evaluation*. A Process for COTS Software Product Evaluation, 2004. url : http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6701.

[16] Eric Cooper. *Distributed Systems Technology Survey*. 1987.

[17] Florian Daniel et al. “Understanding UI Integration : A Survey of Problems, Technologies, and Opportunities”. In : *IEEE Internet Computing* 11.3 (2007), p. 59-66. issn : 1089-7801. doi : 10.1109/MIC.2007.74.

[18] Jana Debasish. “Service Oriented Architecture – A New Paradigm”. In : *CSI Communications* (2006).

[19] Ivan M. Delamer et Jose L. Martinez Lastra. “Loosely-coupled Automation Systems using Device-level SOA”. In : *2007 5th IEEE International Conference on Industrial Informatics*. IEEE, 23/06/2007 - 27/06/2007, p. 743-748. isbn : 978-1-4244-0850-4. doi : 10.1109/INDIN.2007.4384866.

[20] Kiran Doshi. *Enterprise Service Bus : White paper*. 2009.

[21] Zineb El Akkaoui et al. “A model-driven framework for ETL process development”. In : *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP - DOLAP ’11*. Sous la dir. d’Il-Yeol Song, Alfredo Cuzzocrea et Karen Davis. New York, New York, USA : ACM Press, 2011, p. 45. isbn : 9781450309639. doi : 10.1145/2064676.2064685.

[22] Thomas. Erl. *Service-oriented architecture : Concepts, technology, and design*. New York, Boston et London : Prentice Hall, 2005. isbn : 0131858580.

[23] Patrick Th. Eugster et al. “The many faces of publish/subscribe”. In : *ACM Computing Surveys* 35.2 (2003), p. 114-131. issn : 03600300. doi : 10.1145/857076.857078.

[24] José A. Galindo et al. “Automated analysis of feature models : Quo vadis ?” In : *Computing* 101.5 (2019), p. 387-433. issn : 0010-485X. doi : 10.1007/s00607-018-0646-1.

[25] F. J. Garcia-Jimenez, M. A. Martinez-Carreras et A. F. Gomez-Skarmeta. “Evaluating Open Source Enterprise Service Bus”. In : *2010 IEEE 7th International Conference on E-Business Engineering*. IEEE, 2010, p. 284-291. isbn : 978-1-4244-8386-0. doi : 10.1109/ICEBE.2010.12.

[26] *gRPC : A high performance, open-source universal RPC framework*. url : https://grpc.io/.

[27] Thomas Gulledge. “What is integration ?” In : *Industrial Management & Data Systems* 106.1 (2006), p. 5-20. issn : 0263-5577. doi : 10.1108/02635570610640979.

[28] Colombe Hérault et Gaël Thomas. “Mediation and Enterprise Service Bus : A position paper”. In : *Proceedings of the First International 2005*. 2005.

[29] IBM Corporation. *The ESB architectural pattern*. 2009.

[30] Anil S. Jadhav et Rajendra M. Sonar. “Evaluating and selecting software packages : A review”. In : *Information and Software Technology* 51.3 (2009), p. 555-563. issn : 09505849. doi : 10.1016/j.infsof.2008.09.003.

[31] Martin Keen et al. *Patterns : Implementing an SOA using an Enterprise Service Bus*. 1st ed. IBM redbooks. Research Triangle Park NC : IBM International Technical Support Organization, 2004. isbn : 9780738490007.

[32] Michal Kokorceny. “A new cost model for comparison of Point to Point andEnterprise Service Bus integration styles”. In : *Recent advances in information technology*. Sous la dir. de Nikos E. Mastorakis et Josip Musić. Recent advances in computer engineering series. [Athens] : WSEAS Press, 2014, p. 63-70. isbn : 978-1-61804-264-4. url : http://www.wseas.us/e-library/conferences/2014/Geneva/ECCS/ECCS-10.pdf (visité le 01/01/2019).

[33] Jyrki Kontio. “A case study in applying a systematic method for COTS selection”. In : *ICSE ’96 : Proceedings of the 18th international conference on Software engineering*, p. 201-209.

[34] Veit Köppen, Björn Brüggemann et Bettina Berendt. “Designing Data Integration : The ETL Pattern Approach”. In : *The European Journal for the Informatics Professional* 12.3 (2011), p. 49-55.

