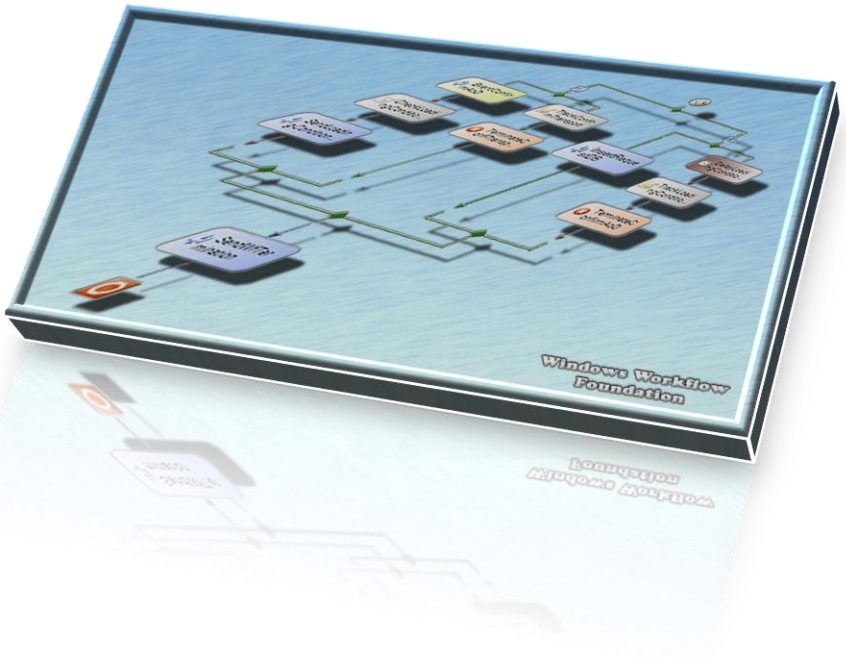
Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Travail de diplôme 2007

Filière Informatique de gestion

Microsoft Windows Workflow Foundation



Etudiant : Stéphane Probst

Professeur : Laurent Bagnoud

www.hevs.ch

I. PREFACE

Le domaine de l'informatique est un univers en constante évolution, autant dans les nouvelles technologies que dans les concepts d'implémentation logiciel. Microsoft, un des leaders du développement logiciel a sorti en fin d'année 2006 son nouveau modèle de génération de code Windows appelé Framework 3.0 ou encore WinFx. Ce Framework contient de nouveaux composants permettant la gestion de processus métiers. Le terme gestion de processus métiers n'est autre que la traduction française pour Business Process Management (BPM), qui est actuellement sur toutes les lèvres dans le domaine de la conception logicielle. En effet, le BPM apparaît comme l'une des solutions permettant d'optimiser les performances des entreprises, tout en diminuant les coûts. La finalité du BPM est d'automatiser et de fluidifier les activités et les échanges entre collaborateurs et services. De ce fait le sujet proposé, consistant à implémenter un scénario d'une chaîne logistique avec la nouvelle technologie de Microsoft Windows Workflow Foundation, est un des thèmes de discussion de l'actualité informatique et des prochaines années.

Dès lors, vous comprendrez aisément ma motivation à réaliser mon travail de diplôme sur cette nouvelle technologie. De plus, l'implémentation de cette chaîne logistique implique l'implémentation de services de communication entre les différents acteurs de la chaîne. Le mot communication, dans le domaine informatique, me fait tout de suite penser à SOA (Service Oriented Application) qui est un autre thème far du moment pour les entreprises agiles. Pour résumer, ce travail touche tous les thèmes de la vague de développement logiciel.

Structure du document

Ce document présente le travail réalisé sur l'utilisation d'une nouvelle technologie développée par Microsoft permettant l'implémentation d'applications orientées processus. Le but de ce point est de guider le lecteur à travers le document en présentant les chapitres.

Le document présente un résumé du travail effectué, page 4. Il renseigne tout d'abord sur l'objectif à atteindre puis définit la problématique du sujet et présente les étapes du travail réalisé tout au long de ce projet. Finalement il présente une synthèse des résultats obtenus.

Page 8 la table des matières permet de se guider au travers du document. Elle permet de retrouver un titre jusqu'à quatre niveaux de profondeur.

Directement après la table des matières, page 11, l'introduction permet de prendre connaissance en détail du sujet abordé, ainsi que du contexte dans lequel il figure. La problématique y est abordée ainsi que la présentation des résultats obtenus. Afin de mieux cerner les enjeux de la technologie, l'état de l'art actuel du domaine est présenté, page 12.

Avant d'aborder le développement du travail réalisé, un chapitre présente, page 19, le scénario de la chaîne logistique dans lequel cette technologie devra être utilisée. Il présente de manière détaillée le domaine de la chaîne, les acteurs, ainsi que les différents flux de communication entre eux.

Le document poursuit avec le chapitre du développement, page 19, celui-ci comprend deux axes distincts. Le premier présente la technologie dans son ensemble ; ses concepts, ses composants ainsi que ses possibilités. Le deuxième axe présente la solution développée de l'implémentation de la chaîne logistique. Le premier chapitre présente l'architecture de la solution ainsi que les modules des logiciels développés. Le deuxième chapitre est un chapitre technique. Il s'adresse aux personnes ayant des connaissances de la programmation DotNet et présente les différentes technologies utilisées et les concepts techniques de l'implémentation de la solution.

La synthèse, page 91 présente une discussion sur les résultats obtenus. Ce chapitre présente également un compte rendu des points développés par rapport au cahier des charges déterminé en début de projet.

Finalement la conclusion, page 93 présente les apports de cette technologie. Les appendices, page 98 contiennent divers délivrables du projet ainsi que les documents utilisés tout au long du projet. Pour la liste complète des documents vous pouvez vous référer à la table des matières. La bibliographie, page 94 présente toutes les sources employées durant le projet.

II. RESUME

But du travail

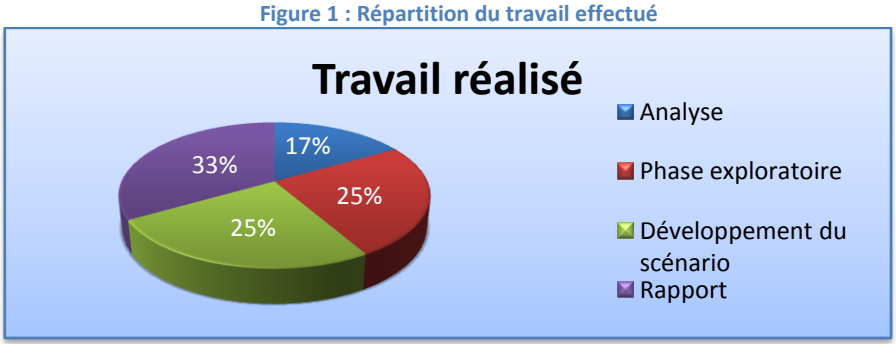
Le travail à réaliser ayant pour but d'implémenter un scénario d'une chaîne logistique avec la technologie Windows Workflow Foundation a deux objectifs. Le premier consiste à découvrir les possibilités ainsi que les fonctionnalités de la technologie. Le deuxième objectif est d'établir une application prototype du scénario, celle-ci servira d'élément de démonstration de la technologie.

Problématique

Pour pouvoir atteindre ces objectifs la technologie doit permettre une certaine souplesse et certaines fonctionnalités sont requises. Tout d'abord, elle doit proposer un outil permettant la modélisation des processus des acteurs du scénario. Cet outil doit permettre de voir graphiquement toutes les étapes définissant le processus. D'autre part, elle doit contenir un module permettant la gestion des processus. Finalement elle doit assurer l'interopérabilité et la communication des processus avec d'autres technologies ou d'autres applications.

Travail effectué

Figure 1 : Répartition du travail effectué



Afin d'atteindre ces objectifs, le déroulement du travail s'est effectué en quatre phases, selon le graphique ci-dessus. La réalisation de ce projet comptabilise un travail de 570 heures réparties entre ces différentes phases.

Analyse

Le travail a commencé par une phase de deux semaines durant laquelle les bases du travail ont été posées. Elle regroupe la compréhension du travail à effectuer, la rédaction du cahier des charges, les prototypes de l'architecture de la solution ainsi que la planification pour le suivi du projet. Afin d'optimiser le rendement du travail et d'avoir une vue d'ensemble du travail à réaliser, la planification est faite dans MS

Project. Le cahier des charges, la planification ainsi que les prototypes du projet se trouvent en appendices, page 98.

Phase exploratoire

Les trois semaines suivantes ont été dédiées à la découverte technologique. Cette phase a débuté par une recherche sur les différents composants de la technologie ainsi que de ses possibilités d'intégration. Finalement elle se termine par la recherche de codes sources pour l'implémentation des différents modules exigés par le scénario de la chaîne logistique.

Développement du scénario

Durant trois semaines, cette phase a permis l'élaboration du prototype intégrant la technologie sur la base du scénario de la chaîne logistique. Le développement des services de communication est le premier module élaboré, il est suivi par le développement du module de gestion des processus. L'implémentation continue par le développement de l'application d'administration et finalement par l'application business.

Rapport

La rédaction du rapport s'est déroulée parallèlement à la phase exploratoire et au développement du scénario.

Résultats obtenus

Après les douze semaines à disposition pour la réalisation du projet les objectifs sont atteints. La découverte technologique est consignée dans le rapport et l'implémentation de l'application prototype sur la base du scénario imposé a pu être réalisée avec cette nouvelle technologie.

III. ZUSAMMENFASSUNG

Zweck der Arbeit

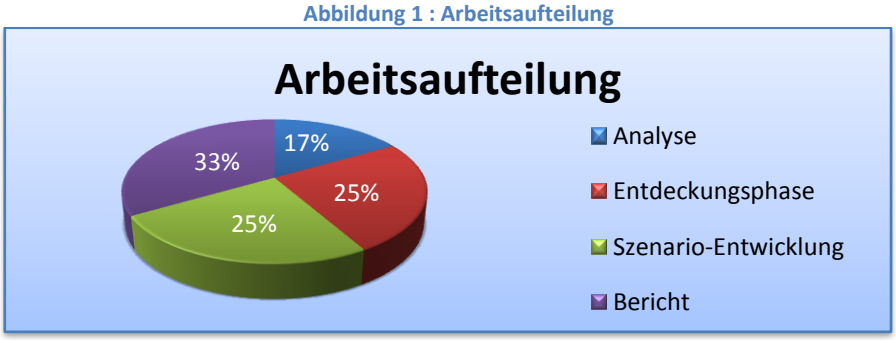
Diese Arbeit bezweckt die Implementierung einer Logistikkette mit der neuen Technologie Microsoft Windows Workflow Foundation und hat zwei Ziele. Erstens, die Aufdeckung der verschiedenen Möglichkeiten und Funktionen dieser Technologie. Zweitens, die Realisierung einer prototypischen Anwendung des Szenarios als Anwendungsbeispiel der Technologie.

Problematik

Zur Erfüllung dieser Ziele muss die Technologie eine gewisse Flexibilität und bestimmte Funktionen besitzen. Erstens, muss sie ein Werkzeug zur Modellierung der Prozesse haben. Dieses Werkzeug soll alle Schritte des Verfahrens visuell darstellen. Zweitens, muss sie ein Modul für die Prozess-Verwaltung beinhalten. Schliesslich muss diese Technologie die Interoperabilität und die Kommunikation der Prozesse mit anderen Technologien oder Anwendungen ermöglichen.

Arbeitsaufteilung

Abbildung 1 : Arbeitsaufteilung



Um diese Ziele zu erreichen, wurde die Arbeit in vier Phasen aufgeteilt (siehe Grafik). Für die Realisierung dieses Projektes wurden insgesamt 570 Stunden benötigt.

Analyse

Die Arbeit hat mit einer Planungsphase von zwei Wochen begonnen. Diese Phase beinhaltet das Verständnis der Arbeit, die Zielfestlegung, die Prototypen der Lösungsarchitektur und die Planung der Arbeit. Um eine bessere Überblick und eine optimale Zeitverbrauch zu ermöglichen, wurde die Planung in MS Project gemacht. Das Lastenheft, die Planung der Arbeit sowie die Prototypen des Projekts befinden sich in Seite 98 des Anhangs.

Entdeckungsphase

Die nächsten drei Wochen waren der Entdeckung der Technologie gewidmet. Diese Phase hat mit der Suche der verschiedenen Technologiekomponenten, sowie derer Integrationsmöglichkeiten begonnen. Schliesslich endete diese Phase mit der Suche von Quellenkoden für die Implementierung der verschiedenen Module, welche es das Logistikkette-Szenario verlangt.

Szenario-Entwicklung

Diese Phase, welche drei Wochen gedauert hat, erlaubte die Entwicklung des Prototyps gemäss Szenariodefinition. Das erste entwickelte Modul beschäftigt sich mit den Kommunikationsdienstleistungen. Das Prozessverwaltungsmodul wurde als zweites entwickelt. Die letzte Phase war der Entwicklung der Verwaltungs- und Businessanwendungen gewidmet.

Bericht

Der Bericht wurde parallel zur Entdeckungsphase und Szenario-Entwicklung verfasst.

Ergebnisse

Nach den zwölf zur Verfügung stehenden Wochen sind alle Ziele erreicht. Die Technologie wird im Bericht erklärt und die prototypische Applikation wurde realisiert.

IV. TABLE DES MATIERES

V. INTRODUCTION

Comme présenté dans la préface, le développement orienté processus est un des concepts au centre du développement logiciel. L'une des principales priorités des responsables informatiques est l'amélioration des processus métier facilitant ainsi les relations entre les entreprises. Les dirigeants recherchent de quelle manière l'informatique pourrait améliorer leur compétitivité et leur position sur les marchés de plus en plus concurrentiels. Afin d'y remédier, ils tentent d'exploiter les outils informatiques. Les entreprises doivent s'adapter rapidement à l'évolution des marchés. Cette dynamique peut être améliorée par l'intégration des personnes, des applications et de l'information aux processus métier.

Ce travail est en relation avec l'Institut Informatique de gestion de la HES-SO Valais. Il contribue à un apport d'informations pour un projet de l'Institut. Le projet consiste à un développement d'une chaîne logistique de démonstration, en utilisant les différentes technologies disponibles pour l'implémentation de solutions orientées processus. Ce scénario sert de base de démonstration. L'objectif de l'Institut est de démontrer aux entreprises, les possibilités des solutions orientées processus, et de fournir des prestations permettant la migration de l'entreprise vers une solution orientée processus.

Le sujet de ce travail de diplôme comporte deux performances. L'analyse de la technologie et son implémentation dans le scénario de la chaîne logistique. L'objectif de l'analyse consiste à présenter les caractéristiques et fonctionnalités de la technologie. L'objectif de l'implémentation est de démontrer la manière de la faisabilité, ainsi qu'une explication de la façon dont les fonctionnalités ont été implémentées à l'aide de ces outils.

L'objectif final est d'avoir une base de connaissances afin de déterminer les possibilités et les performances de cette technologie, pour voir dans quel cadre il est possible de l'utiliser et pour pouvoir évaluer ses forces et ses faiblesses par rapport aux autres technologies existantes.

VI. ETAT DE L'ART

L'importance d'avoir un système de gestion efficient et performant afin de conduire une organisation n'est plus à démontrer. Les entreprises perçoivent de plus en plus les avantages qu'elles peuvent retirer de l'optimisation de leurs processus métier. Ces dernières années le Business Process Management connait un essor de plus en plus important. Mais qu'entend-on exactement par le terme de Business Process Management. La définition exacte de ce terme n'est pas strictement définie et l'on trouve beaucoup de transcriptions différentes. L'explication ci-dessous est celle qui définit de façon la plus simple et la plus complète cette notion.

''Le BPM est avant tout une discipline et est défini comme suit : Ensemble des règles, des obligations qui régissent certains corps ou collectivités ; règlement. La discipline militaire. Manquer, se plier à la discipline (définition issue du Larousse).

Un processus métier est un enchaînement ordonné d'activités, qui se déroulent en série ou en parallèle, qui sont exécutées par des personnes ou par des applications (activités manuelles, semi-automatiques ou automatiques) et aboutissant à un résultat attendu. Un processus se caractérise par un événement déclencheur en entrée, des entrées suivies d'activités permettant de construire un résultat et le résultat final (sortie) attendu par le bénéficiaire.

Le Business Process Management (gestion des processus métiers) est une discipline qui comprend 4 axes principaux :

- *La modélisation des processus : représentation graphique des processus*
- *L'automatisation des processus : pour ce qui est automatisable et intégrable*
- *La gestion des processus : états des processus et gestion de leur cycle de vie*
- *L'optimisation : amélioration des processus basée sur des métriques réelles de performance des processus*

Figure 2 : BPM Life Cycle

Source : ([BPMS02] WebSite)

Cette discipline permet de gérer et d'améliorer les processus de l'entreprise durant leur cycle de vie, et s'accompagne de la mise en place d'outils permettant de modéliser, d'intégrer, de superviser et d'analyser le fonctionnement des processus. Le BPM apporte une méthode permettant de comprendre comment optimiser ces processus. Elle nécessite des profils pluridisciplinaires tels que l'expertise métier, le

développement d'applications et en passant par de l'architecture, et le conseil en technologies.'' ([GesProMe] WebSite)

Afin de pouvoir mettre un tel système en place, les entreprises doivent faire appel à un système permettant ce déploiement. De nombreux systèmes de gestion des processus métiers existent sur le marché. Mais quelles sont les possibilités que doit fournir un système pour être caractérisé comme BPMS (Business Process Management System) :

Le système doit comporter un ensemble de logiciels destinés à formaliser les procédures qui font l'activité d'une entreprise dans le but de les automatiser. Cet ensemble comprend généralement: ([BPMS01] WebSite)

- *''Un outil de modélisation qui servira à formaliser la description des fonctions exercées dans l'entreprise en processus, en applications informatiques. Il permettra de définir également les données échangées, les interfaces avec les autres modules.*
- *Des outils de développement pour formaliser la logique qui régit les processus de l'entreprise, à énoncer les règles de fonctionnement.*
- *Un moteur d'exécution qui supervisera le déroulement des processus ainsi que les échanges de paramètres.*
- *Un moteur de règles qui évaluera l'état de tous les objets impliqués dans le déroulement des processus et déterminera si les conditions sont remplies pour en lancer, poursuivre ou arrêter l'exécution.*
- *Un référentiel qui mémorisera tous les objets manipulés, en particulier les définitions des processus, les règles qui doivent déclencher leur exécution, les contraintes d'intégrité, de sécurité ainsi que les mesures de référence relatives au métier de l'entreprise.*
- *Des outils d'administration qui permettront de régler les paramètres de l'ensemble du système et d'obtenir des indicateurs de performance et des statistiques à partir des données collectées lors de l'exécution des processus.''*

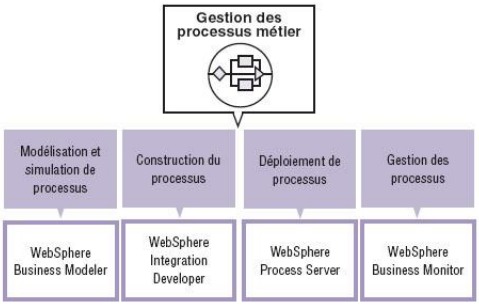
L'avantage de cette séparation des outils dans un tel système permet à tous les intervenants de l'implémentation ; analyste, développeur et gestionnaire, de travailler sur des outils différents que chacun connaît, en fonction de son métier, mais dont le résultat obtenu pourra être mis en commun et intégré dans le système. Cette architecture permet également de redonner le pouvoir décisionnel aux analystes pour modéliser les processus avec le logiciel de modélisation des processus, sans devoir faire appel aux développeurs.

Actuellement il existe un grand nombre de systèmes se disant BPMS, mais ceux-ci respectent-ils les points mentionnés ci-dessous ? Le paragraphe suivant présente quatre des plus utilisés. IBM WebSphere[1], BEA Aqualogic[2], Intalio[3], Expert Ivy[4].

IBM WebSphere

IBM propose une solution fondée sur une architecture orientée services. Cette architecture regroupe une famille complète de produits qui couvre l'ensemble des fonctions de gestion des processus métier.

Figure 3 : Solution IBM WebSphere



Source : SiteWeb IBM

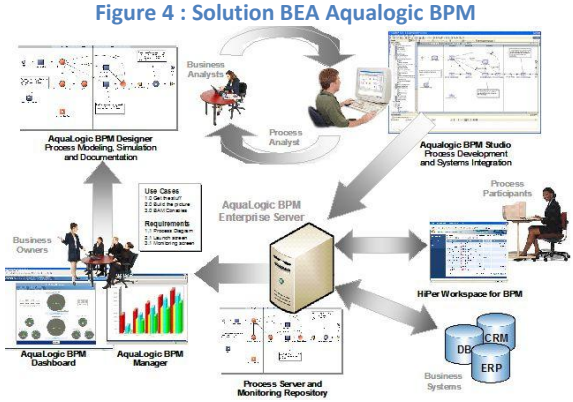
''Grâce à WebSphere Business Modeler, vous pouvez littéralement dessiner les flux de travaux entrant dans ces processus, identifier et concevoir les étapes nécessaires pour mener à bien de simples tâches. Vous pouvez ensuite « traduire » ces modèles d'entreprise dans le langage BPEL (Business Process Execution Language) et les exporter vers WebSphere Integration Developer, où services nouveaux et existants peuvent être combinés pour créer de nouvelles applications composites. Il vous suffit ensuite d'utiliser ce code pour déployer votre application en la diffusant via WebSphere Process Server. WebSphere Business Monitor vous permet enfin de surveiller vos processus et leurs performances pour mieux les ajuster.'' ([IBMWES] WebSite).

[1] Source : http://www-306.ibm.com/software/fr/websphere/
[2] Source : http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic/albpm
[3] Source : http://www.intalio.com/
[4] Source : http://www.consilium-it.ch/xpert-ivy-fr36.html

BEA Aqualogic

La solution BEA Aqualogic est également une solution orientée service. Elle permet la création, l'exécution et l'optimisation de processus.

Figure 4 : Solution BEA Aqualogic BPM



Source : Image Google

''BEA AquaLogic® BPM Designer is the business analyst's design environment. It enables the creation of any type of process by simply dragging and dropping process elements onto swim-lanes.

BEA AquaLogic BPM Studio is the process developer's workbench. It is a superset of AquaLogic BPM Designer, offering additional tools that empower the developer to write business logic, connect to existing applications and assemble user interfaces for human interaction.

BEA AquaLogic® BPM Enterprise Server orchestrates all resources part of each process — people, organizations, applications and systems — coordinating the proper sequence, enforcing business rules, and auditing each step to ensure flawless process execution, escalation and exception management.

BEA AquaLogic® HiPer Workspace for BPM is part of the BEA AquaLogic BPM Enterprise Server and provides a high performance workspace for business process participants interacting with process activities and workflows.

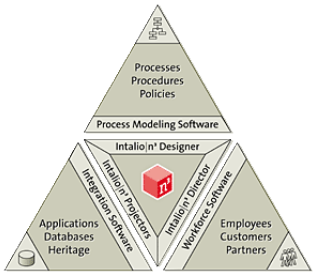
BEA AquaLogic® BPM Manager is an integrated Web console that provides real-time information about all processes managed by the BEA AquaLogic BPM Enterprise Server.

BEA AquaLogic® BPM Dashboard provides IT administrators and business users with historical activity data for business processes running in BEA AquaLogic BPM Enterprise Server.'' ([BEABPM] WebSite)

Intalio

"*Intalio|BPMS 4.0 is architected around three main ideas: support the latest industry standards for BPM, offer a zero code development approach and provide a one-click deployment life cycle.*" ([INTBPM] WebSite)

Figure 5 : Solution Intalio



Source : Image Google

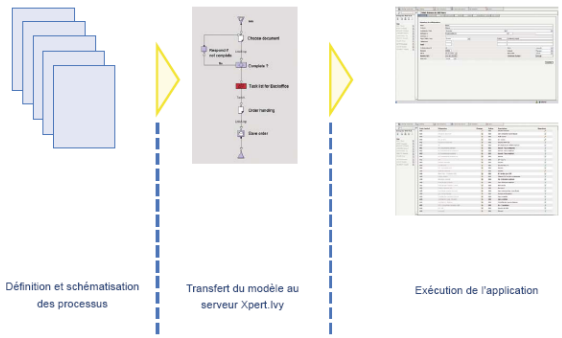
''*Intalio|Designer is an Eclipse-based integrated development environment for BPMN business processes. Business analysts and software engineers can both use Intalio|BPMS, because it's the easiest way we found to bridge the gap between business and IT. Also, since it offers zero code development and one-click deployment, you do not have to be a J2EE guru or an XML expert to use it, and if you are one anyway, you will find in Intalio|Designer a formidable productivity enhancer.*

Intalio|Server is a native BPEL 2.0 process server based on J2EE. Because it is architected around the new Java Business Integration (JBI) technology, it can be deployed on virtually any J2EE application server. Our process server was designed from the ground-up to be deployed on a grid of 1,000 servers by the US Department of Energy (DOE), so if your project requires high-performance process execution, you might have just found what you were looking for.

Intalio|Workflow is an integrated human workflow suite based on the new BPEL4People extensions and compatible with any JSR 168 portal. Because it offers an AJAX-based XForms implementation, it gives workflow participants a productive and engaging user experience, while remaining compatible with any major web browser in use today. Our workflow suite is directly powered by our process server, allowing you to develop your own workflow patterns.'' ([INTBPM] WebSite)

XpertIvy

Figure 6 : Solution XpertIvy



Source : ([XPEIVY] WebSite)

''*Xpert.Ivy se positionne comme un outil permettant la mise en œuvre applicative des processus sans programmation. Comme dans le cadre des certifications ISO habituelles, les procédures sont simplement et graphiquement décrites dans Xpert.Ivy Designer. Des assistants permettent ensuite de décrire de manière intuitive les liens avec les applications tierces et les écrans utilisateur. Durant la phase d'élaboration, les modèles peuvent être simulés et testés à tout moment. Quelques clics suffisent à appliquer le modèle dans le serveur Xpert.Ivy. La transposition des nouveautés dans l'application est alors réalisée de manière totalement automatique en fonction d'une période de validité choisie. Si le nombre de processus parallèles n'est pas limité dans l'exécution de l'application, il en va de même pour ce qui touche à sa maintenance. Plusieurs personnes peuvent simultanément mettre à jour le modèle sans que cela ne pose le moindre problème d'intégrité.*'' ([XPEIVY] WebSite)

L'aperçu de ces différents BPMS montre bien que la tendance est la séparation de la solution en différents logiciels pouvant être utilisés par des intervenants de compétences différentes. Chaque intervenant, l'analyste, le développeur et le gestionnaire dispose de son propre programme et le système permet une mise en commun de l'implémentation faite sur les différents outils.

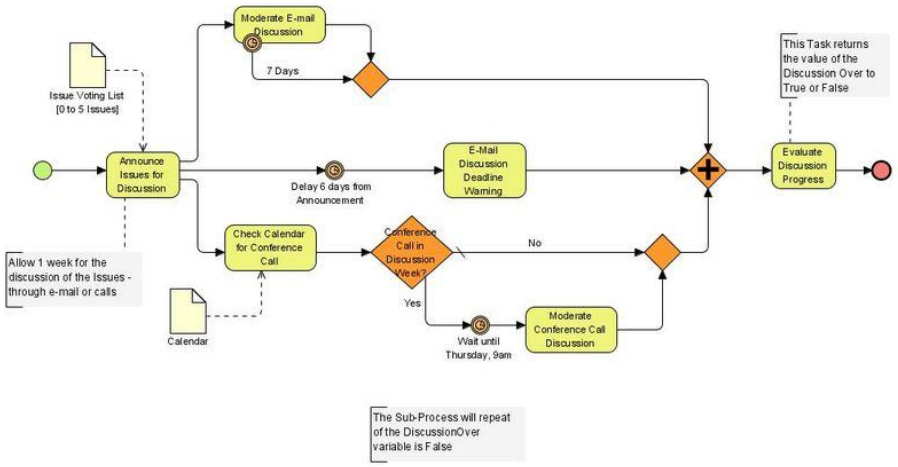
Actuellement, afin de modéliser les processus, on aspire à unifier cette modélisation. Cette unification est connue sous le nom de BPMN (Business Process Model Notation). Son objectif est de fournir un cadre permettant de décrire un processus d'une manière commune à tous les utilisateurs et ce, indépendamment de l'outil utilisé. L'outil étant bien sûr censé supporter la norme. Le deuxième objectif est de permettre aux différents intervenants de l'implémentation de travailler avec la même notification. L'analyste métier crée et redéfini les processus, le développeur est responsable de l'implémentation des processus et le gestionnaire entretien et surveille les processus. Cette norme permet également de combler le fossé qui sépare la modélisation des processus et l'implémentation de ceux-ci, ce qui n'est pas négligeable.

Microsoft arrive sur ce marché en évolution en proposant sa nouvelle technologie fournie avec le Framework 3.0, appelée Windows Workflow Foundation. A l'instar des systèmes actuellement sur le marché, cette brique du Framework 3.0 ne fournit pas un système contenant un ensemble de logiciels permettant la mise en place d'une solution orientée BPM, mais propose tous les outils nécessaires au développement d'une telle solution. De plus, l'outil de modélisation des processus de Windows Workflow Foundation ne suit pas la norme BPMN[5] précédemment citée. Il est intéressant de voir que Microsoft ne suit pas la même direction que la tendance du marché. La question que l'on peut dès lors se poser est ; est-ce que la technologie Workflow Foundation est un outil BPMS ? Mon avis sur la question sera exprimé en

[5] Pour de plus amples informations sur les définitions composant la norme BPMN veuillez vous référer au site suivant : http://en.wikipedia.org/wiki/BPMN

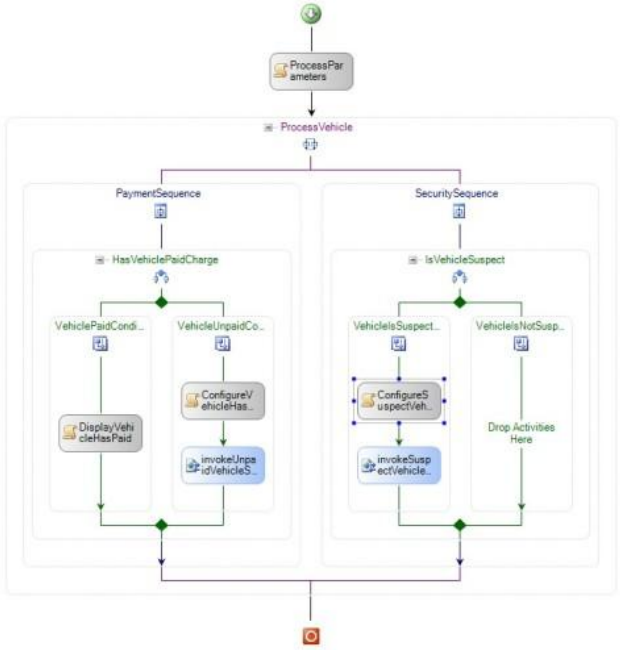
conclusion de ce document en me basant sur l'analyse de la technologie et l'implémentation réalisée.

Figure 7 : Modélisation BPMN



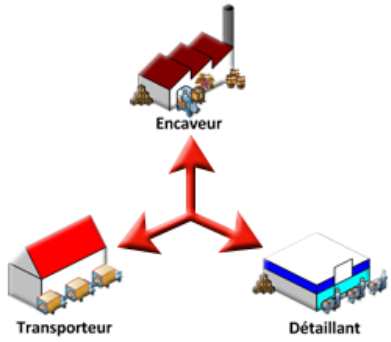
Source : Wikipédia (BPMN)

Figure 8 : Modélisation WF



Source : Google images

VII. SCENARIO DE LA CHAÎNE LOGISTIQUE



Le scénario de la chaîne logistique est composé de trois acteurs : un encaveur, un transporteur et un détaillant. L'encaveur doit livrer le vin au détaillant, faisant appel au transporteur. Le scénario a été simplifié par rapport à la réalité afin de limiter la complexité des interfaces et des intercommunications entre celles-ci et permettant ainsi de terminer l'application dans les temps impartis. Cette simplification est délimitée dans les deux documents suivants en appendice : TD BPMS – Définition du scénario et TD BMPS – Définition des interfaces techniques, page 98.

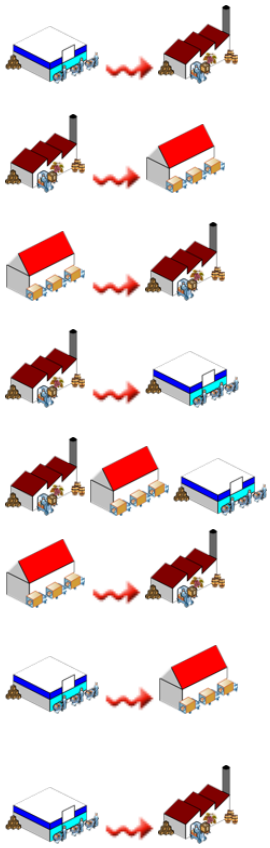
1 Les acteurs

Encaveur : l'encaveur est responsable de la cave, il vinifie le vin, le conditionne et le livre aux détaillants.

Transporteur : le transporteur assure le transport du vin de l'encaveur au détaillant. Il possède des clients dans tous les domaines de marchandises.

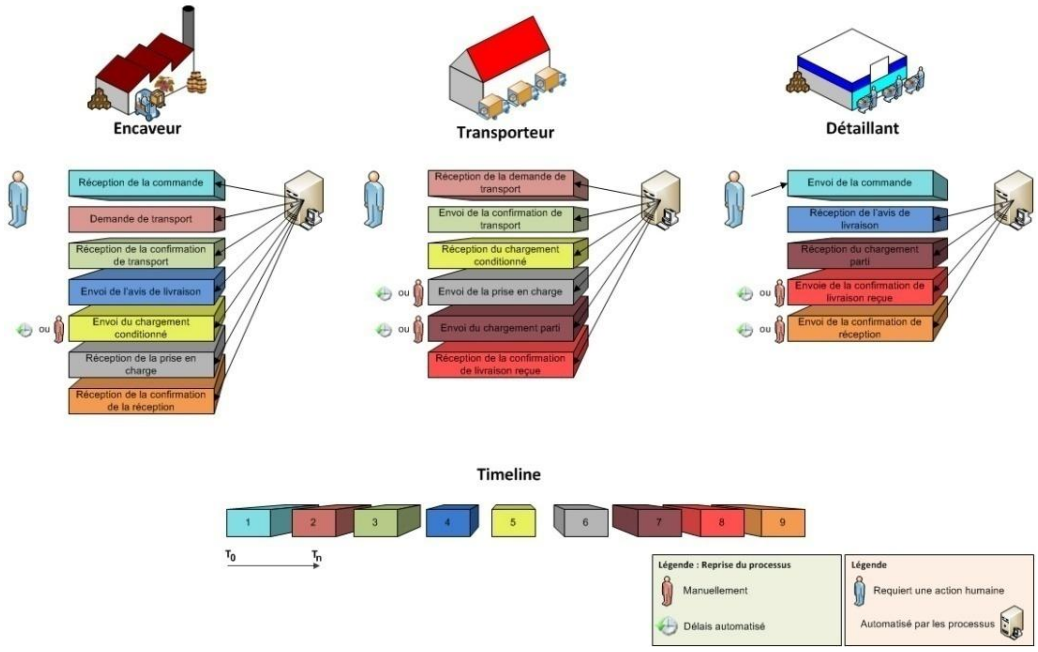
Détaillant : le détaillant exploite une surface de vente au détail. Il réapprovisionne son assortiment auprès de grossistes et de producteurs, parmi lesquels, pour le vin, les encaveurs.

2 Déroulement du scénario



L'élément déclencheur du scénario est le détaillant, en effet, lorsque celui-ci atteint son seuil critique pour les vins, il va passer une commande à l'encaveur.
L'encaveur réceptionne la commande et dans le but d'acheminer les produits au client, fait une demande au transporteur.
Le transporteur réceptionne cette demande et informe l'encaveur qu'il est prêt à exécuter la demande, il envoie une confirmation à l'encaveur.
L'encaveur envoie un avis de livraison au détaillant afin de l'informer qu'il va lui livrer les produits correspondants.
Le scénario continue avec le transporteur qui va charger la marchandise chez l'encaveur afin de la livrer chez le détaillant.
Après le chargement, le transporteur envoie à l'encaveur une confirmation de prise en charge de la marchandise.
Lorsque le transporteur a livré les marchandises chez le détaillant, ce dernier, lui envoie une confirmation de livraison pour confirmer la réception des marchandises livrées.
Finalement, le détaillant envoie un accusé de livraison à l'encaveur afin de lui confirmer la réception des marchandises commandées.

Figure 9 : Activités du scénario



Le schéma ci-dessus liste les activités qui sont accomplies par chaque acteur tout au long du scénario. La time line nous indique la succession ordonnée des activités à l'aide des blocs de couleurs numérotés. Chaque couleur correspond à une interaction entre deux maillons de la chaîne. La légende nous permet de distinguer si une activité donnée est traitée par une action humaine, c'est-à-dire une action manuelle, ou alors grâce à un processus automatisé.

L'objectif de ce scénario consiste à avoir des actions humaines en nombre le plus restreint afin d'avoir une automatisation optimale.

Au sein des entreprises, l'enchaînement des activités tout au long d'un scénario requiert parfois un certain temps. En effet, il peut se passer plusieurs heures voir plusieurs jours avant que l'activité suivante ne commence. L'enchaînement de l'activité « Envoi du chargement conditionné » après l'activité « envoi de l'avis de livraison » peut nécessiter une attente d'un jour. Il est possible de simuler cette attente de deux manières différentes, la première consiste à implémenter un délai entre les tâches à l'intérieur du processus métier, et la deuxième consiste à activer l'activité suivante par une action humaine. L'implémentation réalisée, permet ces deux simulations.

VIII. DEVELOPPEMENT

1 Exploration technologique

1.1 Introduction

Ce chapitre couvre la phase d'exploration de la technologie utilisée pour la réalisation du travail demandé, à savoir, l'implémentation d'une chaîne logistique existante avec la technologie « Microsoft Windows Workflow Foundation ».

Il n'a pas pour but de présenter l'implémentation de codes complets afin de réaliser et exécuter une application, mais présente la technologie dans son ensemble. Comment fonctionne cette technologie, quels sont ces concepts, quels sont les outils et les objets fournis, quelles sont les possibilités de celle-ci, et le plus intéressant, pour quelle tâche et pourquoi employer tel ou tel objet ou fonctionnalité plutôt qu'un ou une autre. Ce chapitre répertorie également certains pièges à éviter et ce à quoi il faut faire attention en utilisant certains composants.

Il commence par présenter le matériel requis afin d'utiliser cette technologie. Puis continue avec une présentation générale expliquant les concepts de base, l'architecture, l'utilité. Ensuite, il présente l'environnement de développement, la place de travail permettant l'implémentation avec cette technologie. La dernière partie présente les objets et composants mis à disposition et leur utilisation.

La découverte, l'apprentissage et la compréhension de cette technologie a été faite avec l'aide des sources suivantes.

Centre de développement DotNet Framework ([FrmDotNet] WebSite)

Microsoft DotNet Framework 3.0 Community (Netfx3) ([NetFxC] WebSite)

Windows Workflow Foundation ([WWFMS] WebSite)

Pro WF Windows Workflow in DotNet 3.0 ([BukBru07] Bukovics, 2007)

Ce chapitre se base essentiellement sur la source « Pro WF Windows Workflow Foundation ». En effet le chapitre ci-dessous présente un condensé de cet ouvrage.

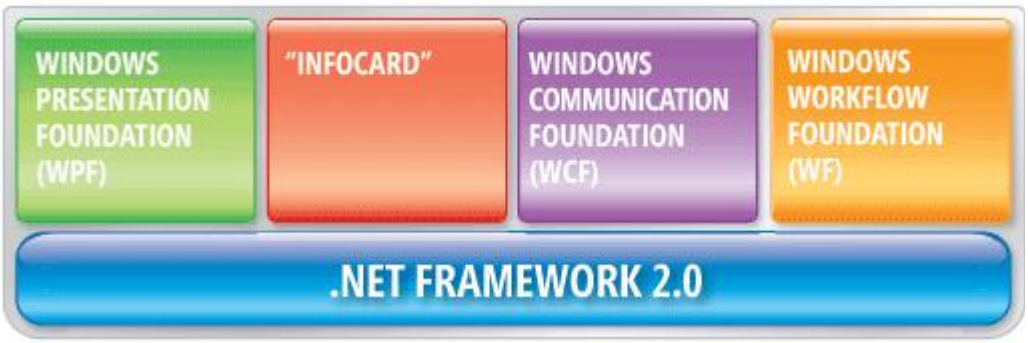
1.2 Pré-requis

Pour pouvoir développer des applications avec cette technologie, il faut une machine de développement supportant le « Framework 3.0 ». C'est-à-dire, une machine avec un système d'exploitation comprenant soit « Windows XP Service Pack 2 »,

« Windows 2003 Serveur Service Pack 1 » ou « Windows Vista ». Niveau logiciel, « Microsoft Visual Studio 2005 » version complète est requis afin de pouvoir utiliser le « Visual Workflow Designer », la version « Express » ne supportant pas le « Visual Workflow Designer ». Il faut l'extension pour « Visual Studio 2005 » qui permet d'utiliser les composants du « Framework 3.0 ». Il faut également une deuxième extension pour « Visual Studio 2005 » pour permettre l'utilisation du « Visual Workflow Designer », de Template et du « Workflow Debugger ». Le matériel requis peut être téléchargé sur le site MSDN de Microsoft.

1.3 Framework 3.0

Figure 10: Framework 3.0



Source : ([FrmDotNet] WebSite)

''Microsoft .NET Framework 3.0 (précédemment appelé WinFX) est le nouveau modèle de programmation de code géré pour Windows. Il combine la puissance de .NET Framework 2.0 à quatre nouvelles technologies : Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF), Windows Workflow Foundation (WF) et Windows CardSpace (WCS, anciennement « InfoCard »). Utilisez .NET Framework 3.0 pour créer des applications visuellement attractives, offrant une communication transparente au-delà des technologies utilisées, prenant en charge un large éventail de processus métier et fournissant un moyen plus simple de gérer vos informations personnelles en ligne. La technologie de pointe WinFX que vous connaissez et appréciez a changé de nom. Sa nouvelle désignation indique qu'il s'agit précisément de la version suivante de l'infrastructure de développement de Microsoft. Ce changement n'affecte pas la planification des versions de .NET Framework 3.0 ni les technologies incluses dans le package.'' ([FrmDotNet] WebSite)

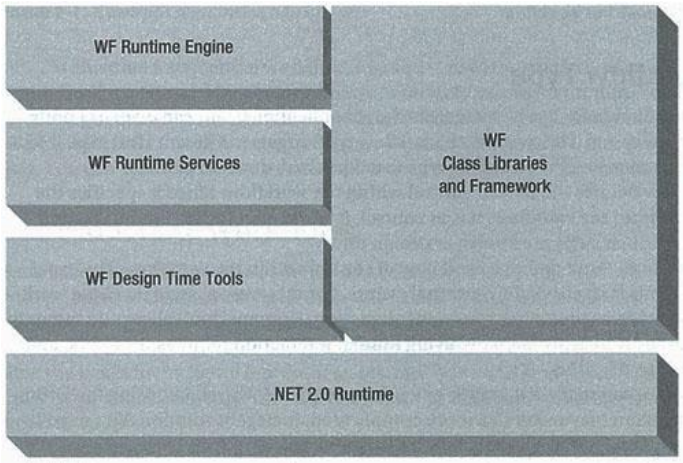
1.4 Windows foundation

Windows Workflow fournit des composants majeurs pour l'exécution des processus. Chaque composant intervient pour une action particulière durant le cycle de vie d'un processus. Certains sont employés seulement durant la phase de « design » alors que d'autres n'entrent en fonction que lors de la phase d'exécution. Ci-dessous, voici la liste des composants faisant partie de la brique « WF » du « Framework 3.0 ».

1.4.1 WF Class Libraries and Framework

Ce composant fournit les classes de bases et les interfaces utilisées lors de la création d'application orientée processus. Les processus que nous construisons sont composés de ces classes de bases, ainsi que de nos propres classes héritant de celles-ci. Toutes les classes de cette librairie font partie du namespace *System.Workflow*.

Figure 11: WF Component categories



Source : ([BukBru07] Bukovics, 2007)

Runtime Engine

Ce composant contient le cœur du moteur d'exécution des processus. Il est représenté par la classe *WorkflowRuntime*. Ce « runtime » n'est pas une application mais un ensemble de classes qui doivent être implémentées dans notre application hôte afin de pouvoir manager les processus. Notre application héberge ce « runtime » et celui-ci héberge les instances de nos processus.

Runtime Services

Le moteur d'exécution est construit avec un concept de services externes. Ces services sont enregistrés auprès du « runtime » pendant le démarrage de l'application. Il y a deux types distincts de services, les « core » services et les « local » services. Les

« core » services sont fournis par Microsoft. Ce sont des services que l'on peut activer selon notre besoin et qui sont également paramétrables. Ces différents « core » services sont présentés un peu plus loin dans le document. Les « local » services au contraire, sont des services développés par nos soins et permettent de coller à nos propres besoins. Par exemple, les « local » services peuvent servir de pont pour l'échange d'informations entre l'application hôte et le processus.

Design Time Tools

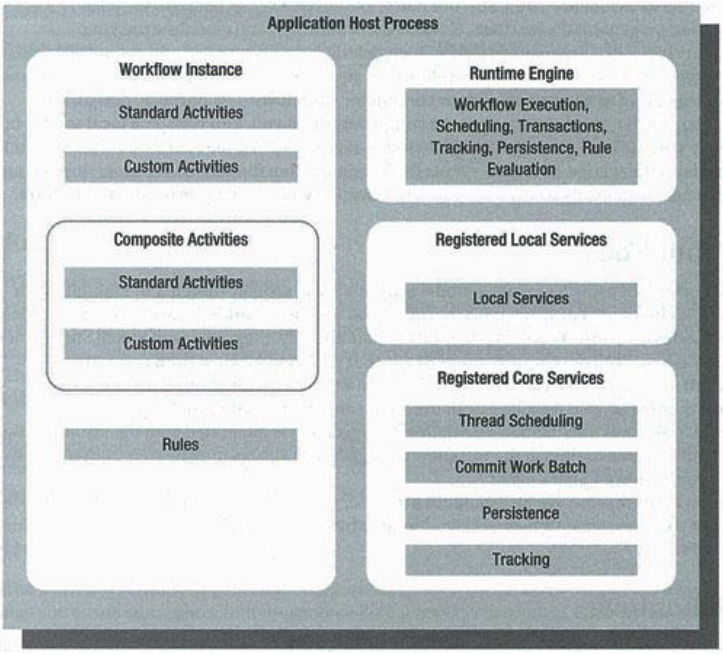
Ces outils permettent de modéliser les processus, les activités, etc., sans devoir quitter l'environnement de développement de « Visual Studio ». Ils supportent le développement à l'aide du « drag & drop » qui est une des fonctionnalités présente dans « Visual Studio ». Ce composant comprend également le débuggeur de processus permettant l'ajout des points d'arrêt et l'analyse de notre code.

DotNet 2.0 Runtime

La brique « WF » est entièrement construite sur le « .Net 2.0 CLR » qui est packagée avec « DotNet 3.0 ». Cela signifie que toues les possibilités du « .Net 2.0 CLR » sont conservées.

1.4.2 Runtime Environment

Figure 12: Workflow runtime environment



Source : ([BukBru07] Bukovics, 2007)

Application Host Process

WF n'est pas fourni en tant qu'application autonome, mais est réellement une base sur laquelle nous pouvons construire notre application. C'est notre application qui va créer une instance du « runtime engine ». La façon dont Microsoft a développé ce composant est très intéressante, car de ce fait il nous permet de construire toutes sortes d'applications orientées processus. Des applications consoles, windows service, windows forms, des applications WPF et également des applications web ASP DotNet.

Cette configuration a une limitation, elle ne permet que la création d'une seule instance par AppDomain, donc pour la majorité des applications cela signifie que l'on ne peut avoir qu'une seule instance du « runtime engine » par application hôte.

Runtime Engine

Le « runtime engine » fournit l'environnement d'exécution des processus, c'est-à-dire que l'on n'exécute pas directement un processus à l'intérieur de notre application, mais que l'on demande au « runtime engine » de créer une instance de processus que l'on va démarrer ensuite.

Par défaut, les processus sont exécutés de manière asynchrone dans un thread managé par le « runtime engine ». Il manage également le cycle de vie des processus, ainsi que ces différents états durant son exécution.

Il s'occupe également de la gestion des services qu'on lui aura assignés avant son démarrage. Ceci entend les « core » et le « local » services.

Il contient également un moteur d'évaluation de règles. Ce moteur est capable d'évaluer des conditions simples comme (If Else ou While).

Les évènements publics que génère le « runtime engine » peuvent être manipulés par l'application hôte, ce qui permet de surveiller l'état de chaque instance de processus. Il n'est pas obligatoire de s'abonner à tous les évènements mais certains sont obligatoires, par exemple, « WorkflowCompleted » et « WorkflowTerminated ». Etant donné que les processus sont exécutés asynchrone, ces évènements permettent de savoir à quel moment un processus n'est plus exécuté.

Registered Core Services

Pour manager les processus, le « runtime engine » peut faire appel à quatre « core » services, services que l'on doit référencer avant le démarrage du « runtime engine ». Ces services n'ont pas été implémentés directement au sein du « runtime engine » afin de permettre une plus grande flexibilité à notre application. Par exemple, il est inutile de s'enregistrer au service de persistance si l'on n'en a pas l'utilité. Le deuxième avantage est qu'il nous est possible de développer nos propres services afin de subvenir le mieux possible à nos besoins. Ces quatre services sont :

Scheduler Service

Ce service manage les threads utilisés par le « runtime engine » pour l'exécution des processus.

Persistence Service

Permet la gestion de la persistance des processus sous la direction du « runtime engine ».

Tracking Service

Permet de surveiller le fonctionnement des processus en enregistrant les évènements de ceux-ci. La particularité de ce service contrairement aux trois autres, c'est que l'on peut en enregistrer plusieurs par « runtime engine », alors que les autres requièrent un enregistrement unique par « runtime engine ».

Commit Work Batch Service

Manage les transactions utilisées par le « runtime engine » pour maintenir l'uniformité entre l'état interne des processus et celui enregistré dans le magasin de données externes (base de données, fichier xml, etc.).

Le scheduler service et le commit work batch service sont les deux seuls services obligatoires pour le « runtime engine ». Si on ne les enregistre pas personnellement avec les paramètres souhaités, ils sont enregistrés automatiquement au démarrage de l'application avec les paramètres par défaut.

Registered Local Service

Les « local » services sont optionnels et servent de conduit de communication entre l'application hôte et une instance de processus. Ils mettent à disposition des méthodes qui peuvent être directement utilisées par une instance de processus. Ils mettent également à disposition des méthodes permettant à l'instance de processus d'attendre sur un évènement généré par un « local » service. Ils font vraiment office de pont entre l'application hôte et une instance de processus. Comme pour les « core » services, les « local » services doivent être enregistrés auprès du « runtime engine » avant le démarrage de ce dernier.

Les évènements générés par un local service ne fonctionnent pas tout à fait de la même manière que des évènements standards. Quand un évènement est levé, il est placé dans une queue d'attente interne à l'instance de processus. L'exécution du processus peut continuer quand il récupère l'évènement dans la queue d'attente. De ce fait, les évènements des « local » services sont exécutés sur le thread du processus et non sur le thread qui l'a créé.

Workflow Instance

Une instance de processus est, comme son nom l'indique, une unique instance exécutable d'un processus. Elle est représentée en code par la classe *WorkflowInstance* qui travaille comme un proxy avec l'instance de processus réelle. Cette classe contient des méthodes permettant d'interagir et de contrôler l'instance d'un processus. Pour pouvoir exécuter une instance de processus, il faut tout d'abord que le « runtime engine » soit initialisé. Ensuite, il faut créer l'instance du processus en utilisant une méthode du « runtime engine » qui nous retourne une *WorkflowInstance.* A ce moment, le processus est créé mais n'est pas encore démarré. Il faut manuellement démarrer le processus à l'aide de la méthode *Start* de l'objet *WorkflowInstance*.

Afin de pouvoir identifier les instances de processus qui auront été créées, l'objet *WorkflowInstance* contient un identifiant unique, le *Guid*.

Workflow Activities

Un processus est composé d'une ou plusieurs activités. On distingue deux catégories d'activitiés. Les « standard » et les « custom ». Les « standard » sont celles fournies par Microsoft dans WF. A l'inverse, les « custom » sont celles développées par nous-mêmes. Ce sont des activités permettant d'effectuer des tâches non standard. Ce sont ces activités qui représentent la puissance de WF permettant une expansion sans limite des possibilités.

Les activités sont classées parmi deux groupes, les « simple » et les « composite » activités. Les « simple » activités sont celles qui ne permettent pas de contenir d'autres activités. Celles-ci effectuent une tâche. A l'inverse, les « composites » activités n'effectuent pas de tâches mais contrôle et manage ces activités « enfants ».

Workflow Rules

Les processus peuvent optionnellement définir des règles qui seront évaluées durant l'exécution de l'instance du processus. C'est le « rules evaluation engine » du « runtime engine » qui se charge de cette évaluation.

Les *PolicyActivity* encapsulent une série de règles et les évaluent en séquence. Après évaluation d'une règle, le moteur détermine si une règle antérieure doit être réévaluée. Cela serait le cas par exemple si une règle modifie une variable dont une règle antérieure est dépendante. Ce processus s'appelle le « forward chaining ».

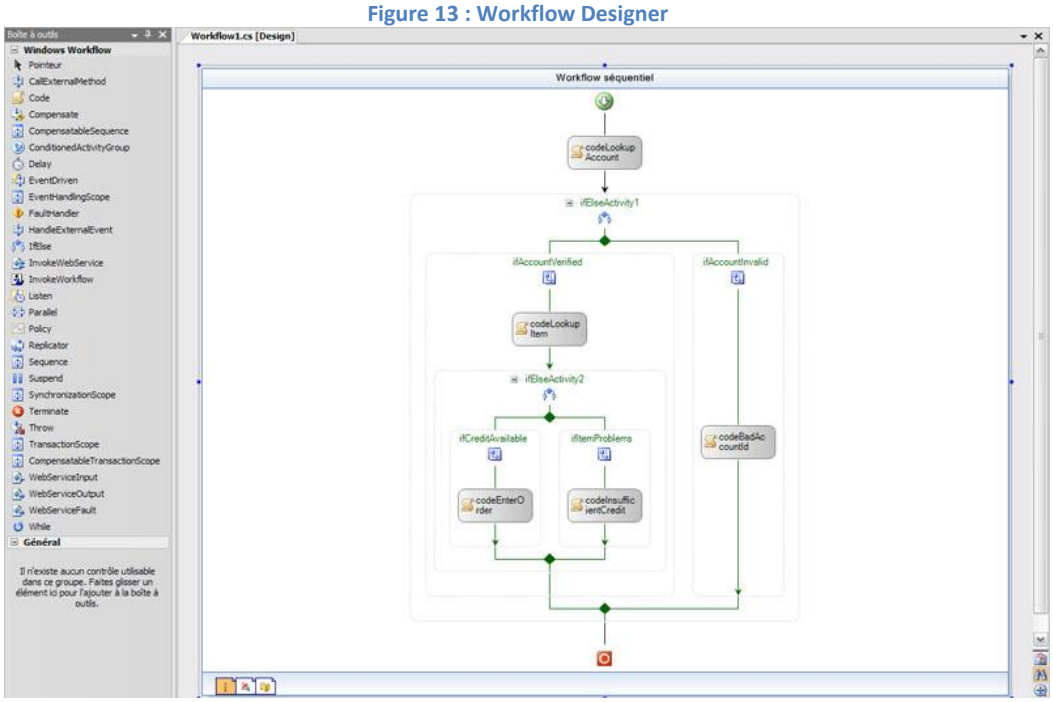
L'avantage des règles est que WF crée un fichier *.rules* avec le même nom que le fichier du processus. Etant donné que ces règles sont évaluées séparément du code compilé, ce fichier peut être modifié au démarrage sans devoir recompiler le code.

1.5 Design Environment

La partie modélisation de processus à l'aide de l'environnement de conception est une des possibilités les plus intéressantes de WF. Il est tout à fait possible de modéliser un processus sans utiliser cet environnement de développement, mais on perd toute la facilité et la puissance d'utilisation du designer de Visual Studio.

Workflow Designer

Figure 13 : Workflow Designer



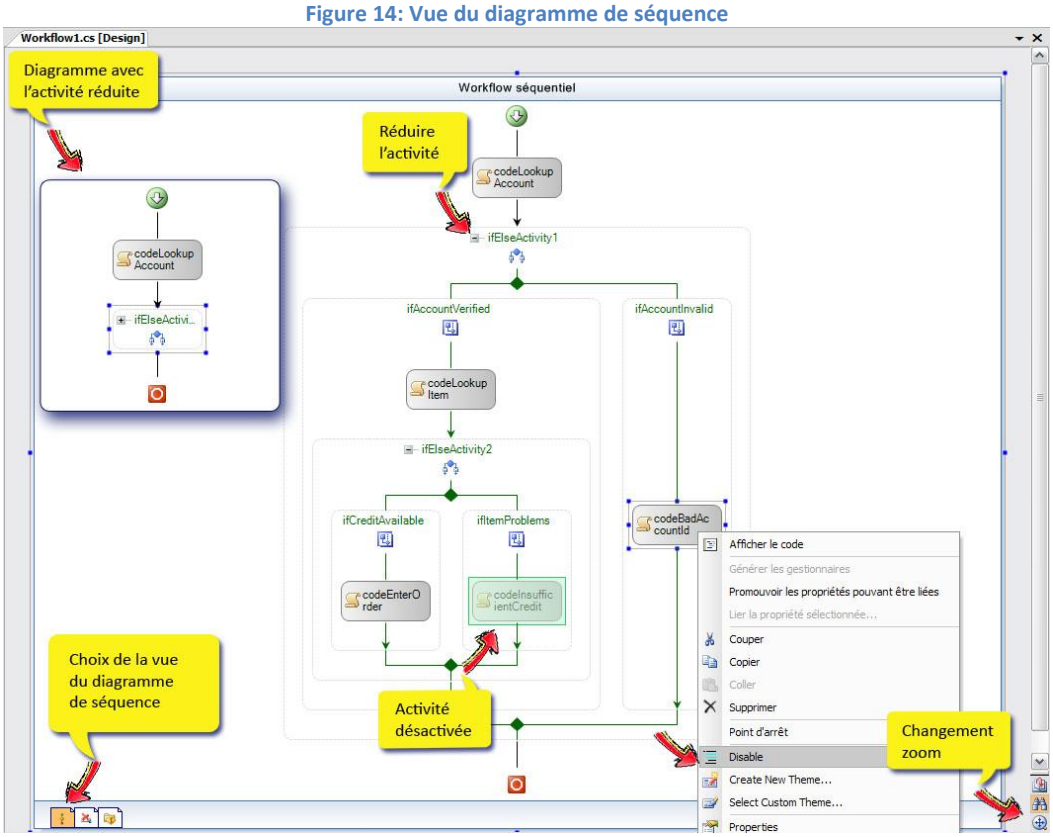
Le « Workflow Designer » est l'outil principal de l'environnement de conception. Grâce à lui, on peut conceptualiser et maintenir le processus visuellement à la place d'écrire des lignes de codes. Le designer complètement intégré dans Visual Studio permet le « Drag & Drop » de la « ToolBox » standard de Visual Studio.

L'utilisation de ce designer est des plus commune, au centre se trouve le schéma du processus, et pour y ajouter des activités, il suffit simplement de « Drag & Dropp », les objets apparaissant dans la « ToolBox ». Les « custom » activités apparaissent également dans la « ToolBox » après avoir généré la solution.

Il y a différentes vues possibles qui sont chacune tournées vers un type de processus ou un problème de conception. On peut passer d'une à l'autre de ces vues à l'aide des boutons en bas à gauche de la fenêtre principale. Il n'est pas discuté de la vue des processus « state machine » ceux-ci n'ayant pas été utilisés pour ce projet.