[35] Jürgen Kress et al. *Enterprise Service Bus*. 2013. url : https://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.html (visité le 19/11/2018).

[36] David Kusak. “Comparison of Enterprise Application Integration Platforms”. Master. Prague : Charles University, 2010.

[37] Ananias Laftsidis. *Chapter 15 Enterprise Application Integration*. 2011.

[38] Lakshminarayan Srinivasan et Jem Treadwell. “An Overview of Service-oriented Architecture Web Services and Grid Computing”. In : 2005. url : https://pdfs.semanticscholar.org/7f5a/c1b5abcfd552b63777c9ad2687a84a78543a.pdf (visité le 04/08/2019).

[39] G. Larry Sanders, Parviz Ghandforoush et Larry M. Austin. “A model for the evaluation of computer software packages”. In : *Computers & Industrial Engineering* 7.4 (1983), p. 309-315. issn : 03608352. doi : 10.1016/0360-8352(83)90014-1.

[40] P. K. Lawlis et al. “A formal process for evaluating COTS software products”. In : *Computer* 34.5 (2001), p. 58-63. issn : 00189162. doi : 10.1109/2.920613.

[41] Guzmán Llambías, Laura González et Raúl Ruggia. “Towards an Integration Platform for Bioinformatics Services”. In : *Service-Oriented Computing – ICSOC 2013 Workshops*. Sous la dir. de David Hutchison et al. T. 8377. Lecture Notes in Computer Science. Cham : Springer International Publishing, 2014, p. 445-456. isbn : 978-3-319-06858-9. doi : 10.1007/978-3-319-06859-6{\textunderscore}39.

[42] Thomas W. Malone et Kevin Crowston. “The interdisciplinary study of coordination”. In : *ACM Computing Surveys* 26.1 (1994), p. 87-119. issn : 03600300. doi : 10.1145/174666.174668.

[43] Consultant :Bernard Manouvrier et Consultant :Laurent Menard. *Application Integration : EAI B2B BPM and SOA*. John Wiley & Sons, 2008. isbn : 978-1-84821-088-2.

[44] Izet Masic. “A Review of Informatics and Medical Informatics History”. In : *Acta Informatica Medica* 15 (2007), p. 178-188.

[45] Tomasz Masternak et al. “ESB-Modern SOA Infrastructure”. In :

[46] Falko Menge. “Enterprise Service Bus”. In : *United Nations Development Program and Association of the Units of the Local Self-Government 2007 – Conference on Free Open Source*.

[47] Xueqiang Mi et al. “Multifactor-Driven Hierarchical Routing on Enterprise Service Bus”. In : *Web Information Systems and Mining*. Sous la dir. de David Hutchison et al. T. 5854. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, p. 328-336. isbn : 978-3-642-05249-1. doi : 10.1007/978-3-642-05250-7{\textunderscore}35.

[48] Denis Morand, Issac Garcia et Philippe Lalanda. “Autonomic enterprise service bus”. In : *ETFA2011*. IEEE, 2011, p. 1-8. isbn : 978-1-4577-0017-0. doi : 10.1109/ETFA.2011.6059231.

[49] David Morera. “COTS Evaluation Using Desmet Methodology & Analytic Hierarchy Process (AHP)”. In : *Product Focused Software Process Improvement*. Sous la dir. de Gerhard Goos et al. T. 2559. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer Berlin Heidelberg, 2002, p. 485-493. isbn : 978-3-540-00234-5. doi : 10.1007/3-540-36209-6{\textunderscore}40.

[50] M. Morisio et A. Tsoukiàs. “IusWare : a methodology for the evaluation and selection of software products”. In : *IEE Proceedings - Software Engineering* 144.3 (1997), p. 162. issn : 0895-7177. doi : 10.1049/ip-sen:19971350.

[51] Shrikant Mulik. “Using Enterprise Service Bus (ESB) for connecting corporate functions and shared services with business divisions in a large enterprise”. In : *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*. IEEE, 2009, p. 430-434. isbn : 978-1-4244-5338-2. doi : 10.1109/APSCC.2009.5394092.

[52] Oracle. *BEA Aqualogic Service Bus : Concept and architecture*. 2005. URL : https://docs.oracle.com/cd/E13171_01/alsb/docs21/concepts/overview.html#1083121.