Sequential Workflow View

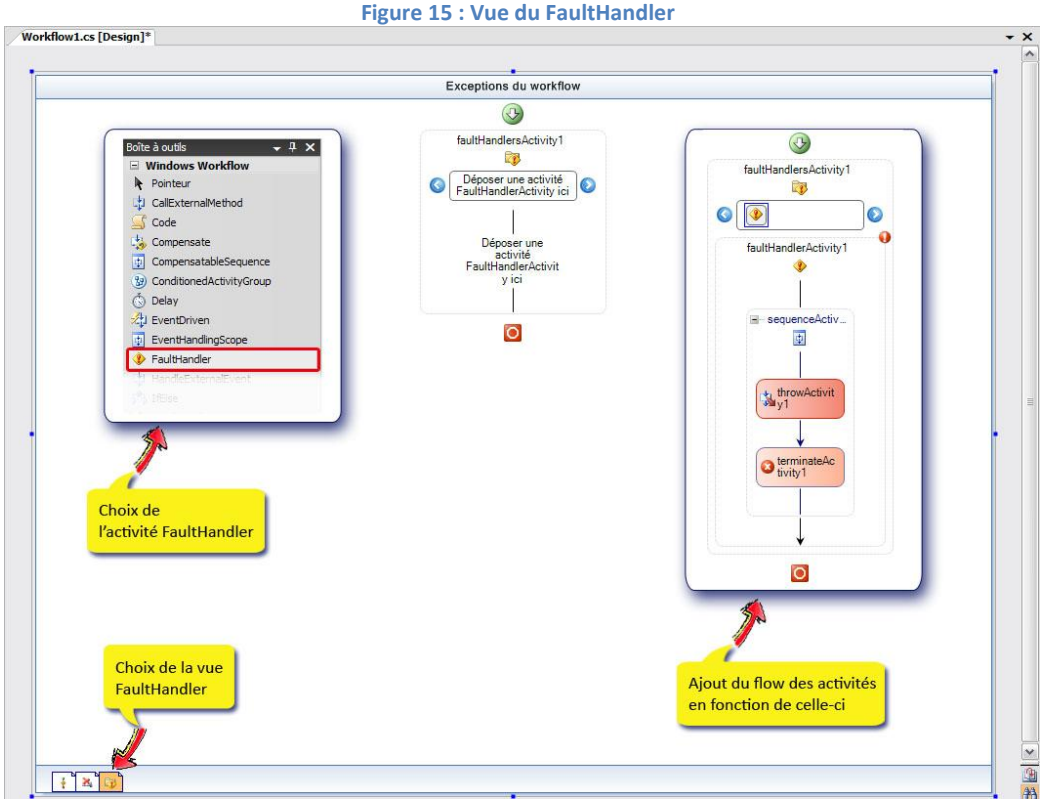
Figure 14: Vue du diagramme de séquence



Permet de voir visuellement notre processus. On peut choisir le zoom, activer ou désactiver des parties du processus, réduire ou augmenter les activités.

Fault Handler View

Figure 15 : Vue du FaultHandler

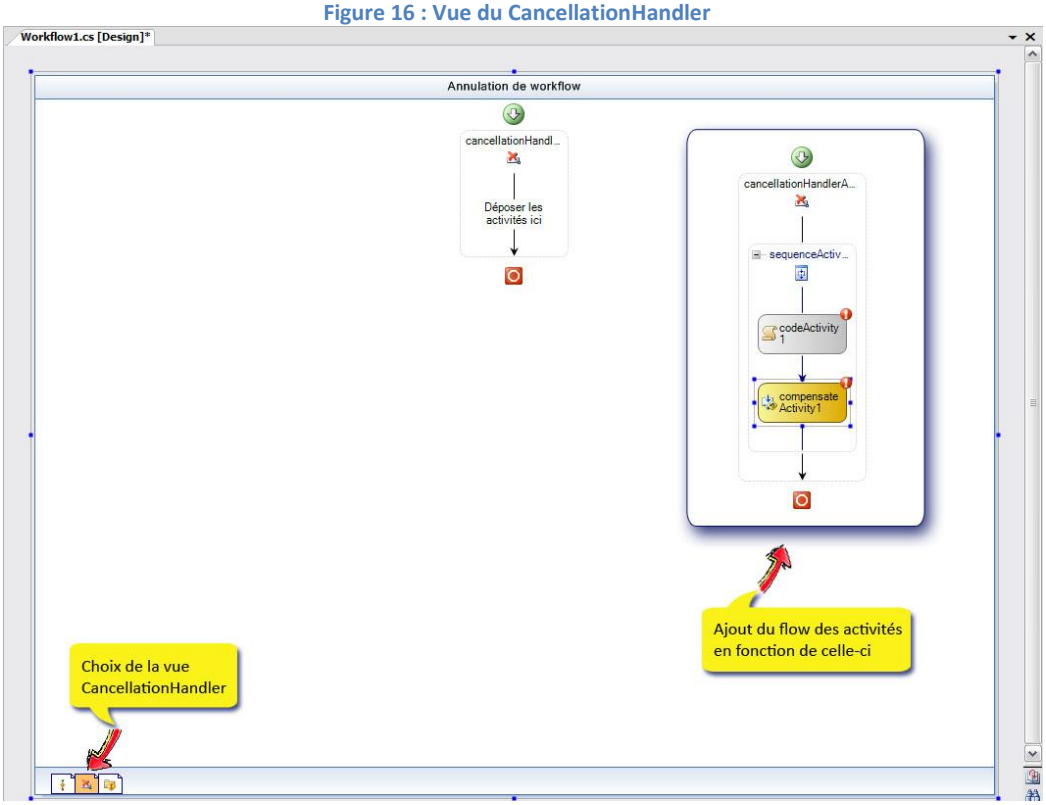


Des exceptions peuvent être levées à tout moment durant l'exécution d'un processus. Ces exceptions peuvent être capturées à l'intérieur des activités par un « try catch » ou alors visuellement à l'aide de cette vue. Cette vue procure un moyen visuel de définir la logique de capture des exceptions. Grâce à cette vue, on peut déterminer quelle activité capture les erreurs et quelles actions elle effectue pour une erreur donnée.

Cancel Handler View

Permet de définir une logique d'annulation dans le processus. Ces activités ne capturent pas les exceptions mais définissent les actions qui sont exécutées lorsqu'une activité est annulée.

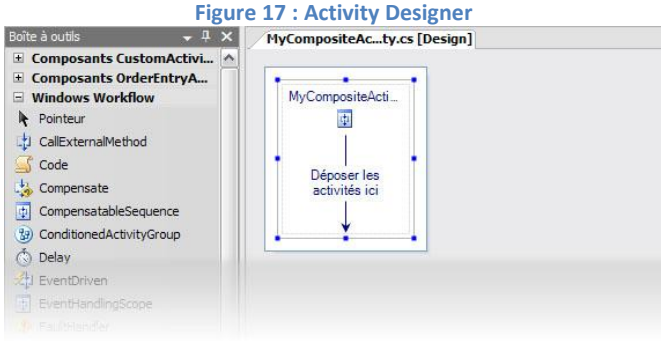
Figure 16 : Vue du CancellationHandler



Activity Designer

Permet de créer visuellement nos propres activités. Il permet le « Drag & Drop » des activités au sein de lui-même. Ensuite grâce au « code behind » nous pouvons écrire le code comportant la logique de cette activité.

Figure 17 : Activity Designer



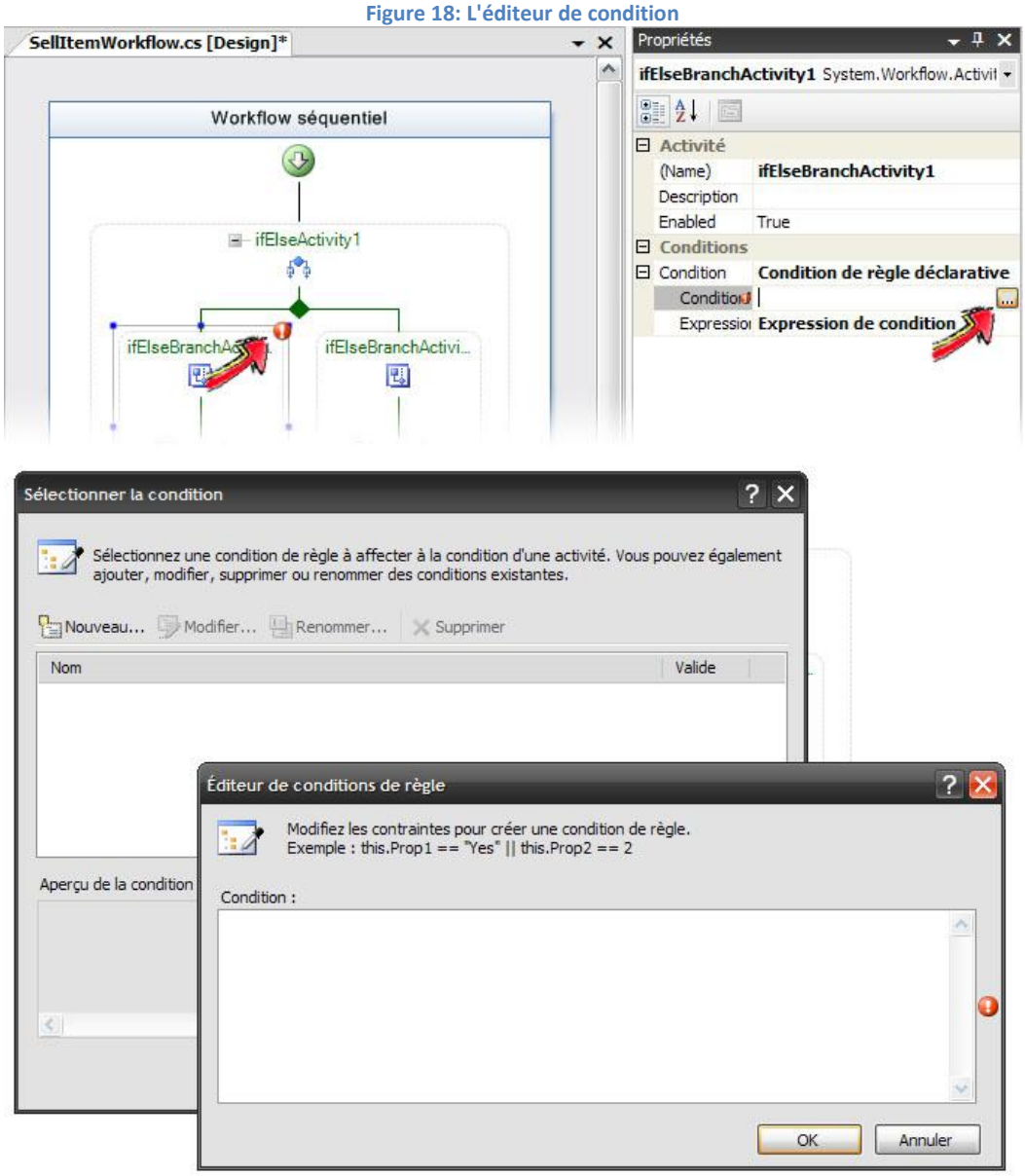
Rule Editor

Permet d'entrer et de maintenir les règles associées à un processus. Les règles peuvent être utilisées de deux manières différentes. Elles peuvent être simplement une condition booléenne pour une activité demandant une condition, par exemple une activité de type « If Else » ; ou alors, elles peuvent être combinées à l'intérieur d'un « Ruleset » et utilisées par une activité de type « PolicyActivity ». Ces règles sont alors évaluées une à la fois et peuvent être réévaluées suivant les actions des règles antérieures.

Rule Condition Editor

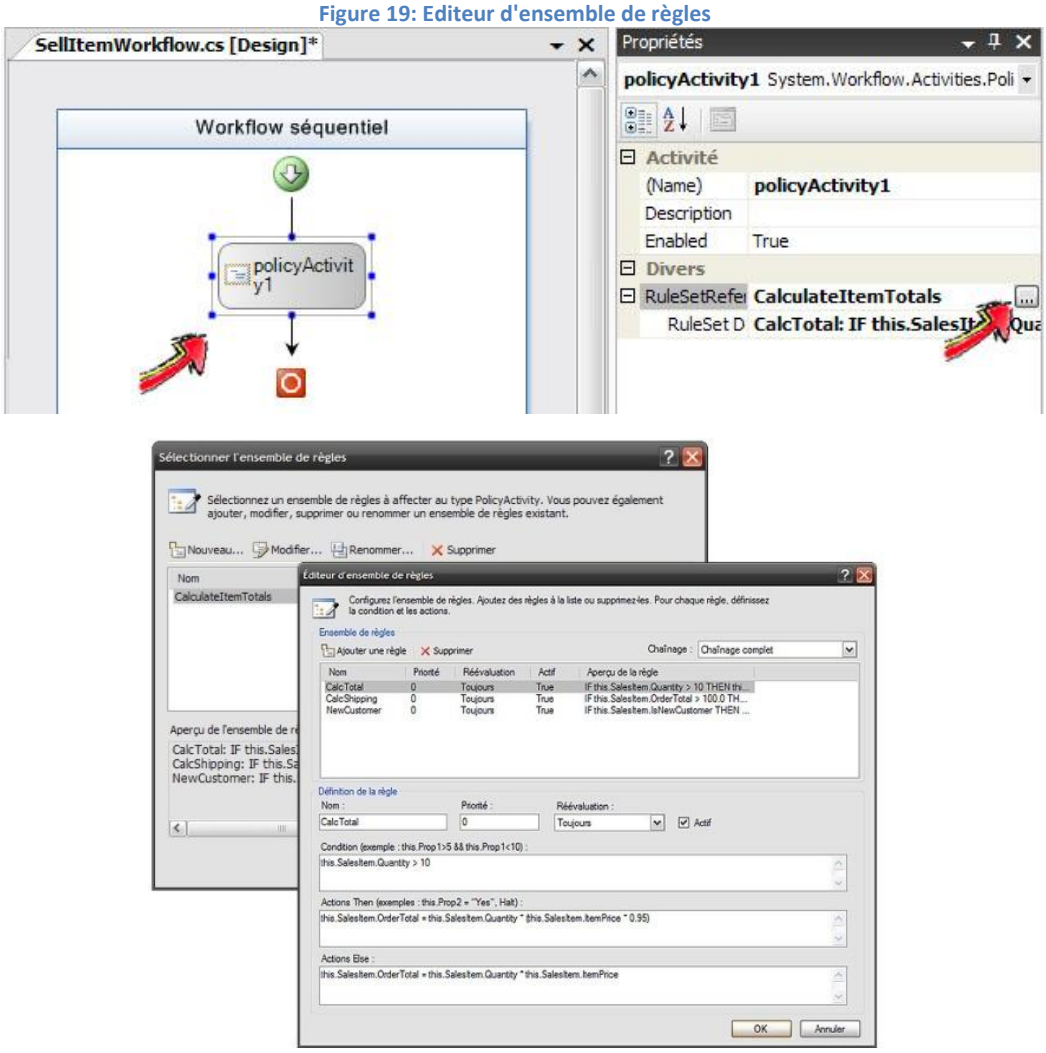
Pour voir apparaître la fenêtre « Rule Condition Editor » voici le pré-requis. Il faut que le processus contienne des champs ou des propriétés et il doit contenir une activité demandant une condition. Lorsque l'on sélectionne l'activité, dans les propriétés de cette activité, il y a un champ C*ondition*, il faut choisir « Declarative Rule Condition » puis dans le champ *Expression*, il faut cliquer sur l'« Ellipsis ».

Figure 18: L'éditeur de condition



Ruleset Editor

Figure 19: Editeur d'ensemble de règles



Permet d'entrer et de maintenir plusieurs règles qui travaillent ensemble, comme un groupe. Pour le voir apparaître, il faut avoir une activité de type « PolicyActivity » dans le processus. Dans les propriétés, dans le champ *RulesetRreference,* il faut cliquer sur l' « Ellipsis ».

Il est possible de déterminer l'ordre d'exécution des règles à l'aide de l'attribut *Priorité*, la règle avec la valeur de cet attribut la plus haute est exécutée en premier et ainsi de suite. Les règles peuvent être actives ou inactives. Il est également possible de choisir le chaînage des règles, c'est-à-dire si les règles sont exécutées une seule fois les une après les autres ou alors avec le « forward-chaining ». Pour chaque règle, il est également possible de déterminer si la réévaluation doit se faire ou pas.

1.6 Type de processus

WF permet la modélisation de deux types de processus, les « sequential workflow » ou les « state machine workflow ».

1.6.1 Sequential Workflow

Les processus de ce type permettent l'exécution d'activités dans un ordre bien précis. Le processus démarre avec un point d'entrée, effectue les opérations en fonction de la logique déterminée par le processus et finalement se termine lorsqu'il a atteint le point de sortie. Dans ce type de processus il est impossible d'atteindre une activité se trouvant au milieu du processus sans passer par le point d'entrée.

1.6.2 State Machine Workflow

A l'opposé d'un « sequential workflow », le « state machine workflow » ne définit pas une séquence d'activité dans un ordre précis. A la place, il définit un groupe d'état avec plusieurs transitions possibles. Chaque état comporte un certain nombre de tâches. Le « state machine workflow » n'est pas contraint à une séquence de tâches statiques. Les transitions entre les états sont exécutées par des évènements externes qui sont levés par l'application hôte. Le contrôle du processus se fait uniquement de l'extérieur.

1.6.3 Le choix du type de processus

Les deux types de processus peuvent être employés dans des applications similaires, mais chaque type permet de régler un problème particulier.

Le type « sequential workflow » définit l'ordre exact ainsi que le nombre exact des étapes d'un processus. De ce fait, il est utilisé lorsque l'on connait exactement le nombre d'étapes, pendant la phase de design, que le processus doit couvrir. Etant donné que le « state machine workflow » ne suit pas en continu les étapes il est contrôlé par des évènements extérieurs. De ce fait, ce type de processus est employé lorsque le processus nécessite une interaction humaine, les personnes n'effectuant pas toujours les tâches dans le même ordre.

Dans la plupart des cas, il est possible d'employer les deux types de processus mais le mauvais choix nécessitera une solution plus compliquée et inélégante.

1.7 Activities

Les activités sont des composants permettant d'effectuer une tâche spécifique durant l'exécution d'un processus. WF comprend un certain nombre de « standard » activités différentes qui permettent de gérer la plupart des logiques de processus. WF permet

également de créer nos propres activités. Travailler avec des « custom » activités permet de résoudre les problèmes métiers spécifiques à nos besoins.

Quelque soit le type d'activité choisie l'utilisation reste identique, à savoir, « Drag & Drop » de la « Toolbox » sur la vue du processus, paramétrage de l'activité à l'aide de l'explorateur des propriétés de l'objet. Les propriétés servent à définir le comportement de l'activité lors de l'exécution ou alors permet également de définir les *inputs* et *outputs* de celle-ci.

Certaines activités sont capables de contenir d'autres activités, on dit que ce sont des « composite » activités alors que d'autres ne peuvent qu'exécuter une certaine tâche, on les appelle alors les « simple » activités.

Une fonctionnalité intéressante des activités est la possibilité de les customiser afin qu'elles apparaissent et réagissent différemment lorsqu'on les glisse sur la vue du processus dans le designer. Il est possible de changer le design des activités, couleur, forme, icône, etc. ce qui peut apporter une meilleure vue d'ensemble du processus. Afin d'améliorer la productivité lors de la phase de design il est possible, à l'aide d'attribut et de composants optionnels d'afficher les erreurs directement sur le schéma du processus, afin de les découvrir directement et non de les voir apparaître seulement lorsque la solution est compilée. Cette fonctionnalité est étudiée plus en détail dans le chapitre Customiser la conception des processus.

1.7.1 Standard Activities

Le but de ce chapitre n'est pas d'expliquer comment utiliser et comment fonctionnent les différentes activités fournies par WF, mais de présenter les différentes catégories d'activités et leur utilisation, afin de pouvoir faire le bon choix pour la tâche qu'il faut réaliser. Voici la liste de ces catégories :

- Custom Workflow Logic
- Flow Control
- State Management
- Event Handling
- Local Service Communication
- Rules
- Web Services
- Transactions, Compensation and Synchronisation
- Exceptions and Error Handling

Custom Workflow Logic

Une seule activité fait partie de cette catégorie, elle permet d'ajouter le code concernant la logique business à un processus.

Code Activity

Flow Control

Les activités de cette catégorie permettent de gérer le cours du processus, l'écoulement des tâches depuis le point de départ jusqu'au point d'arrivée. Cette catégorie comprend également les activités permettant l'exécution d'activités en parallèle ou la réplication d'une activité. Les activités permettant de gérer les délais, de suspendre ou terminer le processus sont également dans cette catégorie car elles agissent sur le cours du processus.

IfElseActivity ,IfElseBranchActivity, WhileActivity, DelayActivity, SequenceActivity, ParallelActivity, ReplicatorActivity,ConditionedActivityGroup, InvokeWorkflowActivity, SuspendActivity, TerminateActivity

State Management

Cette catégorie d'activités n'est pas détaillée dans ce document car ce type de processus n'a pas été étudié.

EventHandling

Les activités de cette catégorie s'occupent toutes de la gestion des évènements et travaillent avec des activités « enfants » qui reçoivent l'évènement. Elles sont les conteneurs pour d'autres activités gérant les évènements.

ListenActivity, EventDrivenActivity, EventHandlersActivity, EventHandlingScopeActivity

Local Service Communication

Les activités de cette catégorie sont utilisées pour la communication entre un « local » service et une instance de processus.

CallExternalMethodActivity, HandleExternalEventActivity,

Rules

L'unique activité de cette catégorie est associée avec le « workflow rules engine ».

PolicyActivity

Web Services

Les activités de cette catégorie permettent l'interaction avec les services web. Avec ces activités il est possible d'appeler un web service ou à l'inverse d'exposer le processus à des clients de service web.

InvokeWebServiceActivity, WebServiceInputActivity, WebServiceOutputActivity, WebServiceFaultActivity

Transactions, Compensation and Synchronization

Les activités de cette catégorie permettent de définir une unique unité de travail qui englobe plusieurs activités. Elle comprend également des activités qui permettent la « compensation » et la synchronisation d'accès aux variables.

CompensatableTransactionScopeActivity, CompensatableSequenceActivity, CompensateActivity, CompensationhandlerActivity, SynchronizationScopeActivity

Exceptions and Error Handling

Les activités de cette catégorie travaillent toutes avec les exceptions DotNet (classe *Exception*). Une (*ThrowActivity*) permet de lancer une exception en dehors de l'instance de processus alors que les autres sont utilisées pour attraper les exceptions qui ont été lancées.

ThrowActivity, FaultHandlerActivity, FaulthandlersActivity, CancellationHandlerActivity

1.7.2 Customiser la conception des processus

La customisation de la conception des processus permet de personnaliser l'apparence et la réaction des activités qui se trouvent sur la vue d'un processus de séquence. Il est possible d'agir de manière différente sur le design. Il est possible d'agir sur :

La validation des activités : la validation permet d'attraper les erreurs de configuration durant la phase de design et de ne pas attendre l'exécution de l'activité d'un processus en cours d'exécution.

Customisation du comportement de la barre d'outils : il est possible de personnaliser l'action d'une activité lorsqu'elle est déplacée de la barre d'outils de Visual Studio sur la vue de conception du processus.

Implémentation d'un design et d'un thème personnalisé : il est possible de créer son propre designer en modifiant le thème, les couleurs, les polices et d'autres attributs visuels.

Cette customisation ne peut être faite que sur des « custom » activités car nous allons ajouter des attributs référençant d'autres classes qui seront également développées. Il est vrai que la conception de ces « custom » activités prend plus de temps, mais étant donné qu'elles peuvent être réutilisées dans d'autres processus et qu'elles améliorent la conception des processus c'est une fonctionnalité non négligeable.

Dans les prochains paragraphes il est expliqué plus en détail comment ajouter la validation des activités sur la vue de conception de processus. Il est apporté une attention plus particulière à cette fonctionnalité car elle permet d'améliorer la productivité et l'efficacité du développeur. D'aucune manière cette fonctionnalité n'améliore les possibilités de l'activité durant son exécution.

Validation des activités

Lors de la conception d'une « custom » activité il est possible d'y ajouter la validation de celle-ci. La principale validation qui peut être implémentée est la vérification des champs manquants ou non valides. Il est par exemple également possible de restreindre la possibilité à certain type d'activité d'être des activités enfants. Les erreurs obtenues sont directement affichées dans la vue de conception des processus.

Pour qu'une activité puisse être validée, le développement consiste en deux étapes, premièrement la création d'une classe de validation et finalement l'assignation de cette classe à notre « custom » activité.

L'exemple ci-dessous montre comment assigner une validation sur une propriété manquante.

La première étape pour pouvoir implémenter cette validation consiste à vérifier que notre « custom » activité possède une ou plusieurs propriétés.

```
using System;
using System.ComponentModel;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Compiler;
using System.Workflow.Activities;

namespace CustomActivityComponents
{
    /// <summary>
    /// A custom activity that demonstrates activity components
    /// </summary>
    [ActivityValidator(typeof(MyCustomActivityValidator))]
    public partial class MyCustomActivity : Activity
    {
        public MyCustomActivity()
        {
            InitializeComponent();
        }

        public static DependencyProperty MyStringProperty
            = System.Workflow.ComponentModel.DependencyProperty.Register(
            "MyString", typeof(string), typeof(MyCustomActivity));

        [Description("A String property")]
        [Category("Custom Activity Components")]
        [Browsable(true)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
        public string MyString...

        public static DependencyProperty MyIntProperty
            = System.Workflow.ComponentModel.DependencyProperty.Register(
            "MyInt", typeof(Int32), typeof(MyCustomActivity));

        [Description("An Int32 property")]
        [Category("Custom Activity Components")]
        [Browsable(true)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
        public Int32 MyInt...
    }
}
```

La seconde étape consiste à créer la classe permettant de valider les propriétés de l'activité.

```
using System;
using System.Workflow.ComponentModel.Compiler;

namespace CustomActivityComponents
{
    /// <summary>
    /// Validator for MyCustomActivity
    /// </summary>
    public class MyCustomActivityValidator : ActivityValidator
    {
        public override ValidationErrorCollection Validate(
            ValidationManager manager, object obj)
        {
            ValidationErrorCollection errors = base.Validate(manager, obj);
            //only validate a single custom activity type
            if (obj is MyCustomActivity)
            {
                MyCustomActivity activity = obj as MyCustomActivity;
                //only do validation when the activity is in a workflow
                if (activity.Parent != null)
                {
                    if (activity.MyInt == 0)
                    {
                        errors.Add(
                            ValidationError.GetNotSetValidationError(
                                "MyInt"));
                    }

                    if (activity.MyString == null ||
                        activity.MyString.Length == 0)
                    {
                        errors.Add(new ValidationError(
                            "MyString Property is incorrect", 501));
                    }
                }
            }
            return errors;
        }
    }
}
```

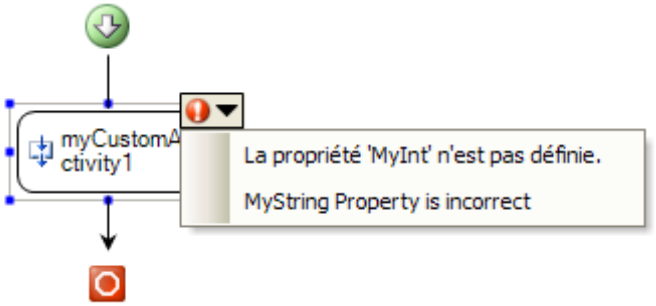
Notre classe de validation doit hériter de la classe *ActivityValidator*. L'implémentation de la validation se passe dans la méthode *Validate*. Cette méthode retourne un objet de type *ValidationErrorCollection* ce qui nous permet de retourner plusieurs erreurs. Dans cette implémentation, la méthode contrôle que l'objet passé est bien du type de notre « custom » activité, ce qui restreint la validation à cette classe. Ensuite l'objet est « casté » du type de notre activité. La méthode contrôle que le parent de l'activité n'est pas nul. Ce test est nécessaire pour que l'activité ne génère pas une erreur lors de sa compilation. Nous voulons que cette validation soit faite que si elle est insérée dans un processus. Ensuite, c'est le tour de la validation de nos propriétés. Il est possible de la gérer de différentes manières. Soit à l'aide de méthode « helper » statique de la classe *ValidationError, ValidationError.getNotSetValidationError.*Soit par la création d'une instance de *ValidationError* sans passer par une méthode « helper ». Ces deux validations sont passées à notre objet de retour.

La troisième et dernière étape consiste à assigner cette classe de validation à notre « custom » activité. Cette assignation est faite par l'ajout d'un attribut à notre « custom » activité afin d'identifier la classe qui va fournir la logique de validation.

```
[ActivityValidator(typeof(MyCustomActivityValidator))]
public partial class MyCustomActivity : Activity
```

Pour tester cette validation il suffit simplement de glisser la « custom » activité sur la vue de conception de processus et l'icône d'erreur rouge apparaît dans le coin de l'activité avec le message de description de l'erreur.

Figure 20: Validation de la CustomActivity



1.8 Enregistrement des services

Le « runtime engine » permet d'enregistrer des services. Il est possible d'enregistrer des « core » services et des « local » services. WF fournit deux possibilités d'enregistrement, enregistrement par le code ou enregistrement à l'aide d'un fichier de configuration. Le choix de l'enregistrement modifie quelque peu l'initialisation du runtime suivant la méthode choisie.

Voici par exemple comment enregistrer les services de persistance fourni par WF à l'aide d'un fichier de configuration.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="WorkflowRuntime"
      type="System.Workflow.Runtime.Configuration.WorkflowRuntimeSection,
        System.Workflow.Runtime, Version=3.0.00000.0, Culture=neutral,
        PublicKeyToken=31bf3856ad364e35" />
  </configSections>
  <WorkflowRuntime Name="ConsoleHostingManaged" >
    <CommonParameters>
      <!--Add parameters common to all services-->
      <add name="ConnectionString"
           value="Initial Catalog=WorkflowPersistence;
             Data Source=localhost\SQLEXPRESS;
             Integrated Security=SSPI;" />
    </CommonParameters>
    <Services>
      <!--Add core services here-->
      <add type="System.Workflow.Runtime.Hosting.SqlWorkflowPersistenceService,
        System.Workflow.Runtime, Version=3.0.00000.0,
        Culture=neutral, PublicKeyToken=31bf3856ad364e35"
        UnloadOnIdle="true" LoadIntervalSeconds="5" />
    </Services>
  </WorkflowRuntime>
</configuration>
```

L'initialisation du « runtime engine » se fait alors de la manière suivante.

```
WorkflowRuntime instance = new WorkflowRuntime("WorkflowRuntime");
```

Voici maintenant l'enregistrement du même service mais cette fois-ci avec une implémentation de code ainsi que l'initialisation du « runtime engine ».

```
String connStringPersistence = String.Format("Initial Catalog={0};Data
Source={1};Integrated Security={2};", "WorkflowPersistence",
@"localhost\SQLEXPRESS", "SSPI");

SqlWorkflowPersistenceService  persistanceService = new
SqlWorkflowPersistenceService(connStringPersistence, true,
new TimeSpan(0, 2, 0), new TimeSpan(0, 0, 5)));

WorkflowRuntime instance = new WorkflowRuntime();

instance.AddService(persistanceService);
```

Maintenant l'enregistrement d'un « local » service auprès du « runtime engine » avec un fichier de configuration ainsi que l'initialisation du « runtime engine ».

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="WorkflowRuntime"
      type="System.Workflow.Runtime.Configuration.WorkflowRuntimeSection,
        System.Workflow.Runtime, Version=3.0.00000.0, Culture=neutral,
        PublicKeyToken=31bf3856ad364e35" />
    <section name="LocalServices"
      type="System.Workflow.Activities.ExternalDataExchangeServiceSection,
        System.Workflow.Activities, Version=3.0.0.0, Culture=neutral,
        PublicKeyToken=31bf3856ad364e35"/>
  </configSections>
  <WorkflowRuntime Name="ConsoleLocalServices" >
    <CommonParameters>
      <!--Add parameters common to all services-->
    </CommonParameters>
    <Services>
      <!--Add core services here-->
    </Services>
  </WorkflowRuntime>

  <LocalServices >
    <Services>
      <!--Add local services here-->
      <add type="SharedWorkflows.AccountService,
        SharedWorkflows,Version=1.0.0.0,
        Culture=neutral, PublicKeyToken=null" />
    </Services>
  </LocalServices >

</configuration>
```

```
//add the external data exchange service to the runtime
ExternalDataExchangeService exchangeService = new
ExternalDataExchangeService("LocalServices");

WorkflowRuntime instance = new WorkflowRuntime("WorkflowRuntime");
instance.AddService(exchangeService);
```

Finalement, voici l'enregistrement du « local »service auprès du « runtime engine » de manière codée ainsi que l'initialisation du « runtime engine ».

```
//add the external data exchange service to the runtime
ExternalDataExchangeService exchangeService = new ExternalDataExchangeService();

WorkflowRuntime instance = new WorkflowRuntime();
instance.AddService(exchangeService);

//add our custom local service
//to the external data exchange service
exchangeService.AddService(new AccountService());
```

1.9 Gestion de la persistance

La persistance est un concept important lorsque l'on parle de processus. En effet, l'utilisation d'applications orientées processus requiert souvent une interaction humaine pour pouvoir continuer les enchaînements. Avant qu'une interaction survienne il peut se passer un temps considérable, quelques jours voire des mois. De ce fait, il est important que l'application puisse sauver le processus en cours et le recharger plus tard.

Sans la gestion de la persistance la durée de vie d'un processus est limitée au temps d'exécution de l'application. Si l'application rencontre une erreur et doit se fermer, notre processus est perdu. Ce qui n'est pas permissible dans un environnement de production.

WF propose en tant que « core » service la gestion de la persistance. Pour ce faire, il utilise le système de gestion de base de données relationnel SQL Serveur. Ce service est optionnel pour l'exécution du « runtime engine » est peut être enregistré avant l'exécution de celui-ci. Le fait que WF ait implémenté ce service de cette façon permet modularité et flexibilité quant au choix de persister les processus suivant les besoins requis. Il est tout à fait possible, par exemple, de choisir un autre SGBDR comme oracle ou alors dans des fichiers XML. Par contre, une telle manœuvre nécessite l'implémentation d'un « local » service.

Le « core » service de WF pour la persistance ne sauvegarde pas un processus mais sauvegarde l'état d'un processus. Il est important de faire cette distinction car on ne peut pas employer ce service pour, par exemple gérer l'historique des processus. Ce « core » service ne sauvegarde que l'état des processus en cours, dès que le processus se termine il est supprimé de la base de données.

La persistance d'un processus est faite dans les situations suivantes :

- Lorsqu'un processus attend un évènement externe ou lors d'une activité nécessitant un délai.
- Lorsqu'un processus est terminé normalement « completed » ou arrêté « terminated ».
- Lorsqu'une activité de type « TransactionScopeActivity » se termine.
- Lorsqu'une activité de type « CompensatableSequenceActivity » se termine.
- Lorsqu'une activité comportant l'attribut « PersistOnCloseAttribute » se termine.
- Lors de l'invocation manuelle d'une méthode causant la persistance, par exemple, « unload ».

Lorsque vous employez à la fois le « core » service de persistance et celui de tracking, il faut suivre une ligne de conduite. Il faut employer la même base de données pour ces deux services. Si les deux services utilisent des bases de données différentes, il y a deux connectionString différentes et une transaction distribuée doit être obtenue par le DTC (Distributed Transaction Coordinator). Il en résulte une diminution de la performance.

Il faut également charger le « SharedConnectionWorkflowCommitBatchService » à la place de celui par défaut. Celui-ci utilise une connexion partagée afin d'éviter les transactions distribuées.

1.9.1 Service de persistance personnalisé

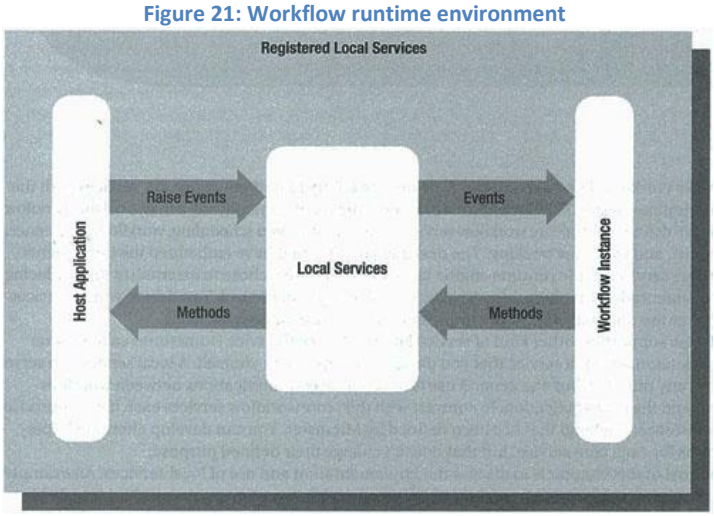
Il est possible de développer son propre service de persistance si le « core » service fourni ne satisfait pas vos besoins. Par exemple, votre application tourne avec une base de données Oracle ou MySQL, ou vous voulez utiliser SQL Serveur mais vous voulez employer des tables d'un autre schéma que vous avez déjà défini ; ou simplement que votre application requiert une gestion avec des fichiers et non avec une base de données.

Les étapes à suivre afin de développer son propre service sont les suivantes :

1. Dériver les classes personnalisées de gestion de la persistance de *WorkflowPersistenceService* du namespace *System.Workflow.Runtime.Hosting.*
2. Implémenter toutes les méthodes abstraites de cette classe de base.
3. Enregistrer le service personnalisé auprès du « runtime » de la même manière que le service standard.

1.10 Local Services

Figure 21: Workflow runtime environment



Source : ([BukBru07] Bukovics, 2007)

Précédemment, nous avons vu que WF est bien un composant et non une application toute définie, il doit être intégré à une application hôte. Le processus ainsi développé peut avoir besoin d'informations sur l'application hôte ou bien interagir avec des données persistées, c'est là que les « local » services ont toute leur utilité. En effet, l'objectif principal de ce genre de service est de permettre une interaction entre un processus et des évènements extérieurs. Il agit comme un pont entre ces deux mondes. En implémentant des fonctionnalités en tant que « local » service, celles-ci sont centralisées et disponibles pour plusieurs instances de processus. Avec WF il est possible d'enregistrer plusieurs « local » services auprès du « runtime engine » ce qui permet de grouper, de compartimenter les fonctionnalités.

Les processus interagissent de deux manières différentes avec des « local » services.

- Appeler une méthode d'un « local » service
- Attendre sur un évènement levé par un « local » service

Lorsqu'un processus interagit avec un « local » service, il interagit indirectement avec une partie de l'application hôte. Par exemple, lorsqu'un processus appelle une méthode d'un « local » service, cette méthode peut appeler une autre méthode de l'application hôte ou bien lever un évènement qui est traité par l'application hôte.

Inversement, un processus peut attendre sur un évènement levé par un « local » service, mais peut-être qu'un autre composant au sein de l'application hôte a levé cet évènement.

L'appel d'une méthode d'un « local » service se fait d'une manière synchrone sur le Thread du processus. De ce fait, il ne faut pas que la méthode exécute des opérations lourdes prenant un temps conséquent car elle va bloquer le Thread du processus pour une longue période.

Voici les différentes étapes d'implémentation d'un « local » service.

- Définir le contrat du service à l'aide d'une interface. Ce contrat définira les méthodes et les évènements qui seront atteignables par les instances de processus.
- Décorer l'interface avec l'attribut *ExternalDataExchangeAttribute* du namespace *System.Workflow.Activites*. Cet attribut identifie l'interface en tant qu'interface d'un « local » service.
- Implémenter une classe qui implémente l'interface précédemment définie.
- Créer l'instance de cette classe et l'enregistrer auprès du « runtime engine » pendant l'initialisation du *« runtime engine ».*

Un « local » service est identifié par le type d'interface, donc il ne peut être enregistré qu'une seule fois auprès du « runtime engine ».Par contre, il est possible d'enregistrer plusieurs « local » services tant qu'ils implémentent une interface différente.

1.10.1 Appeler une méthode d'un « local » service

Il est possible d'atteindre les méthodes d'un « local » service de deux manières différentes.

- Utiliser la méthode *GetService* pour obtenir une référence du « local » service et appeler ensuite une méthode définie par son interface.
- Utiliser l'activité fournie par WF *CallExternalMethodActivity*. Cette activité permet d'appeler un « local » service directement en tant qu'étape d'un processus.

1.10.2 Attendre un évènement d'un « local » service

La manière la plus commune de traiter un évènement d'un « local » service et l'utilisation de l'activité *HandleExternalEventActivity*.

Voici les différentes étapes nécessaires afin d'utiliser une activité de ce type :

- Implémenter une classe qui va être employée comme argument de l'évènement. Afin que celui-ci soit traité par l'activité *HandleExtrenalEventActivity*, il doit dériver de *ExternalDataEventArgs*.
- Définir l'évènement dans une interface de « local » service.
- Implémenter l'évènement dans une classe du « local » service.

- Fournir un moyen de lever l'évènement. Soit par une méthode que l'application hôte peut atteindre, soit levé l'évènement directement dans le « local » service.
- Ajouter une instance de l'activité *ListenActivity* au processus. Celle-ci va créer deux branches de type *EventDrivenActivity* en tant qu'activités enfants.
- Ajouter l'activité *HandleExternalEventActivity* à l'une de ces branches précédemment créées. Certaines propriétés doivent être définies pendant la phase de design. Après avoir choisi l'interface qui identifie le « local » service, il faut sélectionner l'évènement que l'on veut traiter.
- Compléter la deuxième branche de type *EventDrivenActivity* en ajoutant une activité implémentant *IEventActivity*. Couramment, on emploie une activité de type *DelayActivity* qui permet de continuer le processus si aucun évènement n'est traité après un certain temps.

L'étape suivante est de décider comment traiter l'argument de l'évènement qui est reçu. Il est possible de définir un paramètre de liaison afin de définir une propriété du processus comme valeur de l'argument reçu. Comme cela, il est possible d'utiliser cet argument durant tout le processus. L'autre possibilité est d'ajouter une activité de type *CodeHandler* à l'activité de type *HandledExternalEventActivity, c*omme cela, il est possible de travailler avec l'argument.

WF propose également une deuxième possibilité d'attendre un évènement d'un « local » service. A l'aide de l'utilitaire *wca.exe* il est possible de générer des « strongly typed custom activity ». Ces activités sont appelées des activités de communication. Chaque activité générée correspond à une méthode ou un évènement du « local » service.

Pour générer ces activités, il suffit de fournir à l'utilitaire le nom de l'*assembly* contenant les interfaces du ou des services. L'utilitaire va générer un fichier pour toutes les méthodes et un fichier pour tous les évènements définis dans les interfaces. Lorsque la génération est terminée, il suffit de les ajouter au projet, de générer le projet et celles-ci apparaissent et sont utilisables dans la boîte à outils de « Visual Studio » comme toutes autres activités.

1.11 Gestion du suivi des processus

Le suivi des processus est une fonctionnalité fournie par WF à l'aide d'un des ses « core » service. Ce service permet de suivre et d'enregistrer le statut et les données évènementielles pour chaque processus et chaque activité au sein des processus. Ce service de base enregistre et manipule ces données dans une base de données SQL Serveur. Comme pour le service de persistance, il est possible de ne pas activer ce service et à l'aide d'un « local » service de créer et implémenter un service de suivi personnalisé. A la différence des autres « core » service de WF, il est possible

d'enregistrer plusieurs services de suivi au « runtime engine » et ces services peuvent travailler avec les mêmes données de suivi.

Les données suivies peuvent être catégorisées en trois différents types d'évènements.

- Les évènements de processus.
- Les évènements d'activité.
- Les évènements d'utilisateur.

Les évènements de processus correspondent au changement de statut des processus, à chaque changement de statut d'un processus, un évènement est levé et est passé à tous les services de suivi enregistrés. Par exemple, lorsque le processus est créé, persisté, terminé ou encore lorsqu'une exception non capturée survient.

Les évènements d'activité correspondent au changement d'exécution des activités. Par exemple, lorsqu'une activité est exécutée, fermée ou encore annulée.

Les évènements de type utilisateur peuvent être générés tout au long d'un processus. L'implémentation de ceux-ci est totalement sous le contrôle du développeur. Pour créer un tel évènement, il suffit d'utiliser la méthode *TrackData* fournie par la classe de base *Activity*. Lorsque cette méthode est appelée, il est possible de passer n'importe quel type de paramètre suivi d'un champ de type *String* servant à identifier les données passées. La deuxième possibilité afin de générer un évènement de type utilisateur, consiste à ajouter un *UserTrackPoint* au profil de suivi. A l'aide de ce *UserTrackPoint*, un type d'argument, une activité, ou encore un groupe de conditions peuvent être suivis et un évènement est généré lorsque les conditions définies correspondent.

1.11.1 Les profils de suivi

Par défaut, tous les évènements de processus et d'activités sont générés pour tous les types de processus et d'activités. WF ne limite pas le service de suivi uniquement à ce comportement. En effet, l'infrastructure de suivi permet de définir un profil pour chaque type de processus.

WF permet également de définir des profils personnalisés permettant de suivre d'autre type de données. En personnalisant un profil, il est possible par exemple d'avoir un suivi de certains champs ou propriétés du processus. Ce suivi n'est pas inclus dans le comportement de base car il nécessite la connaissance spécifique des champs et propriétés du processus.

1.11.2 Maintenance de la base de données de suivi

Pour pouvoir utiliser le service de suivi fourni avec WF, il est nécessaire d'avoir une base de données SQL Serveur. Le schéma et la logique de cette base sont également fournis par WF. La plupart du temps, les données sont sauvegardées et maintenues

automatiquement sans intervention manuelle. Afin de garantir et de maintenir les performances de la base, une maintenance à l'aide des procédures stockées, fournies également par WF, est également possible.

1.11.3 Partitionnement

Le partitionnement de la base est la maintenance majeure à effectuer. Le partitionnement consiste à déplacer les données de suivi des processus terminés des tables vers un autre groupe de tables définies par une période. Cette période est paramétrable et peut être en jour, en mois ou en année. Ce partitionnement peut se faire de deux manières différentes, soit automatiquement avec la propriété *PartitionOnCompletion* à *true,* soit manuellement si elle a la valeur *false*.

L'exemple suivant montre comment fonctionne le partitionnement avec un intervalle journalier et le partitionnement automatique. Toutes les données de suivi qui sont exécutées aujourd'hui sont sauvegardées dans un groupe de tables ayant une partie de leur nom correspondant à la date du jour. Le lendemain lorsque d'autres données seront sauvegardées, elles le seront dans un autre groupe de tables correspondant à la date du jour suivant.

L'avantage d'un tel partitionnement est que lorsque l'on n'utilise plus ces données, une simple suppression du groupe de table est nécessaire. De ce fait, on évite l'utilisation d'une requête fastidieuse déterminant le groupe de lignes à supprimer.

Le partitionnement en mode manuel nécessite l'utilisation d'une procédure stockée, fournie par WF, à exécuter manuellement. Cette démarche n'est à effectuer que lors de période de maintenance car elle nécessite beaucoup de ressources et ralentit considérablement le système. Malgré cet inconvénient, il est conseillé d'utiliser le partitionnement en mode manuel car le mode automatique réduit potentiellement les performances sur un système chargé.

Afin d'accéder aux données, WF fournit un certain nombre de vues. Celles-ci sont mise à jour à chaque nouvelle entrée de tables. Lorsque les données ne sont plus utiles et qu'il faut les supprimer, l'utilisation d'une procédure permet de détacher les tables et une mise à jour des vues est de nouveau effectuée. A ce stade, les tables ne sont pas supprimées mais simplement détachées des vues. Afin de supprimer physiquement ces tables, il faut employer une autre procédure afin de les supprimer définitivement.

1.12 WF et les Services Web

WF peut être utilisé de plusieurs manières différentes avec des services web. Il est possible de développer un processus et de l'exposer directement en tant que service web. Il est également possible d'appeler directement un service web dans un

processus. WF permet aussi d'héberger le « runtime engine » et d'appeler le processus depuis une application ASP .Net.

L'exposition d'un processus en tant que service web ou encore l'hébergement et l'appel de processus depuis une application ASP.Net ne sont pas couverts dans ce chapitre étant donné que ces notions ne sont pas utilisées dans le cadre de ce projet.

1.12.1 Appel d'un service web depuis un processus

Pour appeler un service web directement au sein d'un processus, WF met à disposition une activité, *InvokeWebServiceActivity*. Afin de pouvoir utiliser cette activité, il faut ajouter la « web reference » au projet contenant le processus. Un proxy est alors généré et permet l'appel au service web. Ensuite il suffit de paramétrer les différentes propriétés de l'activité. La propriété *ProxyClass* est initialisée au nom du proxy, la propriété *MethodName* identifie la méthode du web service que l'on veut appeler et finalement les paramètres d'entrée et de sortie de la méthode peuvent être attachés à des variables du processus ou de l'activité.

2 Présentation de l'implémentation

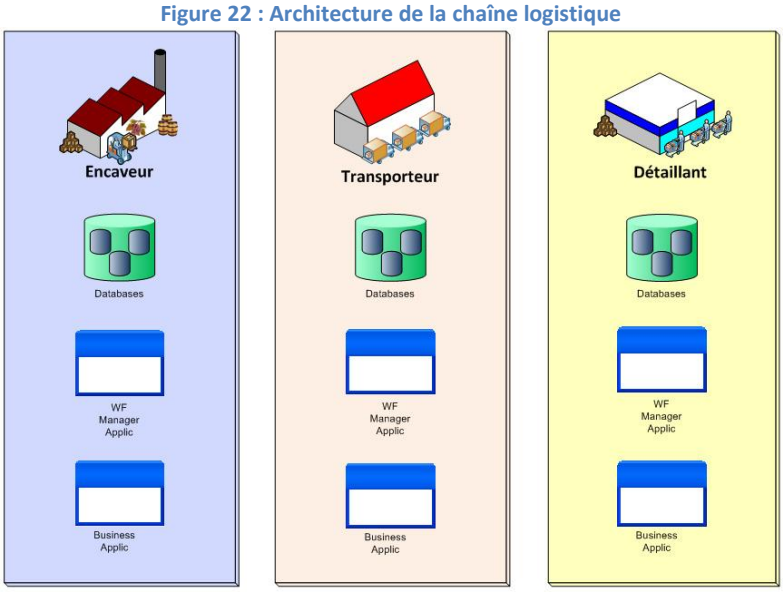
Ce chapitre présente la solution développée pour l'implémentation de la chaîne logistique. La première partie de ce point présente la solution développée pour la chaîne logistique dans son ensemble, le nombre de programmes et quels sont leur utilité. Ensuite une présentation détaillée de la solution d'un des acteurs est faite. Une explication sur les différents modules composant cette solution est exposée.

Ce chapitre n'aspire pas à une présentation technique de l'implémentation de la chaîne logistique mais propose une explication de l'utilité de chacun des modules et services développés. Pour des informations techniques concernant le développement de la solution veuillez vous référer au chapitre 3 Présentation technique, page 57.

2.1 Implémentation globale

2.1.1 Architecture

Figure 22 : Architecture de la chaîne logistique

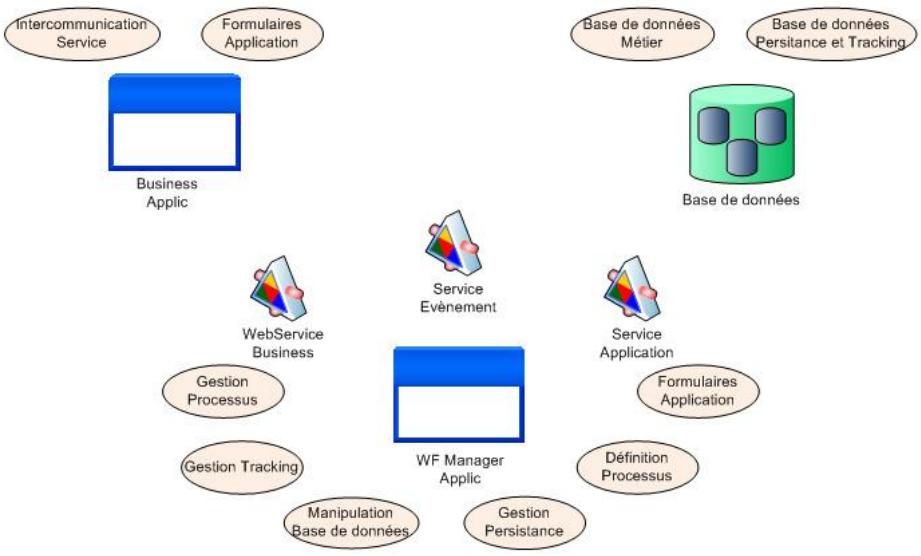


La chaîne logistique se compose, comme décrit au chapitre VII SCENARIO DE LA CHAÎNE LOGISTIQUE en page 19, de trois acteurs. Pour chacun d'eux, une implémentation d'une solution est développée en fonction du métier de leur activité interne. La structure de la solution pour chaque acteur est similaire et comporte les parties suivantes : un logiciel d'administration des processus, un logiciel d'affichage des processus avec les données métier et d'un espace de stockage des données métier et des données des processus. Pour chaque acteur, l'application d'administration est le centre névralgique de la solution. En effet, il est le serveur central permettant le fonctionnement de la solution. Si cette application n'est pas lancée l'acteur ne peut pas utiliser l'application d'affichage des informations métiers. Cette application d'administration manage les processus en cours, communique avec les bases de données, communique avec les services des autres acteurs de la chaîne, informe en temps réel l'application business des changements intervenant au sein des processus et permet également d'afficher l'historique du suivi des processus terminé. L'application business, permettant l'affichage des données métiers, peut être exécutée sur des ordinateurs distants au sein du réseau. Il est possible d'exécuter plusieurs applications business en même temps.

Le point suivant aborde une présentation détaillée de la solution au niveau acteur.

2.2 Implémentation par acteur

Figure 23 : Architecture par acteur



2.2.1 Application d'administration

Définition du processus

Ce module contient toutes les données présentant la définition du processus. Il contient les classes de modélisation du processus, les règles de gestion qui permettent de diriger les actions en fonction des données reçues, ainsi que l'interaction du processus avec l'application d'administration et la gestion des temps d'attente des processus. C'est le cœur de l'application représentant la logique métier de l'entreprise.

Gestion des instances de processus

Ce module permet de manager les processus. Il héberge le moteur d'exécution des processus, contient toutes les références aux services nécessaires pour l'activité de l'entreprise. Par exemple, les données du service de persistance, de suivi, des services locaux permettant à l'application hôte d'interagir avec une instance de processus, et des services internes permettant la personnalisation du moteur d'exécution. Tous les différents états des processus sont gérés dans ce module. Lorsqu'une instance de processus doit être créée ou doit se terminer cela passe automatiquement par le moteur d'exécution des processus.

Gestion de la persistance

La gestion de la persistance est faite au moyen d'une base de données SQL Serveur 2005 dont le schéma et la logique interne permettent de sauvegarder l'état d'une instance de processus à un moment précis durant son exécution. Le module gérant cette base de données doit être référencé auprès du moteur d'exécution des processus de l'application permettant ainsi d'effectuer les activités nécessaires lorsque celui-ci reçoit les évènements en rapport avec la persistance des instances de processus.

Gestion du suivi des processus

La gestion du suivi des processus est également faite au moyen d'une base de données SQL Serveur 2005 contenant un schéma et une logique permettant la sauvegarde de l'exécution complète d'une instance de processus. Ce service permet de sauvegarder, à la période à laquelle le changement s'est effectué au sein de l'instance de processus, tous les changements d'états ainsi que les activités par lequel le processus est passé. Le module gérant cette base de données doit également être référencé auprès du moteur d'exécution des processus. Ce service de suivi des processus offre deux fonctionnalités, la première permet en temps réel de suivre une instance de processus afin de connaître l'état actuel de cette instance, ainsi que l'activité en cours. La deuxième permet, lorsqu'un processus est terminé, de ressortir l'historique de son exécution. Cette dernière fonctionnalité est très intéressante pour les processus qui se sont terminés anormalement afin de discerner à quel moment de l'exécution du processus une erreur est survenue.

Service de communication avec l'application business

Ce service est utilisé par l'application business pour trois usages différents. En effet, ce service est employé lorsque l'application démarre et doit connaître toutes les instances des processus en cours, afin d'afficher les informations métiers pour chacune d'elles. L'application d'administration lui retourne tous les processus figurant au sein du moteur d'exécution des processus. Ce service est également employé lorsque l'application doit récupérer des informations dans la base de données sur des processus terminés. L'application d'administration interroge la base de données métiers et lui transmet les informations demandées. La dernière utilisation de ce service concerne l'interaction utilisateur avec les processus. L'application business permet lorsqu'un processus est dans l'état correspondant de faire continuer l'instance de processus à l'activité suivante.

Service d'évènement pour la communication avec l'application business

Ce service est utilisé par l'application business pour mettre à jour en temps réel les informations des processus en cours. Ce service est employé à chaque fois qu'un

processus change d'état, à chaque fois que l'activité courante d'une instance de processus change et pour informer l'application business de la fin d'une instance de processus courant. Les informations modifiées relatives aux données métiers passent également par ce service évènementiel.

Formulaire de l'application

Ce module regroupe les classes permettant la création du formulaire de présentation de l'application d'administration. Il s'occupe de la gestion du design de l'application suivant les paramètres choisis.

2.2.2 Application business

Formulaire de l'application

Ce module regroupe les classes permettant la création du formulaire de présentation de l'application business. Il s'occupe de la gestion du design de l'application suivant les paramètres choisis.

Intercommunication avec les services de l'application d'administration

Ce module gère l'enregistrement aux services, fournis par l'application d'administration, pour l'affichage des processus en cours et la demande de données relatives.

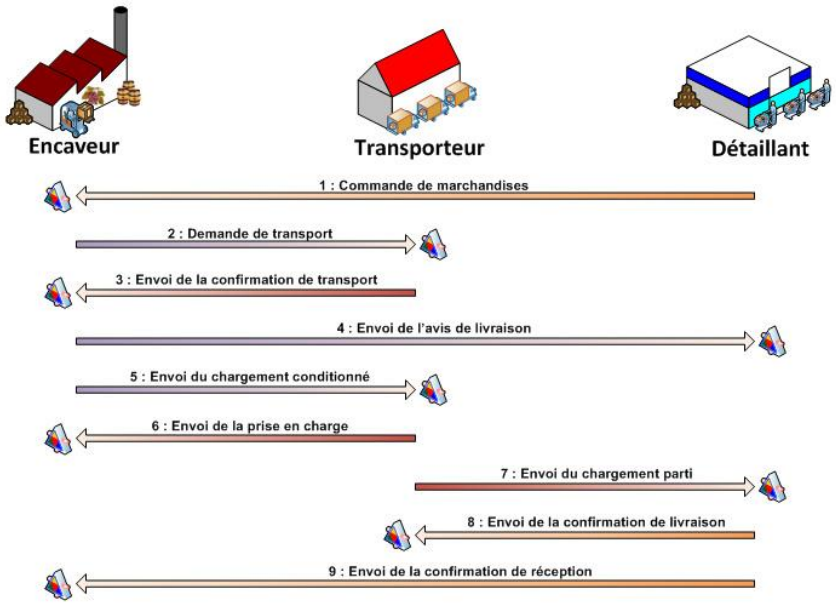
2.2.3 Stockage des données

Le stockage des données est composé de deux bases de données hébergées sur un serveur Microsoft SQL Serveur 2005. La première base de données permet le stockage des données liées à l'activité de l'acteur, par exemple pour l'encaveur, elle contient les commandes reçues, les produits fabriqués, le détail des commandes, etc. La deuxième base de données quant à elle stocke toutes les informations nécessaires à la gestion de la persistance des instances de processus ainsi qu'au suivi de ces mêmes instances de processus. Elle permet de garder l'état des processus en cours lorsque l'application d'administration n'est pas en cours d'exécution, et de garder un suivi des processus terminés.

2.3 Flux transactionnel

2.3.1 Flux du scénario

Figure 24 : Flux des communications du scénario

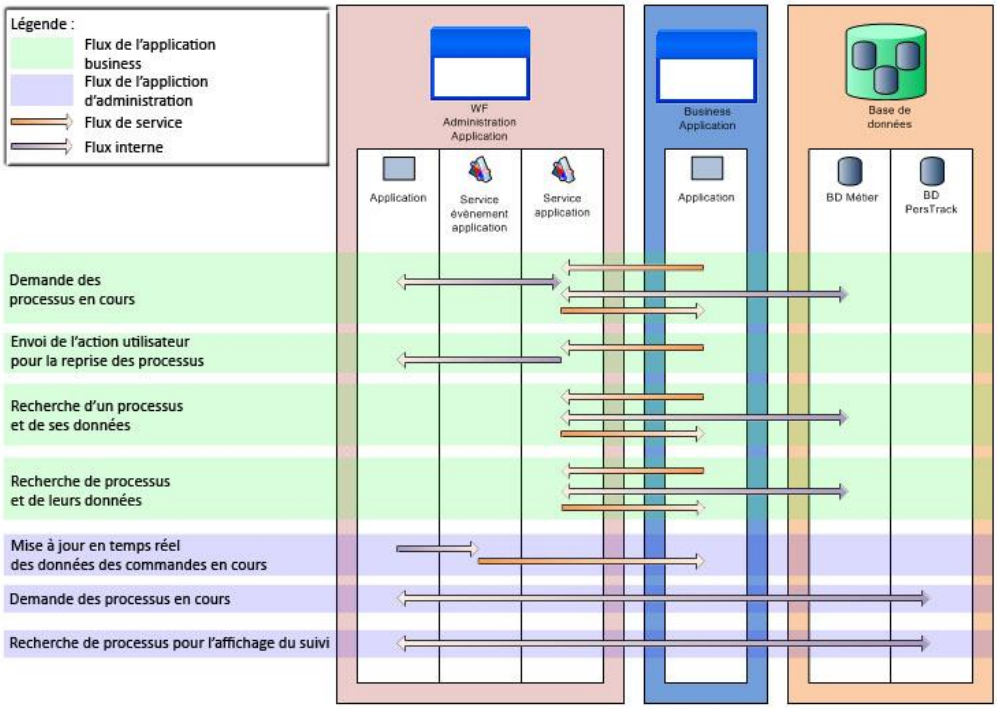


La communication entre les différents acteurs est faite grâce à la technologie des services web. Afin d'assurer l'interopérabilité entre les différents acteurs de la chaîne, si un des acteurs change de technologie pour la gestion de son activité, cela doit se passer de manière transparente pour l'ensemble du scénario. De ce fait, les services web developpés sont conformes au standard assurant cette interopérabilité.