[53] Michael P. Papazoglou. "What's in a Service ?" In : *Software Architecture*. Sous la dir. de Flavio Oquendo. T. 4758. Lecture Notes in Computer Science. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007, p. 11-28. ISBN : 978-3-540-75131-1. DOI : 10.1007/978-3-540-75132-8{\textunderscore}3.

[54] Mike P. Papazoglou et Willem-Jan van den Heuvel. "Service oriented architectures : approaches, technologies and research issues". In : *The VLDB Journal* 16.3 (2007), p. 389-415. ISSN : 1066-8888. DOI : 10.1007/s00778-007-0044-3.

[55] Cesare Pautasso, Olaf Zimmermann et Frank Leymann. "Restful web services vs. "big"' web services". In : *Proceeding of the 17th international conference on World Wide Web - WWW '08*. Sous la dir. de Jinpeng Huai et al. New York, New York, USA : ACM Press, 2008, p. 805. ISBN : 9781605580852. DOI : 10.1145/1367497.1367606.

[56] Tijs. Rademakers et Jos. Dirksen. *Open source ESBs in action : Example implementations in Mule and ServiceMix*. Greenwich, CT et London : Manning et Pearson Education [distributor], 2009. ISBN : 1933988215.

[57] Suja Radha et Rama Krishna Rao Bandaru. "Taxonomy construction techniques - issues and challenges". In : *Civil Eng* 2 (2011).

[58] D. Risimic. "An integration strategy for large enterprises". In : *Yugoslav Journal of Operations Research* 17 (2007), p. 209-222. DOI : 10.2298/YUJOR0702209R. URL : http://elib.mi.sanu.ac.rs/files/journals/yjor/34/yujorn34p209-222.pdf.

[59] Arnon Rotem-Gal-Oz. *Fallacies of Distributed Computing*. 2006.

[60] William A. Ruh, Francis X. Maginnis et William J. Brown. *Enterprise Application Integration : A Wiley Tech Brief*. Hoboken : Wiley, 2002. ISBN : 9780471437864.

[61] S. Masoud Sadjadi. "A Survey of Adaptive Middleware". In : (2003).

[62] M.-T. Schmidt et al. "The Enterprise Service Bus : Making service-oriented architecture real". In : *IBM Systems Journal* 44.4 (2005), p. 781-797. ISSN : 0018-8670. DOI : 10.1147/sj.444.0781.

[63] Themba Shezi et al. "Analysis of Open Source Enterprise Service Buses toward Supporting Integration in Dynamic Service Oriented Environments". In : *e-Infrastructure and e-Services for Developing Countries*. Sous la dir. de Karl Jonas, Idris A. Rai et Maurice Tchuente. T. 119. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013, p. 115-125. ISBN : 978-3-642-41177-9. DOI : 10.1007/978-3-642-41178-6{\textunderscore}12.

[64] Zeeshan Siddiqui, Abdul Hanan Abdullah et Muhammad Khurram Khan. "Qualified Analysis b/w ESB(s) Using Analytical Hierarchy Process (AHP) Method". In : *2011 Second International Conference on Intelligent Systems, Modelling and Simulation*. IEEE, 25/01/2011 - 27/01/2011, p. 100-104. ISBN : 978-1-4244-9809-3. DOI : 10.1109/ISMS.2011.25.

[65] Tariq Soomro et Abrar Awan. "Challenges and Future of Enterprise Application Integration". In : *International Journal of Computer Applications* 42 (2012), p. 975-8887. DOI : 10.5120/5708-7762.

[66] Edward Stohr et Jeffrey Nickerson. “Intra Enterprise Integration : Methods and Direction”. In : (2002).

[67] Longji Tang et al. “Modeling enterprise service-oriented architectural styles”. In : *Service Oriented Computing and Applications* 4.2 (2010), p. 81-107. issn : 1863-2386. doi : 10.1007/s11761-010-0059-2.

[68] Kardi Teknomo. *Analytical Hierarchy Process (AHP) tutorial*.

[69] *The state of API integration : Report*. 2017. url : https://offers.cloud-elements.com/the-state-of-api-integrations-report-2017-download.

[70] Marinos Themistocleous, Zahir Irani et Robert M. O’Keefe. “ERP and application integration”. In : *Business Process Management Journal* 7.3 (2001), p. 195-204. issn : 1463-7154. doi : 10.1108/14637150110392656.