Les interfaces des services web de chaque acteur ont été définies au préalable. La Figure 24 : Flux des communications du scénario présente toutes les communications possibles au sein de ce scénario. Pour de plus amples informations sur les interfaces des services web, le document correspondant se trouve dans les appendices, page 98.

2.3.2 Flux de la solution des acteurs

Figure 25 : Flux des applications



La communication entre les applications est faite avec les services WCF (Windows Communication Foundation). WCF est le package du Framework 3.0 permettant l'implémentation de service de communication. Les services utilisés pour l'échange d'informations entre les deux applications utilisent la couche TCP. Le schéma ci-dessus montre les différents flux échangés entre les applications. Afin de permettre ces échanges, deux services sont implémentés. Le premier permet à l'application business de demander des informations sur les processus à l'application d'administration. Le deuxième quant à lui permet d'envoyer, en temps réel, à toutes les applications business démarrées, les données des processus en cours d'exécution.

3 Présentation technique

Ce chapitre couvre l'aspect technique de la solution développée. Il regroupe différents sujets de l'implémentation : tel que le choix de la technologie utilisée, la présentation de l'environnement de développement, la présentation de la solution développée, et

finalement une explication technique sur tous les modules développés pour l'implémentation de cette chaîne logistique.

3.1 Technologie utilisée

La technologie exigée pour le développement de cette chaîne logistique est « Windows Workflow Foundation ». Cette technologie est une des composantes du Framework de développement DotNet 3.0 et permet la conception d'applications orientées processus.

Faisant partie du Framework 3.0 elle restreint le choix des technologies à employer pour le développement de ce scénario par rapport aux différentes solutions existantes sur le marché. L'implémentation des processus métiers est faite grâce à la plate-forme de développement DotNet proposée par Microsoft. L'implémentation est donc faite en C#.

Les informations permettant le fonctionnement des processus métiers, ainsi que la gestion du cycle de vie des processus de chaque maillon de la chaîne sont stockées dans une base de données SQL Server 2005. Le choix de ce SGBDR est justifié du fait d'une meilleure intercommunication avec les couches « Data Tier » utilisées dans ce développement.

L'accès aux données de la base par les couches « Business Process Tier » est fait grâce au Template Nettiers qui est basé sur les « Microsoft Entreprise Library Application Blocks ». Ce Template construit des objets relationnels pour la base de données reposant sur le Model Driven Design (MDD).

L'intercommunication entre les différentes interfaces de chaque acteur de la chaîne est réalisée par des Web Services DotNet. Cette technologie est choisie afin d'avoir un environnement de développement homogène.

3.2 Structure solution Visual Studio

Le développement étant basé sur le Framework 3.0 de Microsoft, l'environnement de développement ne peut être autre que Visual Studio. La version Visual Studio 2005 est employée pour la réalisation des applications. La configuration nécessaire de Visual Studio 2005, afin de pouvoir développer des applications avec WF, est présentée au chapitre VIII au point 1.2 Pré-requis à la page 22.

Afin de pouvoir travailler sur l'ensemble de la chaîne logistique de manière centralisée, une seule solution Visual Studio pouvant faire référence à d'autre solution Visual Studio est créée. De la même manière, dans un souci de réutilisation de l'existant, le projet représentant le logiciel d'administration des processus, ainsi que le

projet représentant l'application business, affichant les données métiers, sont les mêmes pour les différents acteurs de la chaîne logistique. Lors de l'exécution des applications il suffit alors de choisir dans un fichier de configuration le rôle de l'acteur pour une application donnée et les informations relatives à celui-ci sont affichées.

En appendice en page 98 se trouve une présentation détaillée de la solution démontrant les différents projets, ainsi que de l'utilité des classes relatives.

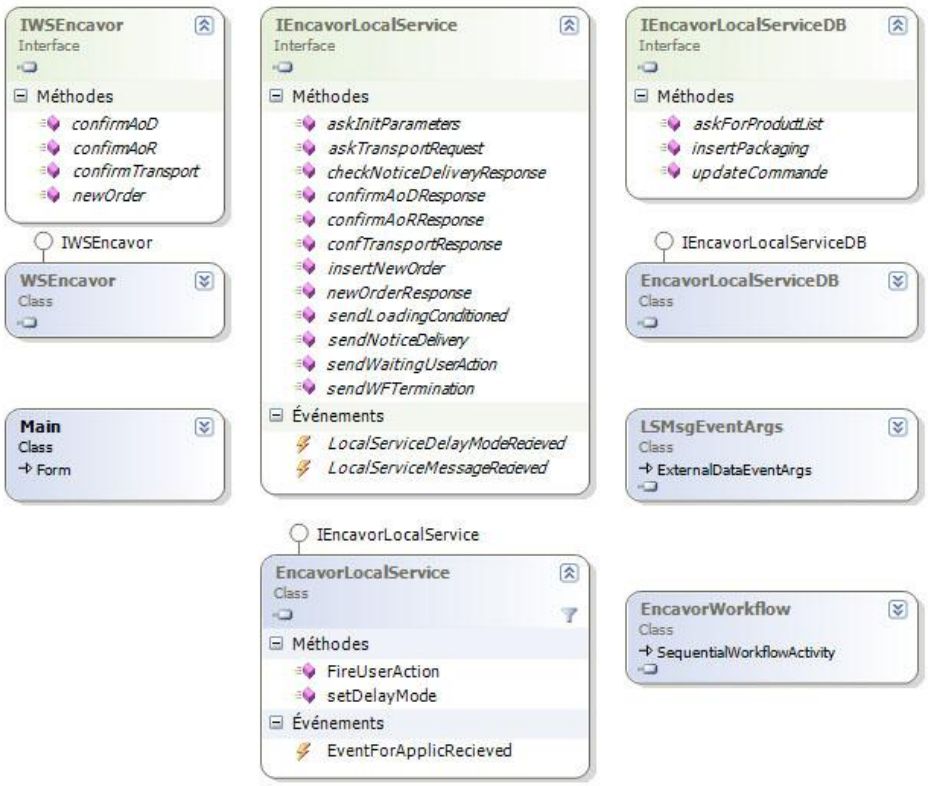
3.3 Implémentation

3.3.1 Processus

Service Local

Le service local est en quelque sorte la plaque tournante de l'implémentation. C'est par ce service que toutes les communications transitent. Le service local peut être défini comme la passerelle entre les instances de processus et « le monde extérieur », à savoir l'application d'administration et tous ses services.

Figure 26 : Passerelle service local



Services locaux référencés :

EncavorLocalService et *EncavorLocalServiceDB* sont les deux services référencés auprès du moteur d'exécution des processus. Ils héritent de leur interface correspondante. Les instances de processus interagissent de deux manières avec ces services locaux. Soit par appel de méthodes pour envoyer des informations vers « le monde extérieur », soit par attente d'un évènement pour recevoir des informations du monde extérieur. Les méthodes et évènements pouvant être utilisés pour la modélisation des processus doivent faire partie des interfaces des services locaux, soit *IEncavorLocalService et IEncavorLocalServiceDB.*

Interaction avec EncavorLocalServiceDB

Ce service est la classe gérant l'accès aux données de la base de données métiers. Il peut être appelé directement par les instances de processus, par les méthodes définies dans l'interface *IEncavorLocalServiceDB* et aussipar le service local central *EncavorLocalService.*

Interaction avec EncavorLocalService

Le service local est utilisé :

Par les instances de processus grâce aux méthodes définies dans l'interface *IEncavorLocalService*.

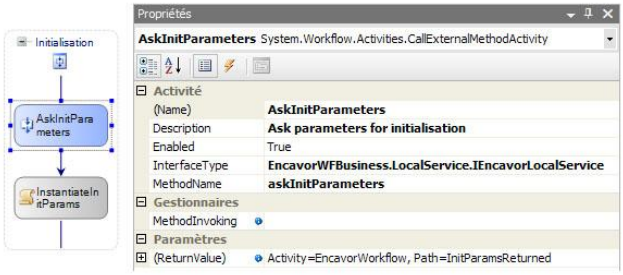
Par le service web de l'encaveur. Chacune des méthodes de l'interface *IWSEncavor* implémentées dans ce service appelle le service local pour atteindre l'instance de processus qui correspond à la commande.

Par l'application d'administration lorsque celle-ci reçoit un évènement utilisateur de la part de l'application business pour informer la bonne instance de processus. Cet appel est fait avec la méthode *FireUserAction* du local service. Egalement lorsque le mode d'attente des instances de processus est modifié avec l'appel de la méthode *setDelayMode* du service local.

Finalement, le service local utilise l'évènement *EventForApplicRecieved* pour envoyer les informations, sur les modifications des données de la commande, à l'application d'administration. L'application d'administration les transmet ensuite à l'application business pour l'affichage des données métiers.

Appel d'une méthode du service

Figure 27 : Activité CallExternalMethod



Prenons l'exemple avec la méthode *askInitParameters*. Cette méthode est utilisée lors de l'initialisation d'une instance de processus pour demander des données de paramétrages. Première étape, il faut une activité de type *CallExternalMethodActivity*, représenté ici par AskInitParameters. Dans l'onglet des propriétés, trois champs sont importants ; InterfaceType, MethodName, ReturnValue. Dans InterfaceType, il faut sélectionner dans la liste l'interface du service. Ensuite, choisir également dans la liste le nom de la méthode correspondante. Finalement, référencer la propriété qui va contenir la valeur de retour de la méthode s'il y en a une. Le service local doit bien entendu implémenter cette méthode et fournir une valeur de retour si nécessaire.

Attente d'un évènement du service

L'implémentation d'une attente d'évènement nécessite plus de travail mais fonctionne selon le même principe. Prenons l'exemple de l'attente du numéro d'identification retourné par le service local. Le service local va employer l'évènement *LocalServiceDelayModeRecieved* pour envoyer la réponse à l'instance de processus. L'objet transmis doit obligatoirement hériter de *ExternalDataEventArgs* pour pouvoir être transmis : représenté ici par la classe *LSMsgEventArgs. LSMsgEventArgs* permet de passer un tableau d'objets en paramètre, de ce fait il peut être employé à chaque fois que l'on veut passer des paramètres à une instance tout au long du processus.

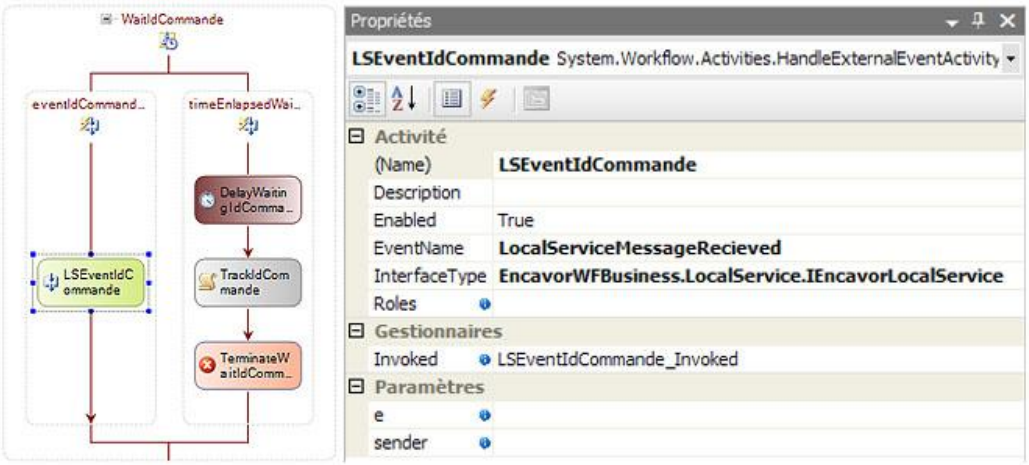
Sur le service local, on envoie l'évènement de manière standard, avec comme paramètre l'objet *LSMsgEventArgs* :

```
LocalServiceMessageRecieved(null, new LSMsgEventArgs(instanceId, c, null, false,
null));
```

Afin de faciliter le transfert, *LSMsgEventArgs* contient un paramètre, ici l'objet « c », faisant référence à l'objet Commande (Entité de la table métiers), contenant tous les attributs pouvant être attachés à une commande.

Le paramètre sender doit être à null sinon l'évènement n'arrive pas à être reçu par la bonne instance. L'objet LSMsgEventArgs doit toujours contenir le Guid (référence unique de l'instance de processus), car l'objet dont il hérite (ExternalDataEventArgs) doit obligatoirement le référencer pour qu'il puisse être envoyé à la bonne instance.

Figure 28 : Activité HandleExternalEvent



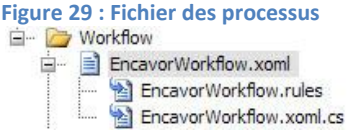
Pour modéliser l'attente de l'évènement, il faut tout d'abord poser une activité de type *ListenActivity* représenté ici par *WaitIdCommande*, permettant d'attendre un évènement. Cette activité comporte toujours deux branches, la première permettant de réceptionner l'évènement et la deuxième permettant de définir la logique si l'évènement n'est jamais attrapé. Sans la deuxième branche, si l'évènement n'arrivait pas, l'instance attendrait indéfiniment et ne pourrait jamais se terminer. Dans la première branche, on pose une activité de type *HandleExternalEvent* permettant de réceptionner l'objet *LSMsgEvent*. De la même manière qu'auparavant on référence l'interface et cette fois-ci le nom de l'évènement et non le nom de la méthode. Ensuite l'attribut *Invoked* permet de définir une méthode qui sera appelée lorsque l'évènement est réceptionné, ceci afin d'extraire les données de *LSMsgEventArgs*. Concernant la deuxième branche, afin de limiter la période d'attente, il suffit de poser une activité de type *DelayActivity* permettant de définir un temps d'attente, et lorsque celui-ci est atteint, de terminer l'instance de processus.

Modélisation

La modélisation des processus est faite dans les projets ***Acteur*WFBusiness** au sein du package *Workflow*. L'explication de la modélisation des processus est présentée par rapport au processus de l'encaveur. L'explication commence par les fichiers nécessaires à la modélisation et finalement une explication sur les différentes activités utilisées. Les schémas complets et détaillés des processus de chaque acteur se trouvent dans les appendices, page 98.

Les fichiers du processus

Figure 29 : Fichier des processus



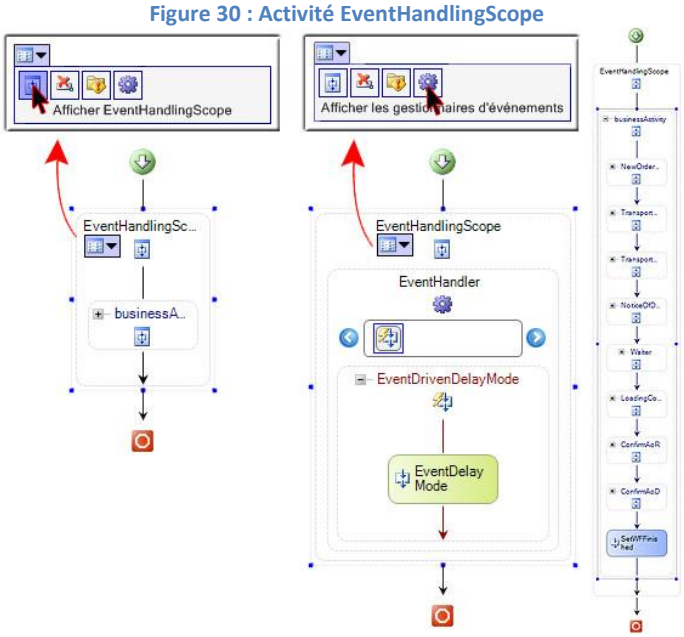
Le développement du processus utilise l'implémentation « code-separated » qui consiste à séparer en fichiers distincts, le design du processus, la

logique et les règles. Lorsque l'on ajoute un nouveau fichier de processus à un projet, le fichier *.xoml* et *.xoml.cs* sont générés. Le fichier *.rules* se crée uniquement si le processus contient une activité nécessitant une condition, typiquement une *IfElseActivity* ou une *PolicyActivity.*

La modélisation du processus se base sur les Use Case définit à la Figure 9 : Activités du scénario en page 21. De ce fait, une *SequenceActivity* est implémentée pour chaque Use Case. De part la fonctionnalité permettant de modifier, durant l'exécution des instances de processus, la manière dont les processus gèrent les temps d'attentes, soit automatiquement, soit par évènement d'un utilisateur, il n'est pas possible de poser directement à la base du processus les *SequenceActivity* représentant les Use Case.

Le « flow » du processus

Figure 30 : Activité EventHandlingScope

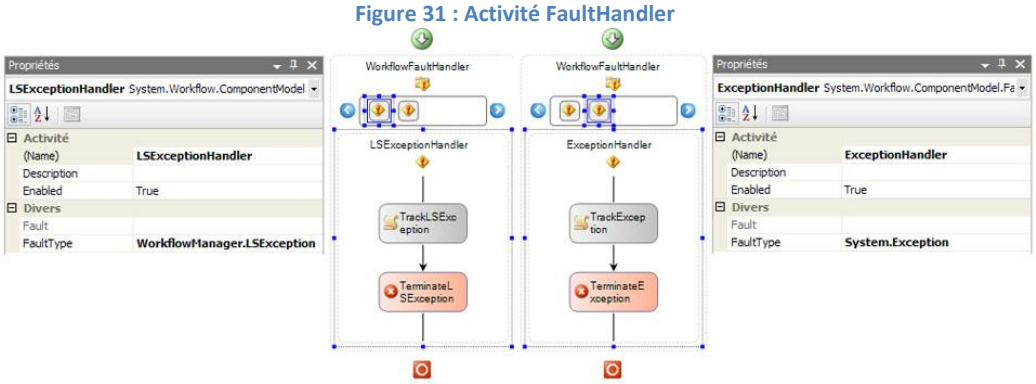


Pour gérer cette fonctionnalité, il faut mettre en place un *EventHandlingScope* qui permet de définir une séquence d'exécution de base, et d'intercepter des évènements durant son exécution. La séquence de base est l'activité *businessActivity.* Le gestionnaire d'évènements contient une *EventDrivenActivity* permettant d'attraper l'évènement servant à modifier la gestion des temps d'attentes. L'activité *EventDelayMode* est de type *HandleExternalEventActivity* et permet de recevoir l'évènement *LocalServiceDelayModeRecieved* qui est levé par le service local à chaque fois que le paramétrage de cette fonctionnalité est effectué. Pour changer l'affichage, comme montré sur l'image ci-dessus, il suffit de cliquer sur l'icône correspondante.

La construction du « flow » de l'entreprise suivant les Use Case se fait dans la séquence *businessActivity*. Deux implémentations supplémentaires sont nécessaires par rapport aux Use Case. La première est la séquence *Waiter* précédent le Use Case de l'envoi du chargement conditionné. Elle permet de suivre le « flow » du processus suivant le mode choisi pour le délai d'attente. La deuxième se trouve en fin de processus et ajoute une activité permettant de sauvegarder les données du processus dans la base de données métier.

Afin de gérer complètement le « flow » du processus, il y a également la gestion des erreurs qui n'apparaît pas directement sur le diagramme. Il faut activer la vue du *FaultHandler*. L'explication de l'activation de cette vue se trouve au point Design Environment, page 29. Lorsqu'une exception intervient au cours de l'exécution d'une instance de processus, c'est cette activité de type *FaultHandler* qui la réceptionne. Cette vue peut être activée pour chaque activité définie dans le processus afin d'agir directement dans l'activité qui l'a levée. Ce qui est très utile afin d'éviter de terminer le processus et lui permettre de continuer son exécution avec l'activité suivante. Pour ce scénario, l'implémentation des exceptions n'est gérée qu'une seule fois et à la racine du processus, ce qui signifie que toutes les activités se terminent les une après les autres en remontant jusqu'à la racine du processus. Cette gestion d'erreur à été implémentée ainsi, car le scénario définit que chaque erreur qui survient au sein d'un processus implique une terminaison de celui-ci.

Figure 31 : Activité FaultHandler



Cette vue permet de définir plusieurs activités de type *FaultHandler* afin de réceptionner différents types d'erreurs, et de permettre une implémentation différente de la gestion de l'erreur pour chaque type. Chaque *FaultHandler* contient une séquence lui permettant d'exécuter une série d'activités, suite à l'erreur qu'il réceptionne. Dans le scénario, deux types d'erreurs sont réceptionnées, une erreur personnalisée de type *LSException* qui est levée lorsqu'une erreur survient dans le service local. Elle contient un message définissant l'erreur. Le deuxième *FaultHandler* réceptionne toutes les autres erreurs et est de type *Exception*. Pour chacune de ces

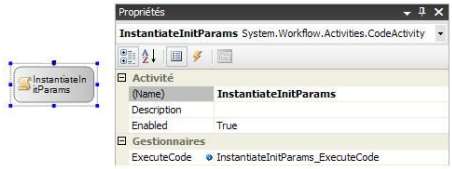
exceptions, la logique est la même, elle contient une *CodeActivity* informant le service de suivi du message de l'erreur et ensuite termine l'instance de processus avec une activité de type *Terminate*. Afin de renseigner le type d'erreur à réceptionner il faut renseigner le champ *FaultType* dans les propriétés du *FaultHandler*.

Les activités utilisées

Ce point répertorie les activités utilisées pour la modélisation du processus ainsi que l'explication de leur implémentation. Les activités *EventHandlingScope, CallExternalActivity, ListenActivity, HandleExternalActivity* et *FaultHandlerActivity* ne sont pas reprises étant expliquées précédemment.

CodeActivity

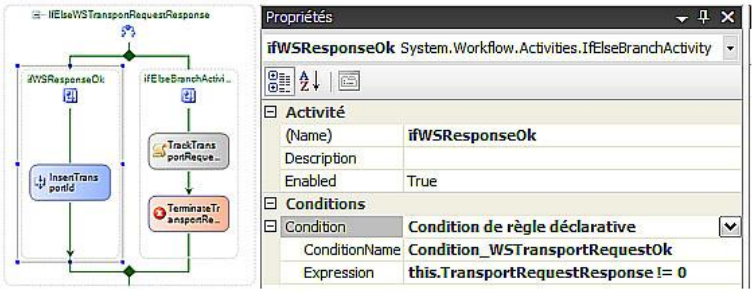
Figure 32 : Activité CodeActivity



Cette activité permet lorsque celle-ci est exécutée d'appeler une méthode. C'est une activité utilisée principalement pour effectuer des calculs ou pour définir des valeurs aux propriétés définies dans le processus. Pour ce faire, il faut renseigner le champ *ExecuteCode* dans la fenêtre des propriétés et définir la méthode à appeler.

IfElseActivity

Figure 33 : Activité IfElse

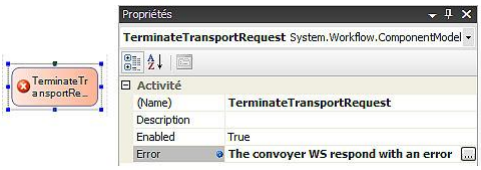


Cette activité est l'une des principales permettant de créer le « fow » du processus. Elle permet de diriger le processus en fonction de conditions définies par des propriétés. L'exemple ci-dessus montre la vérification de la réponse reçue pour la demande de transport. Si elle est valide l'id du transport est inséré dans la base de données, sinon l'erreur est transmise au service de suivi et le processus se termine. Il est possible d'ajouter autant de branches que nécessaires. Toutes les branches

doivent avoir une valeur pour le champ *Condition* dans les propriétés, sauf pour la dernière, car elle est obligatoirement exécutée dans le cas ou les conditions des autres branches ont été refusées.

TerminateActivity

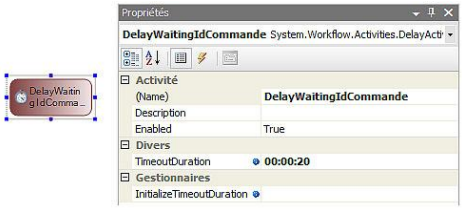
Figure 34 : Activité Terminate



Comme son nom l'indique, cette activité permet de terminer le processus à n'importe quel moment durant l'exécution. Elle contient un champ permettant de spécifier un message d'erreur pouvant identifier la cause de la terminaison du processus.

DelayActivity

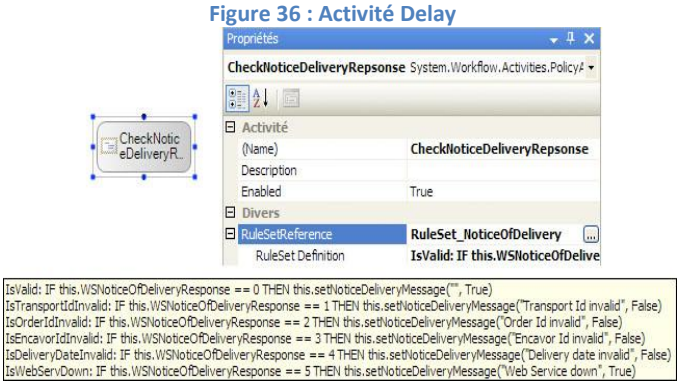
Figure 35 : Activité Delay



Cette activité permet de fixer un temps d'attente pendant l'exécution du processus. Elle est souvent utilisée en collaboration avec l'activité de type *HandleExternalEventActivity*. En effet, elle permet de délimiter le temps maximum pendant lequel le processus attend un évènement d'un service local. Il est possible de définir de deux manières différentes le temps d'attente de cette activité. La première, en référençant le champ *TimeoutDuration* dans les propriétés. La deuxième, en référençant le champ *InitializeTimeoutDuration*, ce champ référence une méthode, cette méthode est appelée juste avant l'exécution de l'activité.

PolicyActivity

Figure 36 : Activité Delay



Cette activité permet l'exécution de règles. Elle contient un assortiment de règles qui sont évaluées par le moteur d'exécution de règles du moteur d'exécution des processus. Dans les propriétés de cette activité, il faut définir le champ *RuleSetReference* par le conteneur de règles et ensuite définir des règles à l'intérieur de ce conteneur, qui correspond au champ *RuleSet Definition*. Au sein du *RuleSet* il est possible de définir l'ordre d'exécution des règles, ainsi que le mode d'évalution. La règle est évaluée selon le concept « Condition-Action Then- Action Else ». L'exemple ci-dessus montre la vérification des paramètres d'entrée de la méthode du service web permettant au transporteur de confirmer la demande de transport. Une règle par paramètre d'entrée de la méthode est définie.

Gestion

L'implémentation basique des processus consistant à instancier une « runtime », de la démarrer et ensuite d'instancier un ou plusieurs processus à la suite, de les démarrer et finalement d'afficher les paramètres de sortie ne suffit pas pour implémenter ce scénario. En effet, il doit être possible d'attendre les informations de plusieurs scénarii et d'obtenir les paramètres de sortie de ceux-ci. Or, l'implémentation décrite ci-dessus ne permet que d'afficher les informations du dernier processus créé. L'implémentation d'un module de gestion des processus est donc requise.

La première étape consiste à créer un objet permettant de stocker l'instance de processus, ainsi que les paramètres de sortie. Cette classe contient une variable permettant de conserver l'exception au cas où le processus se serait terminé anormalement. Elle contient également une variable permettant de stocker l'objet passé en tant que source du *DataGridView* pour l'affichage des données de suivi. C'est la classe *WorkflowInsanceWrapper* du projet *WorkflowManager.*

La deuxième étape est l'implémentation du manager des processus. Cette classe contient la référence de l'objet représentant le moteur d'exécution des processus, ainsi qu'un dictionnaire contenant les processus en cours. Elle implémente également

les méthodes permettant de démarrer un processus de même que la gestion des évènements levés par le moteur d'exécution des processus. C'est la classe *WorkflowRuntimeManager* du projet *WorkflowManager*.

Finalement, l'application d'administration utilise le dictionnaire des processus en cours de la classe *WorkflowRuntimeManager* pour connaître les processus à afficher dans le *DataGridView* et se sert de l'objet *WorkflowDataGrid* de la classe *WorkflowInstanceWrapper* pour afficher les informations de suivi.

Persistance

La persistance des processus est gérée dans l'application d'administration. La persistance est implémentée en utilisant le core service destiné à cet effet. Ce qui a nécessité le développement d'une base de données SQL Serveur 2005. Le développement de cette base est simplifié par WF car il fournit les scripts de création, qui se trouvent dans le dossier du Framework 3.0 ; autant pour le schéma de la base que pour sa logique. Ensuite lorsque la base est créée, il faut enregistrer le service auprès du moteur d'exécution des processus avant le démarrage de ce dernier.

L'ajout du service est implémenté dans le code et non à l'aide d'un fichier de configuration. Voici le code permettant d'ajouter le service au « runtime engine ».

La valeur du *connectionString* se rapporte à l'entrée correspondante dans le fichier de configuration. Ensuite on crée le service en passant le *connectionString* et finalement on ajoute le service au runtime. La variable *_runtimeHoster* et le singleton de gestion de processus qui contient le manager de processus *WorkflowRuntimeManager* de la *CommonLibrary*. *WorkflowRuntime* est la propriété permettant d'accéder à la variable représentant le runtime.

```
String connStringPersTrack =
System.Configuration.ConfigurationManager.AppSettings["EncavorPersTrack"];

_persistenceService = new SqlWorkflowPersistenceService(connStringPersTrack, true,
new TimeSpan(0, 2, 0), new TimeSpan(0, 0, 5));

_runtimeHoster.WorkflowRuntimeManager.WorkflowRuntime.AddService(_persistenceSe
rvice);
```

Comme le moteur d'exécution des processus utilise à la fois le « core » service de persistance et également le « core » service de suivi, le best practice veut qu'il n'y ait qu'une seule base de données pour ces deux services. De ce fait, il faut également utiliser le core service *CommitWorkBatchService* en mode *SharedConnection* afin d'optimiser les transactions pour cette base de données. En page 44, le point Gestion de la persistance contient une explication de cette implémentation.

Suivi

Afin d'assurer le suivi des processus, selon les fonctionnalités qui doivent être implémentées, il est nécessaire de développer deux services de suivi différents. Le premier, utilisant le « core » service de WF pour la gestion de la persistance, permet de remonter les informations de suivi concernant les processus terminés. Le deuxième quant à lui, permet de suivre en temps réel le suivi des processus en cours d'exécution. Pour ce deuxième service, une implémentation d'un service personnalisé est nécessaire car celui-ci n'est pas fourni avec WF. Chaque acteur contient une implémentation d'un profil de suivi permettant d'affiner les informations du suivi selon le métier de l'acteur.

Core service de suivi

Pour l'implémentation de ce service, comme cité dans le point précédent concernant le service de persistance, l'implémentation du core service de suivi est faite dans la même base de données que le service de persistance. WF fournit également le script SQL permettant la création du schéma des tables ainsi que de la logique. Ces fichiers se trouvent dans le dossier du Framework 3.0. L'enregistrement du service auprès du moteur d'exécution des processus s'effectue de façon identique, comme tout core service.

```
String connStringPersTrack =
System.Configuration.ConfigurationManager.AppSettings["EncavorPersTrack"];

_trackingService = new SqlTrackingService(connStringPersTrack);
_trackingService.PartitionOnCompletion = false;
_trackingService.UseDefaultProfile = false;
EncavorWFBusiness.ProfileTracking.EncavorTrackingProfile.GetEncavorTrackingProfile();

_runtimeHoster.WorkflowRuntimeManager.WorkflowRuntime.AddService(_trackingServic
e);
```

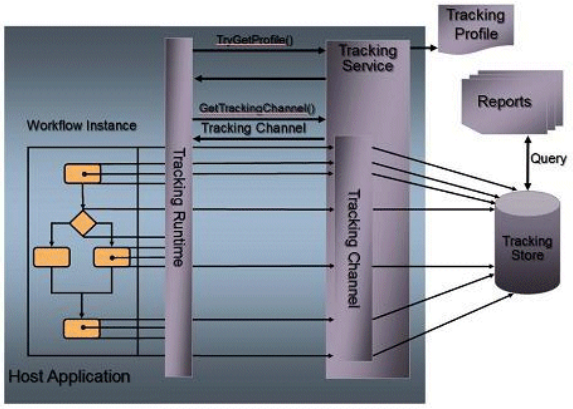
La propriété *PartitionOnCompletion* permet de définir le partitionnement automatique ou non des tables de suivi. Des informations supplémentaires sur le partitionnement de la base se trouvent en page 50 sous le point Partitionnement. La propriété *UseDefaultProfile* est définie à *false*, car chaque acteur de la chaîne implémente un profil de suivi personnalisé. La ligne suivante charge le profil devant être utilisé. Les profils sont présentés dans les paragraphes suivants.

Service personnalisé de suivi

Comme indiqué précédemment, il permet de suivre en temps réel les processus en cours. Il est implémenté spécifiquement pour l'application d'administration car il n'est pas inclus dans WF. Du fait que ce service est également enregistré auprès du moteur d'exécution des processus il doit contenir les interfaces et méthodes utilisées par le moteur d'exécution.

Afin de comprendre comment ce service doit être implémenté il est nécessaire de connaître le fonctionnement du suivi au sein du moteur d'exécution des processus.

Figure 37 : Service de suivi et le moteur d'exécution



Source : ([MSDN01] WebSite)

Ci-dessous sont présentés les différents composants permettant le fonctionnement du service de suivi avec le moteur d'exécution des processus : (Source : MSDN, http://msdn2.microsoft.com/en-us/library/bb264459(vs.80).aspx).

- "**Tracking runtime.** The runtime is responsible for the initialization of the tracking, such as setting up the appropriate tracking profile, and once a workflow instance is running, it picks up events, such as status change, and passes them on to the tracking channel.
- **Tracking channel.** The tracking channel is used to deliver information from the runtime to the appropriate tracking-service store. For the **SqlTrackingService**, every workflow instance has a separate tracking channel for each registered service, which provides developers with an architecturally simple model that makes it easy to develop services that do not need to worry about locking.
- **Tracking service.** The tracking service is the core element of the tracking infrastructure, and is responsible for supplying tracking channels and tracking profiles to the workflow runtime engine.
- **Tracking profile.** The tracking profile determines what information will be tracked and, as such, acts as a filter to reduce the amount of information generated, ultimately making it easier to manage tracking data."

L'implémentation du service est faite grâce aux deux classes du projet *WorkflowManager*, *CustomTrackingChannel* et *CustomTrackingService*.

La classe *CustomTrackingService* représente le service et doit hériter de la classe *TrackingService.* Pour pouvoir communiquer avec le moteur d'exécution, elle doit contenir les méthodes « override » suivantes.

```
protected override TrackingChannel GetTrackingChannel(TrackingParameters
parameters)

protected override TrackingProfile GetProfile(Guid workflowInstanceId)

protected override TrackingProfile GetProfile(Type workflowType, Version
profileVersionId)

protected override bool TryGetProfile(Type workflowType, out TrackingProfile profile)

protected override bool TryReloadProfile(Type workflowType, Guid
workflowInstanceId,out TrackingProfile profile)
```

La première méthode permet de référencer le *channel* qui sera utilisé par le service. Les autres sont utilisées par le moteur d'exécution pour charger le bon profil de suivi.

La classe *CustomTrackingChannel* définit le *channel* utilisé pour délivrer les informations de suivi lors d'évènements correspondants du moteur d'exécution. La méthode *GetTrackingChanne,l* vu précédemment dans la classe du service, retourne une instance de la classe *CustomTrackingChannel.* Cette classe de *channel* doit également contenir des méthodes « override ».

```
protected override void Send(TrackingRecord record)

protected override void InstanceCompletedOrTerminated()
```

A chaque fois que le moteur d'exécution reçoit un évènement de suivi, la méthode *Send* est appelée. La classe de suivi traite les informations reçues dans la variable record de type *TrackingRecord.* Ce type permet de différencier les évènements et, ainsi de connaître s'il s'agit d'un évènement de processus, d'activité ou encore un évènement utilisateur. C'est donc dans cette classe que les données sont formatées suivant l'évènement reçu. Ensuite, quand les données sont formatées, elles sont renvoyées au service de suivi personnalisé qui va envoyer les informations pour l'affichage dans l'application administration. Chaque objet, référencé dans le *DataGridView* représentant une instance de processus, contient une variable de type *StringBuilder* permettant de stocker les données de suivi, afin de les afficher dans le composant *WebBrowser* servant à l'affichage du processus actuellement sélectionné.

L'enregistrement du service auprès du moteur d'exécution se fait de la même manière que le « core » service. Lors de la construction du service, la propriété référençant le profil utilisé qui sera demandé par le moteur d'exécution est définie.

```
_custTrackingService = new CustomTrackingService();

_custTrackingService._trackingProfile =
EncavorWFBusiness.ProfileTracking.EncavorTrackingProfile.MyEncavorTrackingProfile;

_runtimeHoster.WorkflowRuntimeManager.WorkflowRuntime.AddService(_custTrackingS
ervice);
```

Profil de suivi des acteurs

Chaque acteur contient son profil personnalisé. Cette implémentation permet de ressortir des informations renseignant sur le métier de l'activité de l'entreprise. Avec un profil il est possible d'ajouter des annotations spécifiques à l'acteur ainsi que de ressortir les valeurs des propriétés définies dans le processus.

Une classe profil doit être définie pour chaque acteur, classe *ActeurTrackingProfile*. Ces classes sont implémentées dans le projet de la logique métier de chacun des acteurs, à savoir le projet ***Acteur*WFBusiness** dans le package *ProfileTracking.*

Cette classe contient une méthode *GetTrackingProfileModified* permettant de définir les valeurs personnalisées à rajouter. Afin de les ajouter, il faut implémenter des *TrackPoint* en définissant leur positionnement dans les données de suivi.

Par exemple, le code ci-dessous, lié au métier de l'encaveur, ajoute à l'exécution de l'activité « SequenceActivity » l'affichage des variables « InputRetailer » et « InputOrderItems ». Ce *TrackPoint* affiche également une annotation.

```
ActivityTrackingLocation location = new ActivityTrackingLocation();
location.ActivityTypeName = "SequenceActivity";
location.ExecutionStatusEvents.Add(ActivityExecutionStatus.Executing);

ActivityTrackPoint point = new ActivityTrackPoint();
point.Extracts.Add(new WorkflowDataTrackingExtract("InputRetailer"));
point.Extracts.Add(new WorkflowDataTrackingExtract("InputOrderItems"));
point.Annotations.Add("Input parameters recieved");
point.MatchingLocations.Add(location);

profile.ActivityTrackPoints.Add(point);
```

Afin d'enregistrer et de ressortir les profils dans les bases de données correspondantes, ces classes de configuration du profil de l'acteur utilisent la classe *TrackingProfileHelper* se trouvant dans le projet *WorkflowManager*. Cette classe contient des méthodes permettant de retrouver, de supprimer ou de modifier un profil dans la base de données.

Enregistrement des services de 'application administration

Le fichier de configuration permet de définir l'acteur affiché. Ce qui va déterminer quels services de l'application seront instanciés. Ces services sont référencés comme

variables de classe dans la classe *Main* du projet *AdminApplication.* Certains services peuvent utiliser une variable commune selon l'acteur choisi et d'autres doivent définir une variable pour chaque acteur de la chaîne. Les services locaux de chaque acteur doivent avoir leur variable propre étant donné que leur référence vient de projets différents. Le service de persistance, les deux services de suivi, ainsi que le service de connexion partagée, utilisent une variable unique quelque soit l'acteur paramétré.

```
#region Service Variables
private SqlTrackingService _trackingService;
private CustomTrackingService _custTrackingService;
private SqlWorkflowPersistenceService _persistenceService;
private SharedConnectionWorkflowCommitWorkBatchService _sharedConnService;
#endregion

#region Encaveur LocalServices Variables
private static EncavorLocalService _encavorLocalService;
private static EncavorLocalServiceDB _encavorLocalServiceDB;
#endregion

#region Transporteur LocalServices Variables
private static ConvoyerLocalService _convoyerLocalService;
private static ConvoyerLocalServiceDB _convoyerLocalServiceDB;
#endregion

#region Detaillant LocalServices Variables
private static RetailerLocalService _retailerLocalService;
private static RetailerLocalServiceDB _retailerLocalServiceDB;
#endregion
```

Lorsque l'application est lancée, sur l'évènement *OnLoad* de l'application, les services sont instanciés. Ces services sont implémentés en code et non dans le fichier de configuration. Pour savoir quels services instanciers, l'application utilise la valeur *ActorRole* dans le *appSettings* du fichier de configuration.

3.3.2 Communication

L'application devant utiliser la nouvelle technologie de Microsoft faisant partie du Framework 3.0, et la volonté d'avoir un environnement de développement homogène, les services web et la communication, entre les applications, est faite en utilisant le Framework DotNet. Ces services sont développés avec le Framework 3.0 ce qui permet ainsi, également, la découverte de cette nouvelle technologie apportée avec ce Framework. Etant donné la nouveauté de cette technologie, avant la présentation des services développés pour l'implémentation d'une solution, un bref exposé de la technologie est réalisé. La réalisation de cette implémentation se base sur les sources d'informations suivantes : ([WCFMS] WebSite) et ([MSDN01] WebSite)

Présentation des services WCF 3.0

Windows Communication Foundation (WCF) est un SDK pour le développement et le déploiement des services sur Windows. WCF propose une interopérabilité avec les autres services. Il y a plusieurs façons d'héberger ces services. Les services doivent être hébergés dans un processus Windows appelé *host process.* L'hôte peut être de différents types. Il peut être fourni par IIS, les services d'activation Windows (WAS) sur Windows Vista ou également comme partie de l'application développée. Les services de cette solution ont été développés avec cette dernière. Les services sont hébergés directement dans l'application d'administration. Cet hébergement n'est pas la meilleure solution d'héberger les services car il n'offre pas les fonctionnalités nécessaires pour une exécution et un maintient du fonctionnement des services sur une longue période comme par exemple dans un environnement de production. Ce type d'hébergement est souvent utilisé pour des applications de démonstration ou de test.

Les services WCF se basent sur le concept ABC, *address, binding, contract*. Ce triumvirat forme l'*Endpoint* du service. Un *Endpoint* doit posséder ces trois notions pour pouvoir être fonctionnel. *Address* est comme son nom l'indique l'adresse sur laquelle le service est atteignable. Il existe différent *binding* possible, il définit le comportement du service, s'il est synchrone, asynchrone, son protocole, la sécurisation du service, etc. Le *contract* définit l'interface du service, les méthodes mises à disposition.

Pour cette solution, étant donné que les services sont hébergés au sein de l'application et qu'ils ne peuvent pas écouter sur le même port, il faut configurer le réseau afin de permettre le transfert de données sur les ports utilisés.

DynamicProxy

Le scénario de la chaîne logistique est déterminé et il fixe le nombre d'acteurs à trois. Un encaveur, un transporteur et un détaillant, mais une des fonctionnalités devant être implémentée consiste à pouvoir ajouter aisément un nouveau détaillant. Il est parlé ici uniquement d'ajouter un acteur de type détaillant car celui-ci n'engendre aucune modification de l'activité des autres acteurs. En effet, l'ajout d'un nouveau transporteur modifierait la logique interne de l'activité de l'encaveur par exemple. Il devrait y avoir une activité supplémentaire attendant une interaction humaine permettant le choix du transporteur. La modification nécessaire pour l'ajout d'un nouveau détaillant devrait uniquement être son insertion dans la base de données de chaque acteur afin de disposer de ses informations personnelles.

Cette possibilité tout à fait utile et qui paraît anodine ne l'est pas tout à fait pour le Framework DotNet et l'implémentation des services web. En effet, pour pouvoir contacter un service web avec un client DotNet il doit disposer des informations nécessaires sur le service pour pouvoir le contacter. Avec DotNet la mise à disposition

de ces informations de l'application cliente se fait de deux manières différentes. La première est un outil intégré à Visual Studio permettant de générer une classe contenant les informations du service web et les méthodes utilisables. La deuxième possibilité et l'utilisation d'un utilitaire créant cette classe proxy, ensuite il faut incorporer cette classe au projet de l'application cliente. Or, cette démarche n'est pas des plus agréables dans un environnement de production. En effet, elle nécessite une recompilation complète du projet et une redistribution de ce nouveau « build » à tous les utilisateurs. De plus, les classes d'appels du web service ne référençant pas cette nouvelle classe, une modification du code serait nécessaire.

Cette solution n'étant pas adéquate, une solution différente devait être trouvée. Pour que l'implémentation de cet ajout d'acteur puisse être la plus simple possible uniquement la connaissance de l'adresse du service web doit être connue. Cette implémentation est possible grâce à la création dynamique de ce proxy pour l'application cliente.

La génération dynamique du proxy ne se base pas sur un fichier de configuration ou d'une classe précompilée du proxy. Elle accepte la création d'un client permettant d'atteindre un service web lorsque l'application est en cours d'exécution. Ce proxy dynamique utilise le *MetadataResolver* pour télécharger les metadata du service web et ensuite utilise le *WsdlImporter* pour créer le *contract* et le *binding* durant l'exécution. Ensuite, le *DynamicProxy* peut être utilisé pour appeler les méthodes du service web en utilisant la réflexion.

L'exemple ci-dessous illustre l'appel d'une méthode d'un service web avec cette implémentation.

Création du proxy avec la WSDL Uri

```
DynamicProxyFactory factory = new
DynamicProxyFactory("http://localhost:8080/TD2007/DynamicProxy?wsdl");
```

Création du *DynamicProxy* soit par un *Endpoint* soit par un *Contract.*

```
DynamicProxy proxy = factory.CreateProxy("IWSEncavor");

//OR

DynamicProxy proxy = factory.CreateProxy(endpoint);
```

Invocation de la method du service web

```
int result = (int)proxy.CallMethod("newOrder", retailerid,orderItems);
```

Fermeture du proxy

```
proxy.Close() ;
```

Service web métiers

Le service web métiers des acteurs est le service qui définit la logique des communications entre les différents acteurs au cours d'un scénario. Afin de pouvoir implémenter le scénario et assurer son fonctionnement entre les différents acteurs, les méthodes, ainsi que les retours de méthodes mis à disposition sont connus d'avance. Ces interfaces sont définies dans le document « TD-BPMS – Définitions des interfaces techniques » consultables dans les appendices : page 98.

Afin d'assurer l'interopérabilité avec les différents acteurs, au cas ou ils emploient une technologie différente, ce service doit être compréhensible par les différentes technologies. De ce fait le choix du *binding* est très important. Parmi les *binding* possibles dans WCF, deux sont retenus, le *Basic binding* et le *Web Service (WS) binding.* Les autres ne peuvent être implémentés pour ce service et ne sont donc pas discutés dans ce point. Le premier présente le service comme un service héritant des services ASMX, comme cela les anciens clients peuvent travailler avec les nouveaux services. Utilisé par le client ce *binding* permet aux nouveaux clients de travailler avec les anciens services ASMX. Le deuxième *binding* utilise http et https pour le transport, et permet d'offrir différentes fonctionnalités comme, fiabilité, sécurité et transaction. Etant donné que le service n'est pas encore référencé dans d'autres applications clientes, nous n'avons pas d'anciens clients, donc nous pouvons utiliser le *binding Web Service (WS) binding.*

L'implémentation du service au sein de l'application d'administration peut être divisée en quatre parties.

La première consiste en la création de l'interface du service qui sera référencée comme *contract* dans l'Endpoint. Cette interface définit toutes les méthodes pouvant être employées par les clients de ce service (la classe **IWS*Acteur***).

La deuxième consiste à créer la classe du service de l'application héritant de cette interface. Cette classe définit les méthodes de l'interface et la logique présente dans chaque méthode. Ici, pour chaque méthode du service web, une méthode du service local des processus est appelée, renvoyant comme retour la réponse du traitement de l'information par l'instance de workflow correspondante. Le contexte de ce service est en mode *PerCall*. Cela signifie que pour chaque appel d'une méthode, une nouvelle instance du service est créée, et lorsque le client ferme le proxy cette instance est détruite (la classe **WS*Acteur***).

La troisième étape est la configuration du service. Cette étape peut être faite de deux manières différentes, soit en code, soit par un fichier de configuration. La deuxième variante est choisie car elle permet une plus grande flexibilité. Elle permet une modification du comportement du service sans devoir recompiler la solution.

La configuration par fichier de configuration est faite au moyen d'un nœud *servicemodel* dans le fichier de configuration de l'application (fichier **App.config**). La

configuration du service au moyen d'un fichier de configuration est expliquée au point Configuration d'un service au moyen d'un fichier de configuration à la page82.

Ci-dessous le paramétrage du fichier de configuration pour ce service :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <services>
            <service name="EncavorWFBusiness.Service.WSEncavor"
                     behaviorConfiguration="myBehavior">
                <host>
                    <baseAddresses>
                        <add baseAddress="http://192.168.1.119:8080/EncavorService/"/>
                    </baseAddresses>
                </host>
                <endpoint address="" binding="wsHttpBinding"
                          contract="EncavorWFBusiness.Service.IWSEncavor"/>
                <endpoint address="http://192.168.1.119:8080/EncavorService/MEX"
                          binding="mexHttpBinding" contract="IMetadataExchange"/>
            </service>
            </services>
        <behaviors>
            <serviceBehaviors>
                <behavior name="myBehavior">
                    <serviceMetadata/>
                </behavior>
            </serviceBehaviors>
        </behaviors>
    </system.serviceModel>
</configuration>
```

Finalement la dernière étape consiste à héberger le service dans l'application d'administration. Pour ce faire, il suffit de créer une instance de la classe *ServiceHost* et de lui passer en paramètre le service développé.

```
ServiceHost host = new ServiceHost(typeof(EncavorWFBusiness.Service.WSEncavor)) ;
```

Puis de démarrer le service.

```
host.Open() ;
```

Service de communication avec l'application business

Ce service permet à l'application business de demander des informations sauvegardées dans les bases de données ou encore de transmettre une interaction humaine à une instance de processus. Ces demandes ne devant être faites que ponctuellement, à des moments définis et gérables dans l'application, ce service à un contexte *PerCall* comme le service métier des acteurs. Pour rappel, cela signifie, qu'une instance de service est créée pour chaque appel, quelque soit la méthode demandée et que dès que le client ferme le proxy, l'instance est détruite. Aucune donnée n'est gardée sur le service pour un autre appel de méthode venant du même client.

La différence de ce service par rapport au service métier se situe au niveau du *binding*. En effet, étant donné que ce service est utilisé qu'entre l'application d'administration et business et ne requiert aucune interopérabilité avec d'autres technologies, un *binding* plus adéquat fourni par WCF peut être employé. Il s'agit du *TCP binding*. Il est fourni par la classe *NetTcpBinding* et est utilisé spécialement pour la communication

sur internet entre des machines. Il permet de plus une plus grande variété de fonctionnalités comme fiabilité, sécurité et transaction et est optimisé pour la communication de WCF à WCF.

Il faut également rajouter un paramètre dans le fichier de configuration des services concernant le paramétrage de ce *binding* et changer et modifier le protocole de l'adresse de base.

Ci-dessous le paramétrage du fichier de configuration pour ce service :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <services>
            <service name="ServiceManager.ServiceForBusinessApplic"
                     behaviorConfiguration="myBehavior">
                <host>
                    <baseAddresses>
                        <add baseAddress="net.tcp://192.168.1.119:8082/ServiceForBusinessApplic/"/>
                    </baseAddresses>
                </host>
                <endpoint address="MEX" binding="mexTcpBinding" contract="IMetadataExchange"/>
                <endpoint address="" binding="netTcpBinding"
                          bindingConfiguration="netTcp" contract="ServiceManager.IServiceForBusinessApplic"/>
            </service>
        </services>
        <behaviors>
            <serviceBehaviors>
                <behavior name="myBehavior">
                    <serviceMetadata/>
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <bindings>
            <netTcpBinding>
                <binding name="netTcp">
                    <security mode="None" />
                </binding>
            </netTcpBinding>
        </bindings>
    </system.serviceModel>
</configuration>
```

Service d'évènement pour la communication avec l'application business

Ce service diffère quelque peu des autres services de l'application. En effet, les autres services de la solution sont appelés ponctuellement durant l'utilisation de l'application. Un appel d'une méthode reçoit son retour et le service se termine. Pour celui-ci, il faut développer un service différent parce que les informations doivent être affichées sur l'application business, mais proviennent de l'application d'administration. Ce n'est pas l'application business qui demande les informations mais bien l'application d'administration qui les transmet à un moment non ponctuel suivant l'état des instances de processus. Etant donné qu'il peut y avoir plusieurs applications business s'exécutant simultanément sur différents postes, le service doit connaître toutes les instances de ces applications. C'est une implémentation bien plus complexe que les services qui appellent une méthode et attendent la réponse.

Cette implémentation est résolue à l'aide de *callback.* Le principe du callback consiste à appeler une méthode d'un service mais de ne pas attendre une réponse de cette

méthode. La réponse est renvoyée tardivement au client. Cette notion est employée lorsque le service à besoin de temps ou doit attendre un autre évènement extérieur pour renvoyer la réponse au client du service. Ce service peut fonctionner dans les deux sens entre le client et le service et chaque partie appelle et reçoit des demandes entre elle.

Dans WCF l'utilisation des callback en duplex est faite avec des évènements. Les évènements permettent au client d'être notifié de quelque chose qui survient du côté du service, typiquement lorsque le service assure un suivi de données. Dans ce contexte, le service envoyant les évènements est appelé le *publisher* et le client recevant l'évènement le *subscriber*. Bien que les évènements dans WCF soient des opérations en *callback*, cela implique une relation moins importante entre *publisher* et *subscriber* comparé à une relation entre client et service. Le service publie le même évènement à plusieurs *subscriber.* Généralement, le *publisher* ne tient pas compte de l'ordre d'envoi des évènements aux différents *subscriber*. Le *publisher* sait uniquement qu'il doit envoyer des évènements aux *subscriber*, il ne tient pas compte d'une erreur pouvant intervenir du côté du *subscriber* lors de la réception de l'évènement. De plus, il ne tient pas compte des retours de méthode. De ce fait, les méthodes du service d'évènement doivent être de type *void* et doivent être notifiées comme méthodes *OneWay*.

```
[OperationContract(IsOneWay = true)
void WFStateChanged(Guid guid, string state);
```

L'implémentation de l'interface du service comprend trois notions. L'interface du service, l'interface de la classe *callback* et une énumération permettant de définir les types d'évènements auxquels le *subscriber* peut s'enregistrer (la classe **IServiceForBusinessApplic**).

L'interface du service référence dans l'attribut *CallbackContract* l'interface de callback. Elle contient deux méthodes permettant de s'inscrire ou se désinscrire au service d'évènement.

```
[ServiceContract(Name = "WFEventService",Namespace =
"http://probstep.workflowfoundation.ch/td2007/ServiceForBusinessApplicEvent",
CallbackContract = typeof(IWFEvents),SessionMode = SessionMode.Required)]
```

L'interface de callback contient les méthodes permettant au service d'envoyer les évènements aux *subscriber* enregistrés.

```
public interface IWFEvents
{
[OperationContract(IsOneWay = true)]
void WFStateChanged(Guid guid, string state);
```

```
[OperationContract(IsOneWay = true)]
void WFEnded(Guid g);

[OperationContract(IsOneWay = true)]
void WFCreatedEncavor(Guid g, WFEncavor.BLL.Commande order);

//etc…

}
```

L'énumération contient les différents types d'évènements auxquels on peut s'enregistrer.

```
public enum EventType
{
StateChange = 1,
CreatedEncavor = 2,
CreatedConvoyer = 3,
CreatedRetailer = 4,
Ended = 5,
ChangeEncavor = 6,
ChangeConvoyer = 7,
ChangeRetailer = 8,
NoticeDelivery = 9,
WaitingUserAction =10
EventEncavor =
StateChange|CreatedEncavor|Ended|ChangeEncavor|NoticeDelivery|WaitingUserAction
}
```

La configuration du service dans le fichier de configuration se fait de la même manière que pour le service TCP vu précédemment.

Ci-dessous le paramétrage du fichier de configuration pour ce service :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <services>
            <service behaviorConfiguration="myBehavior"
                     name="ServiceManager.EventPublisherForBusinessApplic">
                <host>
                    <baseAddresses>
                        <add baseAddress="net.tcp://192.168.1.119:8083/EventPublisherForBusinessApplic"/>
                    </baseAddresses>
                </host>
                <endpoint address="" binding="netTcpBinding" bindingConfiguration="netTcp"
                          contract="ServiceManager.IServiceForBusinessApplicEvent"/>
                <endpoint address="MEX" binding="mexTcpBinding" contract="IMetadataExchange"/>
            </service>
        </services>
        <behaviors>
            <serviceBehaviors>
                <behavior name="myBehavior">
                    <serviceMetadata/>
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <bindings>
            <netTcpBinding>
                <binding name="netTcp">
                    <security mode="None" />
                </binding>
            </netTcpBinding>
        </bindings>
    </system.serviceModel>
</configuration>
```

L'hébergement de ce service se fait de la même manière que pour les autres services de cette solution.

Le service gère la publication des évènements à l'aide de *delegate*. Chaque *delegate* correspond à la signature d'un type d'évènements auxquels on peut s'enregistrer.

```
static GenericEventHandler<Guid, string> m_WFStateChanged = delegate { };

public void Subscribe(EventType mask
{
   IWFEvents subscriber =
OperationContext.Current.GetCallbackChannel<IWFEvents>();
   if ((mask & EventType.StateChange) == EventType.StateChange)
     m_WFStateChanged += subscriber.WFStateChanged;
}
```

Le service contient une méthode statique pour chaque type d'évènements permettant de les publier aux *subscriber*.

```
public static void FireEventStateChange(Guid guid, string state)
      {
         m_WFStateChanged(guid, state);
      }
```

Après avoir généré le proxy dans l'application business, l'enregistrement au service de la part du *subscriber* se fait durant le lancement de l'application.

```
SrvEventBusiApplic.WFEventServiceCallback subscriber = new EventSubscriber();
eventclient.Subscribe(SrvEventBusiApplic.EventType.StateChange);
```

De la même manière pour se dé- enregistrer de ce type d'évènement.

```
eventclient.Unsubscribe(SrvEventBusiApplic.EventType.StateChange);
```

Configuration d'un service au moyen d'un fichier de configuration

Afin de configurer un service WCF à l'aide du fichier de configuration de l'application, il faut rajouter un nœud *system.serviceModel* et configurer les paramètres. Ci-dessous la configuration du service pour le service web de l'encaveur.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <services>
            <service name="EncavorWFBusiness.Service.WSEncavor"
                     behaviorConfiguration="myBehavior">
                <host>
                    <baseAddresses>
                        <add baseAddress="http://192.168.1.119:8080/EncavorService/"/>
                    </baseAddresses>
                </host>
                <endpoint address="" binding="wsHttpBinding"
                          contract="EncavorWFBusiness.Service.IWSEncavor"/>
                <endpoint address="http://192.168.1.119:8080/EncavorService/MEX"
                          binding="mexHttpBinding" contract="IMetadataExchange"/>
            </service>
            </services>
        <behaviors>
            <serviceBehaviors>
                <behavior name="myBehavior">
                    <serviceMetadata/>
                </behavior>
            </serviceBehaviors>
        </behaviors>
    </system.serviceModel>
</configuration>
```

Le nom du service correspond à la classe implémentant le service. L'attribut *behaviorConfiguration* définit le nom du nœud *behavior* se trouvant dans le nœud serviceB*ehaviors* du nœud *behaviors*. Ce nœud définit le comportement du service et les fonctionnalités de celui-ci. Ici, *serviceMetadata* permet aux *metadata* d'être découverts. Ensuite le nœud *host* permet de spécifier une adresse de base pour le service. Un service peut contenir plusieurs *endpoint* et ce sera cette adresse qui servira de base à tous les *endpoint.* Ensuite, on ajoute l'*endpoint* du service de l'encaveur. L'adresse n'est pas nécessaire du fait que l'on dispose de l'adresse de base du service, ensuite on spécifie le *binding* choisi et le *contract*. Le *contract* référence l'interface du service. Ici, un deuxième *endpoint* a été ajouté car il permet de publier les *metadata* sur un *endpoint* différent.

3.3.3 Couches d'accès aux données

La couche d'accès aux données des bases de données métiers des acteurs est faite grâce aux librairies nettiers [6]et à l'outil de génération Codesmith[7]. L'accès aux données pour la base de données de suivi et de persistance est fait grâce aux procédures stockées et aux classes développées à cet effet dans le Framework 3.0 dans le package WF.

Nettiers & Codesmith

Après la création des bases de données métiers de chaque acteur, un fichier de projet Codesmith a été implémenté afin de générer une solution Visual Studio par base de données. Ces solutions ont ensuite été incorporées à la solution racine du scénario. Chaque solution contient trois projets, ainsi que le fichier.csp du projet Codesmith. Les trois projets générés contiennent les classes d'accès aux données, ainsi que les classes représentant les entités des bases de données. Les paramètres de configuration de nettiers se trouvent dans le fichier de configuration de l'application hôte. Pour chaque solution générée, une entrée dans ce fichier de configuration est nécessaire.