[71] Manoochehr Toshtzar. “Multi-criteria decision making approach to computer software evaluation : Application of the analytical hierarchy process”. In : *IEE Proceedings - Software Engineering* 11 (1988), p. 276-281. issn : 0895-7177. doi : 10.1016/0895-7177(88)90498-0.

[72] Evangelos Triantaphyllou et Stuart H. Mann. “Using the analytic hierarchy process for decision making in engineering applications : Some challenges”. In : *Inter’l Journal of Industrial Engineering : Applications and Practice* 2 (1995), p. 35-44.

[73] Mohammad Hadi Valipour et al. “A brief survey of software architecture concepts and service oriented architecture”. In : *2009 2nd IEEE International Conference on Computer Science and Information Technology*. IEEE, 8/08/2009 - 11/08/2009, p. 34-38. isbn : 978-1-4244-4519-6. doi : 10.1109/ICCSIT.2009.5235004.

[74] Panos Vassiliadis, Alkis Simitsis et Spiros Skiadopoulos. “Conceptual modeling for ETL processes”. In : *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP - DOLAP ’02*. Sous la dir. d’Il-Yeol Song et Dimitri Theodoratos. New York, New York, USA : ACM Press, 2002, p. 14-21. isbn : 1581135904. doi : 10.1145/583890.583893.

[75] Mark Velasquez et Patrick Hester. “An analysis of multi-criteria decision making methods”. In : *International Journal of Operations Research* 10 (2013), p. 56-66.

[76] Ken Vollmer, Like Gilpin et Sander Rose. *The Forrester Wave™ : Enterprise Service Bus, Q2 2011*. Sous la dir. de Forrester. 2011.

[77] Dieter Welzel et Hans-Ludwig Hausen. “A five step method for metric-based software evaluation — effective software metrication with respect to quality standards”. In : *Microprocessing and Microprogramming* 39.2-5 (1993), p. 273-276. issn : 01656074. doi : 10.1016/0165-6074(93)90104-S.

[78] Eberhard Wolff. *Microservices : Flexible software architecture*. Boston : Addison-Wesley, 2017. isbn : 978-0-134-60241-7.

[79] Bobby. Woolf et Gregor. Hohpe. *Enterprise integration patterns : Designing, building, and deploying messaging solutions*. The Addison-Wesley signature series. Boston : Addison-Wesley, 2004. isbn : 978-0321200686.

[80] Robert Woolley. *Enterprise Service Bus (ESB) Product Evaluation Comparisons*. 2006.

[81] *WSO2 Enterprise Service Bus*. URL : https://wso2.com/products/enterprise-service-bus/ (visité le 21/10/2019).

[82] Chen Wu et Elizabeth Chang. “An Analysis of Web Services Mediation Architecture and Pattern in Synapse”. In : *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. IEEE, 2007, p. 1001-1006. ISBN : 0-7695-2847-3. DOI : 10.1109/AINAW.2007.76.

[83] Jieming Wu et Xiaoli Tao. “Research of enterprise application integration based-on ESB”. In : *2010 2nd International Conference on Advanced Computer Control*. IEEE, 27/03/2010 - 29/03/2010, p. 90-93. ISBN : 978-1-4244-5845-5. DOI : 10.1109/ICACC.2010.5487292.

[84] Feng Xinyang, Shen Jianjing et Fan Ying. “REST : An alternative to RPC for Web services architecture”. In : *2009 First International Conference on Future Information Networks*. IEEE, 14/10/2009 - 17/10/2009, p. 7-10. ISBN : 978-1-4244-5158-6. DOI : 10.1109/ICFIN.2009.5339611.

[85] Olaf Zimmermann et al. “A Decade of Enterprise Integration Patterns : A Conversation with the Authors”. In : *IEEE Software* 33.1 (2016), p. 13-19. ISSN : 0740-7459. DOI : 10.1109/MS.2016.11.

[86] Răzvan Daniel Zota, Radu Stefan Moleavin et Laurenłiu Ciovică. “Enterprise Service Bus - A Backbone for SOA”. In : *Recent researches in business & economics*. Sous la dir. de Zeljko Panian. [Greece] : WSEAS, 2012, p. 241-244. ISBN : 9781618041029.