Le fichier de projet Codesmith permet de générer ces solutions autant de fois que nécessaire. Ce qui est une fonctionnalité non négligeable lorsque l'on ne connaît pas toutes les informations qui doivent être contenues dans la base. Ainsi, toutes modifications de la base de données pendant l'implémentation de la solution, ajout de tables ou ajout de champs ne nécessitent aucun travail supplémentaire. Il suffit de générer la solution à l'aide du fichier Codesmith correspondant et les nouvelles entités sont immédiatement disponibles pour l'implémentation. Ce système permet une grande flexibilité et un gain de temps considérable durant l'implémentation.

3.3.4 Fichier de configuration

Les deux applications développées, l'application d'administration, ainsi que l'application business contiennent un fichier de configuration. Ce fichier permet de

[6] Nettiers is a set of open source code generation templates that simplify the tasks of creating customized Application Tiers for your Microsoft.Net applications. netTiers generated architecture is custom to your domain, uses familiar patterns, and follows the guidance of Microsoft's recommended patterns and practices. http://www.nettiers.com/

[7] CodeSmith is a software development tool to help you get your job done faster. Technically speaking it is a template driven code generator that automates the creation of common application code for any language (C#, Java, VB, PHP, ASP.NET, SQL, etc.).
CodeSmith includes many useful templates as well as entire sets of templates for generating proven architectures (.netTiers, CSLA, NHibernate, PLINQO, Wilson's ORMapper, APOSA, and more). You can easily modify any templates or write your own to generate your code exactly the way you want it. http://www.codesmithtools.com/

configurer, de paramétrer l'application avant son exécution sans devoir la recompiler, ce qui permet une plus grande flexibilité.

Application d'administration

Le premier paramétrage est celui du choix de l'acteur, des constantes, ainsi que des bases de données dont l'accès ne se fait pas par le module nettiers. Il contient un nœud *appSettings* à cet effet. Les constantes concernent les informations sauvegardées dans la base de données. Il est possible de configurer les clés primaires des trois types d'acteurs, des statuts des processus, ainsi que des constantes concernant le packaging. Cette implémentation améliore la flexibilité au cas ou ces clés primaires ne seraient pas les mêmes dans les bases de données de chaque acteur. Ce nœud contient également les informations nécessaires afin de se connecter aux bases de données de suivi de chaque acteur.

Le deuxième paramétrage concerne la configuration des services WCF. Il permet de modifier l'adresse permettant de contacter ce service. Il est également possible de modifier le *binding,* pour autant que l'implémentation du service l'accepte. Etant donné que les services s'exécutent dans l'application et non en tant que services Windows ou encore dans IIS, et que la présentation de la solution doit pouvoir se faire sur une seule machine, il est nécessaire pour chaque service de spécifier un port différent. Pour cette raison, il faut que le Firewall de la machine permette les communications sur ces ports spécifiés.

Le dernier paramétrage concerne la configuration des bases de données utilisant le module d'accès aux données basé sur nettiers. Il permet de spécifier la configuration nettiers nécessaire à son exécution. Le *connectionString,* des bases de données dont l'accès se fait avec le module nettiers est inclus dans le nœud *connectionString*. Il faut une configuration nettiers pour chaque acteur du scénario car chaque acteur utilise des objets différents représentant les entités de leur base de données.

Application business

Le premier paramétrage permet également de spécifier le rôle de l'acteur de l'application, ainsi que les constantes permettant de déterminer les clés primaires des différents statuts des processus.

Le deuxième paramétrage référence le paramétrage des services WCF utilisés par l'application. Lors de la création des « services references » dans visual studio, cette section est automatiquement créée. Elle contient des informations sur le paramétrage des services.

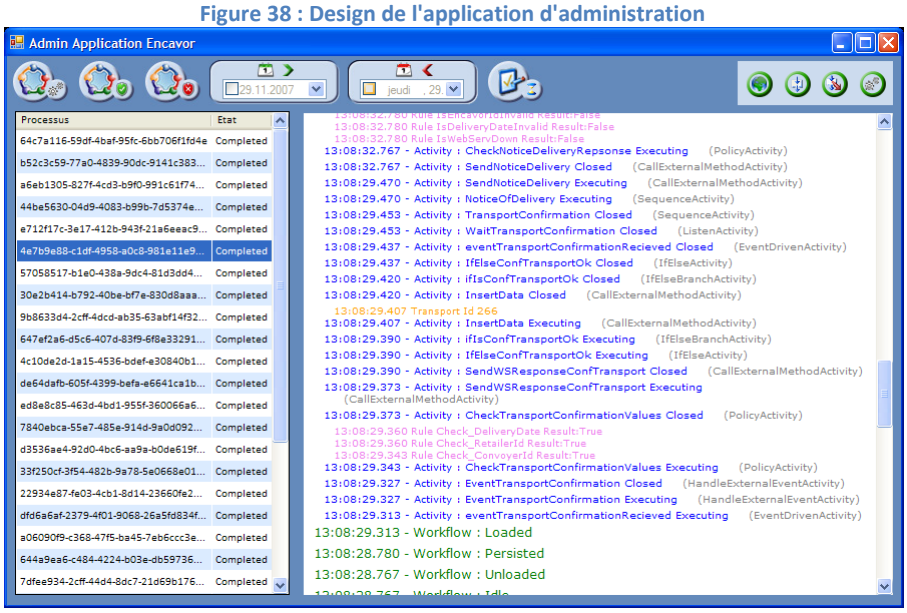
3.3.5 Formulaire des applications

Les deux applications, l'application d'administration et l'application business sont implémentées de manière similaire. Elles disposent toutes deux d'une barre de

boutons au sommet, permettant l'affichage des informations et l'activation des fonctionnalités. La partie gauche contient le listing des processus et la partie droite les informations du processus sélectionné. Une attention particulière a été portée au design des applications afin que cela soit le plus intuitif et le plus agréable possible.

Application d'administration

Figure 38 : Design de l'application d'administration



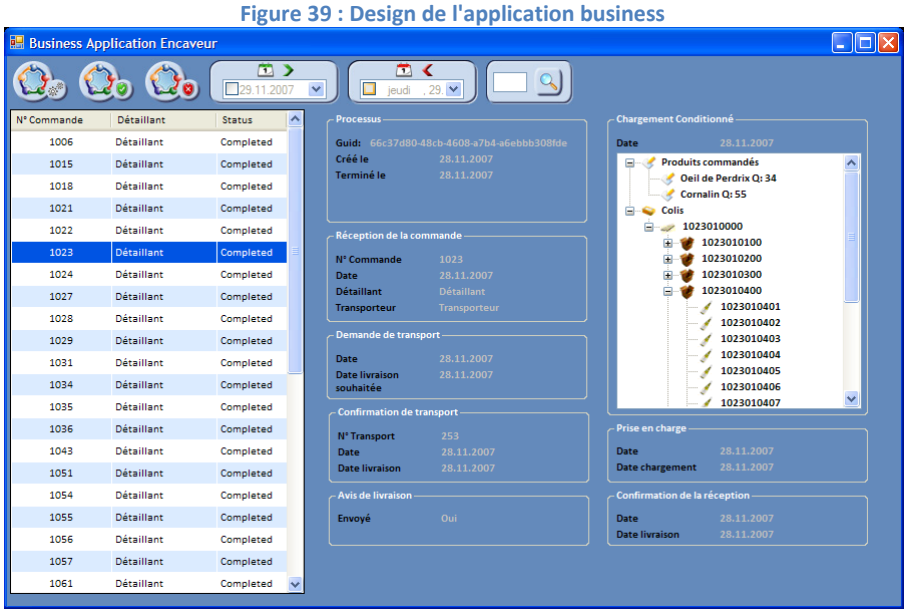
Le *DataSource* du *DatagridView* contient un type d'objet personnalisé. Il contient les propriétés affichées en tant que colonnes du *DatagridView* et la variable permettant de stocker les données du service de suivi personnalisé. Ce sont ces données de suivi qui sont ensuite affichées dans la partie droite de l'application. L'application utilise deux *BindingSource* différentes, une pour l'affichage des processus en cours et l'autre pour l'affichage des processus terminés.

L'objet utilisé pour l'affichage des informations de processus est de type *Webbrowser*. L'utilisation de ce conteneur permet le formatage des données. Un objet de type *RichTextBox* n'a pas été utilisé car il ne permettait pas l'affichage de couleurs différentes, suivant le type de données transmises, car incompatible avec la manière dont le service de suivi fournit les données.

La barre de menu contient, à l'extrémité droite, quatre icones permettant l'affichage du statut des services. Cette fonctionnalité est intéressante, elle permet de visualiser rapidement si les services ont pu s'exécuter normalement au démarrage de l'application.

Application business

Figure 39 : Design de l'application business



Le *DataSource* du *DatagridView* contient également un type d'objet particulier. Celui-ci comprend les propriétés affichées ainsi qu'une variable référençant les données métiers du processus. Ces données sont ensuite affichées dans la partie gauche de l'application pour le processus sélectionné.

L'affichage des informations de processus est fait à l'aide de *UserControl,* un *UserControl* par acteur à été développé. Au démarrage de l'application, suivant le rôle attribué à l'application, le bon *UserControl* est instancié.

4 Application de démonstration

L'objectif principal de l'implémentation de cette solution est de montrer les possibilités de développement à l'aide de la technologie WF. Les applications ainsi développées sont des prototypes et ne peuvent être utilisées en tant qu'applications productives.

Les services ne peuvent soutenir la charge d'une application mise en production, car ils sont directement embarqués au sein de la solution et n'implémentent pas les fonctionnalités de base fournies, par exemple par IIS ou les services WAS, permettant le recouvrement après erreur ou encore le maintien du service sur la durée.

L'affichage du suivi en temps réel des processus, ainsi que du suivi des processus terminé dans l'application d'administration peut être amélioré afin d'afficher uniquement les informations souhaitées. L'affichage proposé dans cette solution a été choisi afin de montrer toutes les informations pouvant être ressorties des services de suivi.

Des tests poussés ainsi qu'une montée en charge n'ont pas été effectués, il n'est dès lors pas possible de déceler toutes les erreurs pouvant survenir au cours de tel test.

5 Limitations et problèmes rencontrés

Ce point référence les problèmes rencontrés durant l'implémentation de la solution ainsi que les limitations de la technologie

5.1 Intercommunication service web – instance de processus

Le service web de communication entre les acteurs devant être utilisé avec des technologies différentes, il doit respecter les standards. De ce fait, il n'est pas possible d'utiliser le service en mode callback. Le client appel et une réponse doit être fournie directement. Cette méthode complique l'interaction avec les processus. Lorsqu'un appel au service est effectué, les données sont transmises au service local inscrit dans le moteur d'exécution des processus, le service local transmet les données au processus et les traite suivant la logique implémentée pour finalement renvoyer au service local une réponse qui sera retournée au client par le service web. Les complications surviennent lorsque le service local transmet les données à l'instance de processus correspondante. La méthode appelée par le service web sur le service local attend un retour, or, ce retour ne peut être transmis directement car les données sont traitées par la bonne instance de processus au sein de son *Thread* d'exécution. Lorsque l'instance de processus renvoie les informations au service local, il n'est plus dans la méthode attendant le retour. Le problème a été résolu en mettant en attente le retour de la méthode du service web. L'attente se termine lorsque l'instance de processus envoie les données de retour au service local.

5.2 Designer des processus dans Visual Studio

Une lenteur et un blocage de Visual Studio survient lors de l'utilisation du designer des processus pour un fichier contenant un nombre important d'activités. La modélisation des processus est faite en mode « code-separated » de ce fait à chaque ouverture et modification du fichier des processus Visual Studio doit reconstruire le diagramme à

l'aide de ces fichiers XML. Afin de développer le processus il est nécessaire d'utiliser à la fois le fichier contenant les propriétés et méthodes du processus et le fichier utilisant le designer pour les affecter aux activités. Il est donc nécessaire de passer de l'un à l'autre de ces fichiers plusieurs fois.

Lors de la première ouverture du fichier utilisant le designer lorsque Visual Studio est lancé, Visual Studio se fige pendant environ 45 secondes avant d'afficher le processus.

Lorsque l'on navigue dans le designer à l'aide de la « scrollbar », Visual Studio se fige aléatoirement pendant environ 5 secondes avant d'afficher les activités.

Lorsqu'une boîte de dialogue doit être ouverte pour affecter une propriété il peut se passer également jusqu'à 5 secondes avant que la boîte de dialogue s'ouvre.

Ce problème est extrêmement gênant et fait perdre un temps considérable. De ce fait, l'implémentation dans le designer est plus une contrainte qu'une aide au développement. L'implémentation directe des activités et de l'affectation des propriétés sans le designer dans le script XML est beaucoup trop fastidieuse pour contourner ce problème et peut compromettre le fichier.

5.3 Versioning et gestion des build

Le core service de persistance et de suivi des processus peut vite devenir un cauchemar. Les données insérées dans les bases de données pour la persistance et le suivi sont sérialisées. La sérialisation ne persiste ou sauve pas seulement les champs des processus, mais le processus dans son ensemble.

Lorsqu'on effectue une modification d'un processus, par l'ajout ou la suppression d'activités par exemple on modifie le « build » du processus. Ces modifications peuvent être faites, soit par une mise à jour dynamique d'une instance de processus en cours d'exécution, soit par le développeur devant modifier le processus.

Le problème survient avec les données persistées dans la base, les données sont en version 1.0 alors que le processus est actuellement en version 2.0. La déshydratation du processus ne marchera pas, et génèrera une erreur car la base contient le « template » complet du processus. Le processus de déshydratation essaie de « mapper » le processus persisté et échoue car l'arbre des activités ne correspond pas. Afin de résoudre ce problème, deux solutions sont possibles. La première consiste à exécuter les deux « assembly » côte à côte. La deuxième consiste à utiliser les changements afin de passer le processus persisté en version 2.0. Ces solutions ne sont pas expliquées plus en détail. Un article de Ruurd Boeke sur son blog présente cette problématique à l'adresse ([RUUBOE] Ruurd Boeke).

5.4 Maintenance de la BD du « core » service de suivi

Un bug subsiste dans le Framework lors de l'utilisation du partitionnement automatique de la base de données du « core » service de suivi. Si la propriété permettant le partitionnement automatique est définie à *true (PartitionOnCompletion)*, alors tous les évènements utilisateurs dans le service de suivi ne sont pas ressortis. Ce problème est connu de Microsoft et en cours de traitement.

6 Gestion du versioning et des sources

Le projet s'étalant sur douze semaines, il est nécessaire d'établir une certaine rigueur dans la gestion des sources et du versioning des documents pour éviter la perte de fichier, ce qui pourrait être désastreux pour la réussite du projet et pour les délais à respecter. Le versioning permet également d'ordonner ces fichiers et avoir une vision sur le développement du projet. Ce projet étant un travail individuel, il ne nécessite pas une structure lourde pour la mise en place d'un système de projet collaboratif permettant à un team de travailler simultanément sur la même solution et gérant également le versioning et la sauvegarde des données. Concernant la gestion des sources, un simple stockage des données de sauvegarde sur différents supports permet d'éviter toutes pertes. Concernant la gestion du versioning, chaque modification importante sur un fichier engendre une nouvelle version.

La gestion des sources et du versioning est gérée de manière simple mais efficace comme exposé préalablement. Ci-dessous, voici les détails de cette gestion.

6.1 Gestion des sources

Le projet étant important et planifié sur douze semaines, il ne peut être acceptable de perdre des données liées à un crash matériel. Les données n'étant utilisées que par un seul intervenant, une sauvegarde de l'ensemble de l'existant est enregistrée à chaque période de temps déterminée sur trois différents supports dont deux sont des supports de données externes. Cette sauvegarde est faite quotidiennement en fin de journée pendant toute la durée du projet.

Les fichiers concernant la documentation du travail sont placés dans un dossier séparé de l'implémentation de la solution. Ce dossier est également subdivisé en dossier représentant les différents délivrables requis.

6.2 Gestion du versioning

Le versioning des fichiers et de la solution est géré de la manière suivante. Pour les documents devant être rédigés hebdomadairement, un nouveau fichier comportant le numéro de la semaine de travail est créé. Pour les documents plus importants, comme par exemple le fichier du document final ou encore la solution Visual Studio, chaque modification importante fait office d'une nouvelle version.

IX. SYNTHESE

Le travail effectué avait comme objectif la découverte des possibilités et des fonctionnalités de Windows Workflow Foundation, ainsi que l'implémentation d'une application prototype du scénario. Comme présenté tout au long de ce rapport, il est possible d'affirmer que les deux objectifs ont été atteints. En effet, le point Exploration technologique, page 22, présente la technologie dans son ensemble, des concepts inhérents aux différentes fonctionnalités proposées. La réussite de l'implémentation de l'application prototype confirme également le succès de la réalisation du deuxième objectif.

Le document exploratoire est un condensé de l'ensemble de la technologie ce qui en fait sa principale valeur. Il permet de prendre connaissance de l'étendue de ladite technologie, depuis ses fondements et ses concepts, jusqu'aux possibilités d'implémentation. Il permet en un minimum de temps d'en avoir une idée générale.

Le premier point positif de l'application prototype est la réussite de l'implémentation complète du scénario imposé en respectant les interfaces de communication. Le deuxième, comme le montre le tableau ci-dessous, est l'incorporation totale des fonctionnalités « Must Have », ainsi que de la majorité des fonctionnalités « Nice to Have » du cahier des charges. Finalement l'utilisation des services WCF, pour la communication entre les deux applications au sein d'un acteur permettant à plusieurs applications business de s'exécuter en même temps sur des postes différents, est un atout supplémentaire à l'application.

Fonctionnalités	Must Have	Nice to Have	Développé
Gestion des stocks et des commandes avec historique	✗		✓
Monitoring des processus	✗		✓
Interface de communication entre les différents maillons	✗		✓
Processus métier automatisés au sein de chaque maillon et entre les différents maillons.	✗		✓
Interface de gestion, d'affichage des processus en cours.	✗		✓
Interface de commande de vins pour le détaillant	✗		✓
Interface globale du scenario		✗	
Choix de la reprise des processus		✗	✓
Service de log, archivage des processus		✗	✓
Service de mail au sein de chaque maillon et entre les différents maillons		✗	
Commande du détaillant en fonction du stock de l'encaveur		✗	✓
Sécurisation des web services		✗	
Interface d'ajout de produits		✗	

Le point négatif du travail concerne la perte de temps considérable lors de la modélisation des processus dans le designer, empêchant le développement de fonctionnalités supplémentaires et l'amélioration des données présentées dans l'application prototype.

X. CONCLUSION

Le Framework 3.0 apporte un outil de développement orienté Business Process Management avec le package Windows Workflow Foundation. Cet outil n'offre pas une solution prête à l'emploi comme proposé par les technologies concurrentes actuellement sur le marché, mais fournit en revanche tous les outils permettant le développement d'applications orientées processus. Avec les composants fournis, il est possible de développer une architecture pouvant comprendre l'ensemble des fonctionnalités des BPMS actuels. De ce fait, cette proposition permet un système pouvant s'ajuster aux spécificités de chaque entreprise.

Toutefois, alors que la tendance actuelle des BPMS se dirige vers une normalisation de la modélisation, ainsi que de proposer des outils permettant un travail collaboratif entre les analystes et les développeurs, Windows Workflow Foundation quant à lui n'aspire pas complètement à cette tendance. En effet, les outils proposés sont destinés principalement aux développeurs, étant donné que le designer de Visual Studio doit conjointement être utilisé avec les propriétés et les méthodes développées.

Finalement, il est très intéressant de se tourner vers cette technologie, plutôt que le développement d'application DotNet standard, lorsque la solution doit s'exécuter sur une longue période et nécessite des temps d'attente entre les différentes activités. En revanche, si l'on doit disposer de toutes les fonctionnalités présentent dans la plupart des BPMS actuels, il est préférable de se tourner vers une technologie concurrente. WF permet une implémentation orientée processus permettant de satisfaire les besoins spécifiques des entreprises, moyennant une charge de développement conséquente.

XI. BIBLIOGRAPHIE

[BEABPM] WebSite. AquaLogic BPM. *BEA.* [En ligne] [Citation : 1 11 2007.] http://www.bea.com/framework.jsp?CNT=overview.htm&FP=/content/products/aqualogic/albpm/.

[BPMS01] WebSite. BPMS- Business Process Management System. *Journal du Net.* [En ligne] [Citation : 1 10 2007.] http://www.journaldunet.com/encyclopedie/definition/325/51/20/business_process_management_system.shtml.

[BPMS02] WebSite. Business Process Management. *Wikipedia.* [En ligne] [Citation : 1 10 2007.] http://en.wikipedia.org/wiki/Business_process_management.

[BukBru07] Bukovics, Bruce. 2007. *Pro WF Windows Workflow in DotNet 3.0.* s.l. : Apress, 2007.

[FrmDotNet] WebSite. Centre de développement DotNet Framework. *Microsoft Corporation.* [En ligne] [Citation : 01 10 2007.] http://msdn2.microsoft.com/fr-fr/netframework/default.aspx.

[GesProMe] WebSite. La gestion des processus métiers. *La gestion des processus métiers.* [En ligne] [Citation : 1 10 2007.] http://www.processus-metier.fr/2007/03/18/mais-le-bpm-cest-quoi/.

[IBMWES] WebSite. IBM WebSphere - Logiciel - France. *IBM.* [En ligne] [Citation : 1 11 2007.] http://www-306.ibm.com/software/fr/soa/launch/bpmsoa.html?ca=hpwebsphere.

[INTBPM] WebSite. Intalio Leader in Open Source BPMS. *Intalio.* [En ligne] [Citation : 1 11 2007.] http://www.intalio.com/products/components/.

[MSDN01] WebSite. Accueil du site MSDN. *MSDN.* [En ligne] [Citation : 01 10 2007.] http://msdn2.microsoft.com/fr-fr/default.aspx.

[NetFxC] WebSite. Microsoft DotNet Framework 3.0 Community (Netfx3). *Microsoft DotNet Framework.* [En ligne] [Citation : 01 10 2007.] http://www.netfx3.com/.

[RUUBOE] Ruurd Boeke. Ruurd Boeke Entreprise Development and technobabble. *Sitechno.* [En ligne] [Citation : 01 11 2007.] http://www.sitechno.com/Blog/CategoryView,category,WF+(Workflow).aspx.

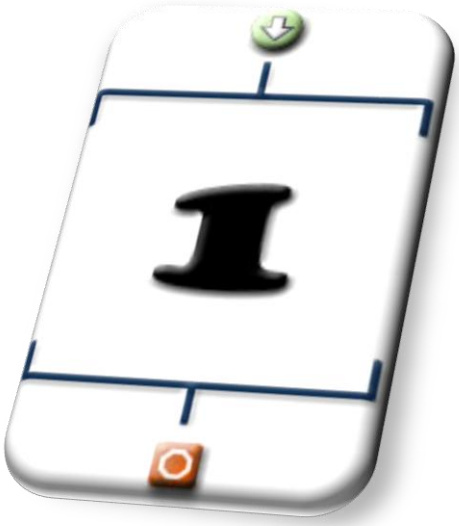
[WCFMS] WebSite. Windows Communication Foundation (WCF). *Microsoft DotNet Framework.* [En ligne] [Citation : 01 10 2007.] http://wcf.netfx3.com/.

[WWFMS] WebSite. Windows Workflow Foundation. *WinFX Windows Workflow Foundation.* [En ligne] [Citation : 01 10 2007.] http://www.workflow-foundation.com/default.aspx.

[XPEIVY] WebSite. Consilium IT - Conseil en Technologies de l'Information. *Consilium IT.* [En ligne] [Citation : 1 11 2007.] http://www.consilium-it.ch/xpert-ivy-fr36.html.

XII. TABLE DES ILLUSTRATIONS

XIII. APPENDICES



Cahier des charges

Description :

Le cahier des charges établi en début de projet définit le cadre du travail à réalise ; il définit la technologie, le scénario, les fonctionnalités, l'architecture, ainsi que les délivrables du travail.

1 Présentation

Dans le cadre de mon travail de diplôme pour l'institut d'informatique de gestion de la HES-SO // Valais, j'ai comme objectif, d'intégrer « Microsoft Windows Workflow Foundation » dans une chaîne logistique.

Le scénario de cette chaîne logistique a été défini par l'institut d'informatique de gestion. Mon travail consiste à le concevoir et le gérer. On distingue 3 acteurs pour cette chaîne, un encaveur, un transporteur et un détaillant. L'encaveur doit livrer le vin au détaillant en faisant appel au transporteur.

L'objectif est d'implémenter une interface pour chaque maillon de cette chaîne logistique et de les faire communiquer entre eux de la manière la plus automatisée possible. Le moyen de communication entre les différents acteurs est imposé par le scénario proposé et doit se faire à l'aide de Web Services.

2 Technologie

La technologie exigée pour le développement de cette chaîne logistique est « Windows Workflow Foundation ». Cette technologie est une des composantes du Framework de développement DotNet 3.0 et permet la conception d'applications orientées processus.

2.1 Besoins technologiques

La technologie « Windows Workflow Foundation » restreint le choix des technologies à employer pour le développement de ce scénario par rapport aux différentes solutions existantes sur le marché. L'implémentation des processus métiers est faite grâce à la plateforme de développement DotNet proposée par Microsoft. L'implémentation est donc faite en C#.

Les informations permettant le fonctionnement des processus métiers, ainsi que la gestion du cycle de vie des processus de chaque maillon de la chaîne sont stockées dans une base de données SQL Server 2005. Le choix de ce SGBDR est justifié du fait d'une meilleure intercommunication avec les couches « Data Tier » utilisées dans ce développement.

L'accès aux données de la base par les couches « Business Process Tier » est fait grâce au Template Nettiers qui est basé sur les « Microsoft Entreprise Library Application

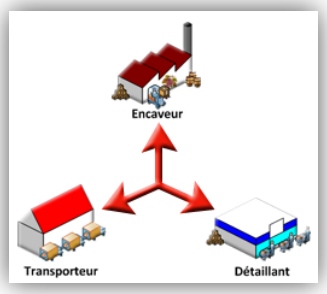
Blocks ». Ce Template construit des objets relationnels pour la base de données reposant sur le Model Driven Design (MDD).

L'intercommunication entre les différentes interfaces de chaque acteur de la chaîne est réalisée par des Web Services DotNet. Cette technologie est choisie afin d'avoir un environnement de développement homogène.

Récapitulatif des technologies :

- Framework .Net 3.0
- Langage de développement : C #
- SQL Server 2005
- Service IIS
- Web Services .Net
- Nettiers (Accès aux données)

Scénario de la chaîne logistique

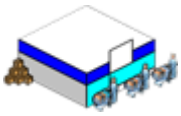


Le scénario de la chaîne logistique est composé de trois acteurs : un encaveur, un transporteur et un détaillant. L'encaveur doit livrer le vin au détaillant, faisant appel au transporteur. Le scénario a été simplifié par rapport à la réalité afin de limiter la complexité des interfaces et des intercommunications entre celles-ci et permettant ainsi de terminer l'application dans les temps impartis. Cette simplification est délimitée dans les deux documents annexes suivants : TD BPMS – Définition du scénario et TD BMPS – Définition des interfaces techniques.

2.2 Les acteurs

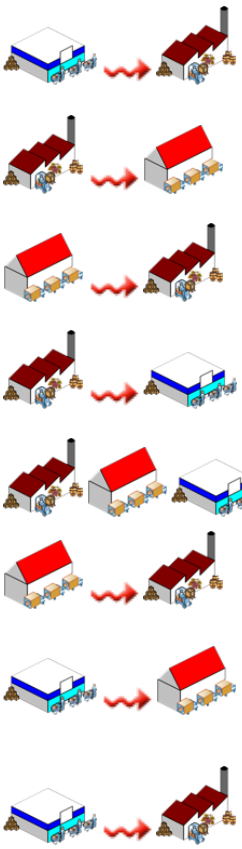
Encaveur : l'encaveur est responsable de la cave, il vinifie le vin, le conditionne et le livre aux détaillants.

Transporteur : le transporteur assure le transport du vin de l'encaveur au détaillant. Il possède des clients dans tous les domaines de marchandises.



Détaillant : le détaillant exploite une surface de vente au détail. Il réapprovisionne son assortiment auprès de grossistes et de producteurs, parmi lesquels, pour le vin, les encaveurs.

2.3 Déroulement du scénario



L'élément déclencheur du scénario est le détaillant, en effet, lorsque celui-ci atteint son seuil critique pour les vins, il va passer une commande à l'encaveur.

L'encaveur réceptionne la commande et dans le but d'acheminer les produits au client, fait une demande au transporteur.

Le transporteur réceptionne cette demande et informe l'encaveur qu'il est prêt à exécuter la demande, il envoie une confirmation à l'encaveur.

L'encaveur envoie un avis de livraison au détaillant afin de l'informer qu'il va lui livrer les produits correspondants.

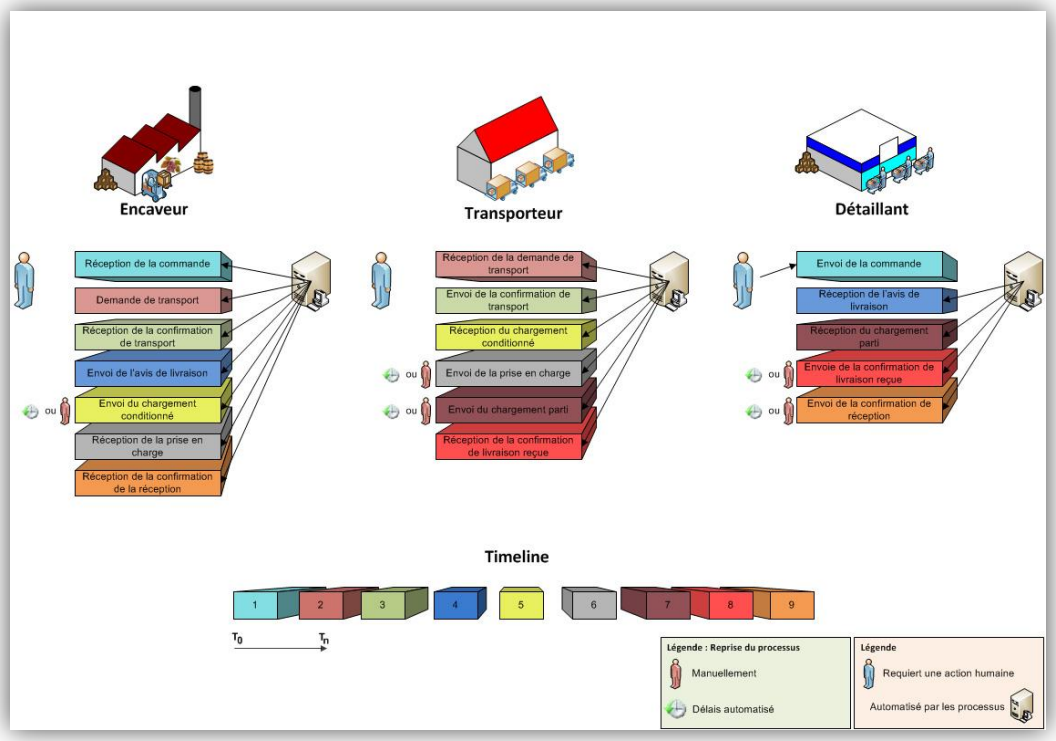
Le scénario continue avec le transporteur qui va charger la marchandise chez l'encaveur afin de la livrer chez le détaillant.

Après le chargement, le transporteur envoie à l'encaveur une confirmation de prise en charge de la marchandise.

Lorsque le transporteur a livré les marchandises chez le détaillant, ce dernier, lui envoie une confirmation de livraison pour confirmer la réception des marchandises livrées.

Finalement, le détaillant envoie un accusé de livraison à l'encaveur afin de lui confirmer la réception des marchandises commandées.

Use Case



Le schéma ci-dessus liste les activités qui sont accomplies par chaque acteur tout au long du scénario. La time line nous indique la succession ordonnée des activités à l'aide des blocs de couleurs numérotés. Chaque couleur correspond à une interaction entre deux maillons de la chaîne. La légende nous permet de distinguer si une activité donnée est traitée par une action humaine, c'est-à-dire une action manuelle, ou alors grâce à un processus automatisé.

L'objectif de ce scénario consiste à avoir des actions humaines en nombre le plus restreint afin d'avoir une automatisation optimale.

Au sein des entreprises, l'enchaînement des activités tout au long d'un scénario requiert parfois un certain temps. En effet, il peut se passer plusieurs heures voir plusieurs jours avant que l'activité suivante ne commence. L'enchaînement de l'activité « Envoi du chargement conditionné » après l'activité « envoi de l'avis de livraison » peut nécessiter une attente d'un jour. Il est possible de simuler cette attente de deux manières différentes, la première consiste à implémenter un délai entre les tâches à l'intérieur du processus métier, et la deuxième consiste à activer l'activité suivante par une action humaine. Dans ce scénario ces attentes sont simulées par un délai paramétrable à l'intérieur des processus métiers.

Fonctionnalités

Voici la liste des fonctionnalités qui sont implémentées pour ce scénario. Ces fonctionnalités sont développées pour chaque maillon de la chaîne si rien n'est précisé.

2.4 Must have

2.4.1 Gestion des stocks et des commandes avec historique.

Les différentes interfaces doivent permettre de gérer des commandes de produits ou des demandes de transport, en fonction des produits ou du matériel référencés, dans une base de données et garder une trace de celles-ci. Les interfaces d'ajout, de suppression ou de gestion de données des bases ne sont pas prises en compte dans cette fonctionnalité.

2.4.2 Monitoring des processus.

Les interfaces doivent permettre de connaître le statut des processus en cours. De connaître en tout temps le nombre de processus en cours, ainsi que l'état de chaque processus.

2.4.3 Interface de communication entre les différents maillons.

Les interfaces de chaque acteur de la chaîne doivent pouvoir communiquer entre elles au moyen de Web Services.

2.4.4 Processus métiers automatisés au sein de chaque maillon et entre les différents maillons.

Les interactions de l'être humain pour le lancement des activités au sein des processus doivent être les plus restreintes possible, afin d'avoir un scénario le plus automatisé qu'il soit.

2.4.5 Interface de gestion, d'affichage des processus en cours.

Les interfaces doivent disposer d'un formulaire d'affichage permettant de visualiser des informations concernant les processus actifs, par exemple le détail de la commande.

2.4.6 Interface de commande de vins pour le détaillant.

L'interface du détaillant de vin doit comprendre un formulaire pour effectuer une nouvelle commande auprès de l'encaveur.

2.5 Nice to have

2.5.1 Interface globale du scénario

Cette interface doit permettre d'afficher l'état actuel d'un scénario en cours et de mettre à jour en temps réel l'activité courante de celui-ci. Elle doit être utilisée avec tous les acteurs quelque soit la technologie employée.

2.5.2 Choix de la reprise des processus

L'implémentation doit permettre de choisir comment on désire reprendre les processus mis en attente. Soit automatiquement à l'aide de paramètres déterminés à l'avance, ou alors manuellement, par intervention humaine.

2.5.3 Service de log, archivage des processus.

Un service d'archivage permet d'avoir un historique de l'exécution des processus terminés permettant ensuite, par exemple, l'analyse de ceux-ci afin d'améliorer les performances, ou de revoir la modélisation du processus de l'entreprise.

2.5.4 Service de mail au sein de chaque maillon et entre les différents maillons.

Un service de mail permet d'informer l'acteur sur les processus en cours, sans qu'il ait besoin de consulter l'interface de gestion des processus.

2.5.5 Commande du détaillant en fonction du stock de l'encaveur

L'interface de commande doit permettre d'effectuer une commande de marchandises en visualisant les produits disponibles chez l'encaveur.

2.5.6 Sécurisation des web services

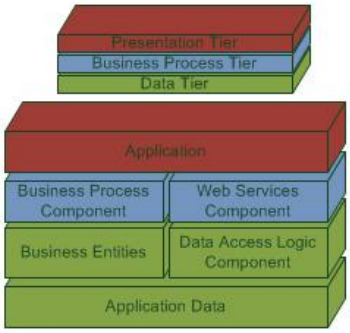
Cette fonctionnalité permet aux seuls utilisateurs de cette chaîne logistique d'utiliser ces services, et le transfert de données est crypté.

2.5.7 Interface d'ajout de produits.

Cette interface permet de diversifier la palette de produit de l'encaveur et du détaillant.

Architecture

2.6 Logiciel



Source : Basé sur « MS Patterns & Practices »

L'architecture des interfaces composants ce scénario est inspirée de « Microsoft patterns & practices » qui consiste à séparer le développement en 3 couches, la couche « Data Tier », la couche « BusinessProcess Tier » et la couche « Presentation Tier ». Pour chaque acteur de la chaîne une interface sera implémentée selon ce concept.

2.6.1 Data Tier

La partie « Application Data » comprend les données permettant l'exercice de l'activité de l'acteur, ainsi que les données relatives à la gestion du cycle de vie des processus. Les parties « Business Entities » et « Data Access Logic Component » sont dédiées au Template Nettiers. La couche « Business Entities » contient toutes les entités correspondant aux tables de la base de données. Finalement la partie « Data Access Logic Component » fait la jonction entre « Application Data » et « Business Entities ». Elle permet d'aller interroger la base de données afin d'attribuer des valeurs correspondantes aux entités qui seront crées et utilisées par la couche « Business Process Tier ».

2.6.2 Business Process Tier

La partie « Business Process Component » comprend toutes la logique, les règles de gestion de l'interface et le processus métier permettant le déroulement séquentiel de l'activité de l'acteur. La partie « Web Services Component » permet quant à elle l'intercommunication de l'interface avec les autres interfaces des acteurs de la chaîne.

2.6.3 Presentation Tier

La partie « Application » permet l'affichage des données à l'utilisateur et permet également l'interaction de celui-ci avec les données.

2.7 Matériel

Afin de réaliser ce scénario le matériel suivant est nécessaire. Une machine Windows Xp ou 2003 Serveur contenant les applications et services suivants : le serveur SQL 2005 permettant d'héberger la ou les bases de données, le Framework .Net 3.0 permettant l'exécution de l'interface et le service IIS pour la publication des Web Services. Cette machine fera office de serveur. Un autre poste de travail hébergera l'application de gestion des processus. Une telle configuration est nécessaire pour chaque acteur de la chaîne logistique.

Les délivrables du projet

Le projet est rendu en deux parties. La première partie est le document final et la deuxième comprend la solution développée. Ci-dessous, voici la liste exhaustive de chaque partie :

2.7.1 Document final :

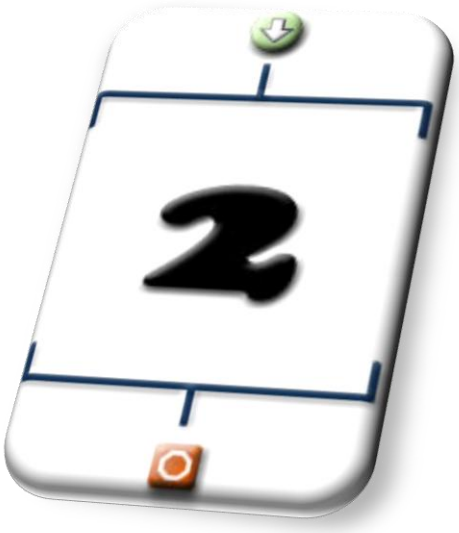
- Présentation du projet
- Présentation de la technologie
- Compte rendu de la phase d'analyse de la technologie
- Compte rendu de la phase de développement de la solution
    - Diagramme de Use Case
    - Diagramme de séquence
    - Modélisation des différentes bases
    - Modélisation des différents processus
    - Architecture de la solution
    - Manuel d'utilisation
    - Manuel de déploiement
- Bibliographie
- Compte rendu de la gestion des sources
- Compte rendu du versioning
- Bilan du projet réalisé
- Présentation PowerPoint (rendue après la défense orale)
- Annexe :
    - Cahier des charges
    - Documentation utilisée durant le projet
    - Délivrables reçus et réalisés durant le projet

2.7.2 Solution

- Solution Visual Studio 2005 du projet
- Scripts de création des bases de données et des utilisateurs
- Machine virtuelle permettant la présentation du scénario de la chaîne logistique.

Délais

Le projet commence le 17 septembre 2007 et se termine le 10 décembre 2007. Les délivrables du projet sont remis le 10 décembre 2007 avant midi.



Définition du scénario

Description :

Document reçu en début de projet définissant le scénario de la chaîne logistique

Auteur Nicole Glassey, Laurent Bagnoud
Date 12.3.2007
Thème Scénario

Destinataire Equipe de projet
Pour info

BPMS-4-SME – Définition du scénario

Doc-ID	BPMS-4-SME.012
Version	0.3
Statut	En révision
Sauvegarde sur	Q:\I\BaA\Ra&D\RCSO\BPMS_4_SMEs\ProjectDoc\08_Requirements
Archivage	5 ans

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition du scénario

Modifications

Version	Date	Exécutant	Remarques sur la modification
0.1	15.1.2007	Nicole Glassey	Première version
0.2	12.3.2007	Laurent Bagnoud	Divers modifications
0.3	27.3.2007	Laurent Bagnoud	Divers modifications

Contrôles - Reviews

Version	Date du contrôle	Exécutant(s)	Remarques

Libérations

Version	Date de libération	Personne en charge	Remarques

Table des matières

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition du scénario

1. Introduction

L'objectif du projet BPME-4-SME consiste, entre autres, à définir une architecture BPMS adaptée aux PME. Dans le but de fournir une illustration susceptible d'éveiller l'intérêt du plus grand nombre, l'industrie viticole a été choisie comme contexte pour ce projet.

Dans le cadre du projet BPME-4-SME le présent document constitue la première étape de l'analyse des processus. Son objectif est de définir avec précision les différents acteurs de la chaîne logistique, leurs rôles respectifs ainsi que le détail des opérations qu'ils exécutent dans le cadre de leurs activités.

L'analyse est caractérisée par une approche top-down. Les acteurs de la chaîne logistique seront tout d'abord présentés d'un point de vue global (chapitre 2), de même que la chaîne logistique elle-même (chapitre 3). Les processus propres à chaque acteur seront ensuite identifiés (chapitre **Erreur ! Source du renvoi introuvable.**).

L'analyse passera ensuite à une phase plus approfondie, détaillant tout d'abord les flux physiques et d'information échangés par les acteurs (chapitre 5) avant de terminer par un examen détaillé de chacun des processus de chaque acteur (chapitre 6).

1.1 Domaine de validité

Ce document est valable pour le projet RCSO BPMS-4-SME.

1.2 Public cible et conditions préalables à la lecture

Ce document est destiné à toutes les personnes concernées par le projet BPMS-4-SME.

1.3 Termes et abbréviations

Terme / Abbréviation	Définition / Description
BPMS	Business Process Management System
BPM	Business Process Management

Tableau 1 Termes et abbréviations

1.4 Documents référencés

Nr.	Doc-ID	Document	Auteur	Version
[1]	-		Laurent Bagnoud	1.0
[2]				
[3]				

Tableau 2 Documents référencés

Hes·SO
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

2. Acteurs

La chaîne logistique est composée de divers *acteurs*. Ceux-ci représentent en quelques sortes les maillons de la chaîne logistique, et ce sont leurs actions et leurs interactions qui permettent à la chaîne d'atteindre son objectif: l'encaveur doit livrer le vin au détaillant, faisant appel au transporteur.

Les acteurs modélisés dans le cadre de la démonstration sont présentés dans le tableau suivant:

Encaveur	**1 – Encaveur** L'encaveur est responsable de la cave. Il vinifie le vin, le conditionne (mise en bouteille) et le livre aux détaillants. → L'encaveur sera implémenté et maintenu par la HES-SO Valais.
Transporteur	**2 - Transporteur** Le transporteur assure le transport du vin de l'encaveur au détaillant. Il possède des clients dans tous les domaines de marchandises. → Le transporteur sera implémenté et maintenu par HE-ARC
Détaillant	**3 – Détaillant** Le détaillant exploite une surface de vente au détail. Il réapprovisionne son assortiment auprès de grossistes et de producteurs, parmi lesquels, pour le vin, les encaveurs. → Le détaillant sera implémenté et maintenu par la HES-SO Valais

Tableau 3 : Acteurs

Laurent Bagnoud
laurent.bagnoud@hevs.ch
☎ +41 27 606 90 13
🖷 +41 27 606 90 00

Institut Informatique de Gestion
HES-SO Valais
TechnoArk 3
3960 Sierre

Hes·SO
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition du scénario

Laurent Bagnoud
laurent.bagnoud@hevs.ch
☎ +41 27 606 90 13
🖷 +41 27 606 90 00

Institut Informatique de Gestion
HES-SO Valais
TechnoArk 3
3960 Sierre

3. Chaîne logistique: vue générale

Le fonctionnement d'une chaîne logistique peut être assimilé à une chorégraphie complexe exécutée par les acteurs qui la composent. Le diagramme suivant présente une vue générale de cette chorégraphie et met en évidence les *interactions* entre acteurs. L'encaveur fait appel au transporteur pour livrer du vin au détaillant.

Ces interactions feront l'objet d'une description détaillée au chapitre 4 à la page 8.

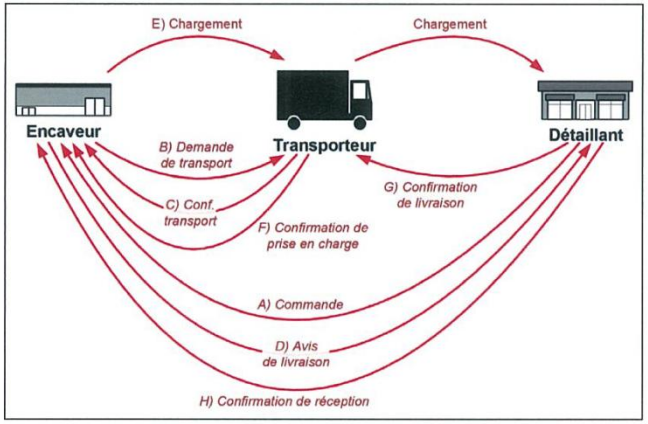


Figure 1 Scénario - Vision d'ensemble

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition du scénario

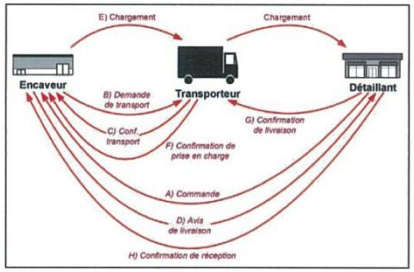
4. Interfaces entre acteurs

Les interactions entre acteurs représentent l'un des aspects clef de la modélisation de la chaîne logistique. Ce chapitre présente les différentes *interfaces* entre paires d'acteurs et définit précisément les *flux* (flux physiques et flux d'informations) qu'ils échangent.

La représentation de la chaîne logistique est une modélisation très simplifiée de la réalité, permettant une réduction significative de la complexité de l'interface entre acteurs (de même que de leurs processus internes). Les simplifications par rapport à une situation réelle sont énumérées à la fin de l'interface.

4.1 Interfaces Encaveur-Transporteur-Détaillant

Trois interfaces sont regroupées ici (Encaveur-Transporteur, Transporteur-Détaillant et Encaveur-Détaillant) en raison de l'étroite interdépendance entre les acteurs concernés. Ils décrivent les échanges à l'occasion de la livraison par le transporteur d'un chargement de vin expédié par l'encaveur à destination du détaillant.



Description des flux:

A. *Commande*: Une commande est rédigée par un détaillant et envoyée à l'encaveur. Elle énumère les produits que le détaillant souhaite recevoir de la part de l'encaveur. Une commande comporte les éléments suivants:
   - Commande: numéro unique identifiant la commande.
   - Client: identité de l'entité formulant la commande (le détaillant).
   - Fournisseur: identité de l'entité destinataire de la commande (l'encaveur).
   - Date: date et heure de l'envoi de la commande.
   - Articles: liste des articles commandés, avec, pour chaque article:
     - Code: code de l'article (correspond à un type de vin).
     - Quantité: quantité commandée (nombre de cartons).

B. *Demande de transport*: Dans le but d'acheminer les produits commandés au client, l'encaveur formule une demande de transport qu'il adresse à un transporteur. La demande de transport comporte les informations suivantes:
   - Demande: numéro unique identifiant la demande de transport.
   - Expéditeur: identité de l'entité envoyant la marchandise (l'encaveur).
   - Transporteur: identité de l'entité mandatée pour effectuer le transport (le transporteur).
   - Destinataire: identité de l'entité destinataire pour la marchandise (le détaillant).
   - Date: date et heure de l'envoi de la demande de transport.
   - Commande: numéro de la commande à laquelle correspond le transport.

- Date de livraison: date de livraison souhaitée (le jour même).
- Capacité nécessaire: poids et volume totaux pour l'ensemble du chargement (en kg et m^3).

C. *Confirmation de transport*: Sur réception d'une demande de transport, le transporteur établit une confirmation de transport qu'il adresse à l'auteur de la demande. Il confirme par ce document qu'il a bien reçu la demande de transport et qu'il est prêt à l'exécuter. La confirmation de transport comporte les éléments suivants:
- Transporteur: identité de l'entité confirmant le transport (le transporteur).
- Expéditeur: identité de l'entité à l'origine de la demande de transport (l'encaveur).
- Destinataire: identité de l'entité destinataire pour la marchandise à transporter (le détaillant).
- Demande: numéro unique identifiant la demande de transport à laquelle la confirmation de transport répond.
- Date: date et heure de l'envoi de la confirmation de transport.
- Date de livraison: date de livraison prévue (le jour même).
- Transport: numéro d'identification de l'opération de transport.
- Véhicules: liste des véhicules prévus pour le transport, avec, pour chaque véhicule:
  - Identification: identification unique du véhicule (plaque d'immatriculation).
  - Capacité: capacité maximale du véhicule, en kg et m^3.

D. *Avis de livraison*: L'avis de livraison annonce au client (le détaillant) que l'encaveur va lui envoyer les produits correspondant à une commande. Il comporte les éléments suivants:
- Fournisseur: identité de l'entité destinataire de la commande (l'encaveur).
- Client: identité de l'entité destinataire de l'avis de livraison (le détaillant).
- Date: date et heure de l'envoi de l'avis de livraison.
- Date de livraison: date de livraison prévue (le jour même).
- Transport: numéro d'identification de l'opération de transport.
- Commande: numéro unique identifiant la commande à laquelle correspond l'avis de livraison.

E. *Chargement*: Un chargement est composé de 0 à C_P palettes pleines, 0 à C_C cartons pleins, et des informations relatives au chargement (bulletin de livraison, par exemple). Ces informations comportes les éléments suivants:
- Client: identité du destinataire du chargement (le détaillant).
- Fournisseur: identité de l'expéditeur du chargement (l'encaveur).
- Commande: numéro unique identifiant la commande à laquelle correspond le chargement.
- Date: date et heure de conditionnement du chargement.
- Colis: Liste des colis contenus dans le chargement, avec, pour chaque colis:
  - Identifiant: identifiant unique du colis (i.e., de la palette si le colis est une palette, du carton si le colis est un carton).
  - Packaging: Liste des identifiants contenus dans le colis (identifiants bouteilles et cartons), structurée de sorte à ce que les relations contenant-contenu soient clairement identifiées.
- Articles: liste des articles contenus dans le chargement, avec, pour chaque article:
  - Code: code de l'article (correspond à un type de vin).
  - Packaging: Liste des identifiants liés au produit (identifiants bouteilles, cartons et palettes), structurée de sorte à ce que les relations contenant-contenu soient clairement identifiées.

F. *Confirmation de prise en charge*: Confirmation par le transporteur qu'il a pris en charge (i.e., qu'il a chargé dans son camion). La confirmation de prise en charge comporte les éléments suivants:
- Transporteur: identité de l'entité confirmant la prise en charge (le transporteur).
- Expéditeur: identité de l'entité ayant remis le chargement au transporteur (l'encaveur).
- Destinataire: identité de l'entité destinataire du chargement (le détaillant).
- Date: date et heure de la confirmation de prise en charge.
- Date prise en charge: date et heure de prise en charge.
- Transport: numéro d'identification de l'opération de transport.
- Commande: numéro unique identifiant la commande à laquelle correspond le chargement.

G. *Confirmation de livraison*: Confirmation par le détaillant qu'il a bien reçu le chargement livré par le transporteur. La confirmation de livraison comporte les éléments suivants:
- Transporteur: identité de l'entité destinataire de la confirmation de livraison (le transporteur).
- Expéditeur: identité de l'entité ayant remis le chargement au transporteur (l'encaveur).
- Destinataire: identité de l'entité destinataire du chargement (le détaillant).
- Date: date et heure de la confirmation de livraison.
- Date livraison: date et heure de réception du chargement.
- Transport: numéro d'identification de l'opération de transport.
- Commande: numéro unique identifiant la commande à laquelle correspond le chargement.

H. *Accusé de réception*: Confirmation par le détaillant qu'il a bien reçu le chargement envoyé par l'encaveur. L'accusé de réception comporte les éléments suivants:
- Client: identité de l'entité émettrice de l'accusé de réception (le détaillant).
- Fournisseur: identité de l'entité destinataire de l'accusé de réception (l'encaveur).
- Date: date et heure de l'envoi de l'accusé de réception.
- Date réception: date et heure de réception du chargement.
- Commande: numéro unique identifiant la commande dont on accuse réception.

Simplifications:

Encaveur - Détaillant:
- L'unité de commande ne peut être que le carton. Les commandes comportant des bouteilles à l'unité ne sont pas admises.
- La facturation et le payement relatifs aux commandes entre détaillant et encaveur ne sont pas modélisés. Tout se passe comme si l'encaveur livrait ses produits gratuitement au détaillant.
- La gestion du catalogue d'articles mis à disposition par l'encaveur n'est pas modélisée. On part du principe que le détaillant sait quels sont les produits proposés par l'encaveur et connaît leurs codes respectifs.
- Toutes les commandes sont livrées en une fois. Les livraisons partielles ne sont pas autorisées.
- Une livraison correspond toujours à une seule commande. La consolidation de plusieurs commandes dans une livraison unique n'est pas autorisée.

Transporteur:
- Une demande de transport n'est jamais refusée par le transporteur. La demande est acceptée automatiquement et un véhicule est toujours disponible pour l'exécuter dans les plus brefs délais.
- Une livraison passe toujours par le transporteur. Un détaillant ne peut en aucun cas se rendre chez l'encaveur pour prendre lui-même livraison de sa commande, et, à l'inverse, l'encaveur ne peut en aucun cas se rendre lui-même chez le détaillant pour effectuer la livraison.
- La facturation et le payement relatifs aux opérations de transport ne sont pas modélisés. Tout se passe comme si le transporteur offrait ses services gratuitement.
- Le transporteur ne dispose que d'un unique véhicule. Il ne peut donc effectuer plusieurs transports en parallèle et n'enverra jamais plus d'un unique véhicule pour effectuer un transport.
- L'import-export n'est pas géré. Les formalités douanières et les agents de dédouanement ne sont pas modélisés.
- Le transport ne comporte toujours qu'un unique segment. Le transport multi-segments faisant intervenir plusieurs transporteurs (par exemple encaveur -> camion -> bateau -> train -> camion -> détaillant) n'est pas géré.

Général:
- Chaque acteur est situé géographiquement en un lieu unique. L'identité d'un acteur suffit donc à connaître sa position géographique exacte (i.e., on ne risque jamais de se tromper d'adresse).
- Il n'existe qu'une seule taille de palettes, capable de contenir un nombre n_P de cartons (avec n_P>1). La taille des palettes ne peut changer que de manière globale, i.e., par le remplacement de toutes les palettes utilisées dans la démonstration. Par ailleurs, en cas de changement de taille de palette, l'ensemble des données historiques de la démonstration doivent être soit effacées, soit adaptées pour refléter la nouvelle taille des palettes.

Hes·SO
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition du scénario

- Il n'existe qu'une seule taille de cartons, capable de contenir un nombre n_c de bouteilles (avec n_c>1). La taille des cartons ne peut changer que de manière globale. i.e. par le remplacement de tous les cartons utilisés dans la démonstration. Par ailleurs, en cas de changement de taille de carton, l'ensemble des données historiques de la démonstration doivent être soit effacées, soit adaptées pour refléter la nouvelle taille des cartons.
- Un carton ne peut contenir qu'un seul type de vin/bouteille (i.e., on ne mélange pas différents types de vin/bouteilles dans un même carton).
- La gestion des dommages en cours de transport et des retours de marchandise n'est pas modélisée. Les commandes sont toujours considérées comme satisfaites lorsque l'encaveur a envoyé la quantité commandée. Si pour une raison quelconque cette quantité ne parvient pas intégralement au détaillant, celui-ci doit être en mesure de clore la/les commande(s) correspondante(s) manuellement sans conséquences pour l'encaveur et le transporteur.

Laurent Bagnoud
laurent.bagnoud@hevs.ch
+41 27 606 90 13
+41 27 606 90 00

Définition des interfaces techniques

Description :

Document reçu en début de projet définissant les interfaces des services web des acteurs de la chaîne logistique.

Auteur Simon Martin, Laurent Bagnoud
Date 27.3.2007
Thème Interfaces techniques

Destinataire Equipe de projet
Pour info

BPMS-4-SME – Définition des interfaces techniques

Doc-ID	BPMS-4-SME.030
Version	0.2
Statut	En révision
Sauvegarde sur	Q:\II\BaA\Ra&D\RCSO\BPMS_4_SMEs\ProjectDoc\08_Requirements
Archivage	5 ans

Laurent Bagnoud
laurent.bagnoud@hevs.ch
☏+41 27 606 90 13
🖷 +41 27 606 90 00

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition des interfaces techniques

Modifications

Version	Date	Exécutant	Remarques sur la modification
0.1	15.3.2007	Simon Martin, Laurent Bagnoud	Première version
0.2	5.4.2007	Laurent Bagnoud	Divers modifications

Contrôles - Reviews

Version	Date du contrôle	Exécutant(s)	Remarques

Libérations

Version	Date de libération	Personne en charge	Remarques

Simon Martin
simon.martin@hevs.ch
☏+41 27 606 90 33
🖷 +41 27 606 90 00

Table des matières

Simon Martin
simon.martin@hevs.ch
☏+41 27 606 90 33
🖷 +41 27 606 90 00

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition des interfaces techniques

1. Introduction

Ce document a pour objectif de définir les interfaces techniques liées au scénario défini dans le document *BPMS-4-SME.012 Définition du scénario*.

1.1 Domaine de validité

Ce document est valable pour le projet RCSO BPMS-4-SME.

1.2 Public cible et conditions préalables à la lecture

Ce document est destiné à toutes les personnes concernées par le projet BPMS-4-SME.

1.3 Termes et abbréviations

Terme / Abbréviation	Définition / Description
BPMS	Business Process Management System
BPM	Business Process Management

Tableau 1 Termes et abbréviations

1.4 Documents référencés

Nr.	Doc-ID	Document	Auteur	Version
[1]	BPMS-4-SME.012	Définition du scénario	Nicole Glassey, Laurent Bagnoud	0.3
[2]				
[3]				

Tableau 2 Documents référencés

Simon Martin
simon.martin@hevs.ch
☏+41 27 606 90 33
🖷 +41 27 606 90 00

2. Technologies et Standards utilisés

Les interactions entre acteurs de la chaîne logisitique sont effectuées par des WebServices.

3. Identifiants uniques des acteurs

Chaque acteur possède un identifiant connu de tous les acteurs du scénario, antérieurement à l'exécution du scénario. Ces identifiants, des entiers, sont définis ci-dessous :

Acteur	Identifiant
Détaillant	1000
Encaveur	2000
Transporteur	3000

Tableau 3 : Identifiants des Acteurs

Chaque système de la chaîne logisitique est implémenté de manière à pouvoir facilement intégrer de nouveaux acteurs (p.ex. de nouveaux Détaillants ou Encaveurs).

4. Identifiants uniques des produits

Pour raison de simplification, la gamme de produits disponible se résume aux articles suivants :

Produit	Identifiant
Johannisberg	9001
Œil de Perdrix	9002
Cornalin	9003
Merlot	9004

Tableau 4 : Identifiants des Produits

Chaque système de la chaîne logisitique est implémenté de manière à pouvoir facilement intégrer de nouveaux produits (p.ex. Rèze, Dionlinoir, etc.).

5. Définition des interfaces techniques

Chaque acteur de la chaîne logistique définie dans le document *BPMS-4-SME.012 Définition du scénario* implémente et met à disposition ses services sur la base des informations contenues dans les chapitres suivants.

5.1 Interface de l'Encaveur

5.1.1 Classe WSEncavor

WSEncavor
newOrder(int retailerId, int[][] orderItems) : int confirmTransport(int convoyerId, int transportRequestId, int retailerId, Date deliveryDate, int transportId) : int confirmAoR(int convoyerId, int transportId, int orderId, Date aorDate) : int confirmAoD(int retailerId, int orderId, Date loadingReceptionDate) : int

5.1.2 Service newOrder – Commande

Signature du service	newOrder(int retailerId, int[][] orderItems) : int
Description du service	Ce service permet l'enregistrement d'une commande chez l'encaveur
SLA	Disponibilité : du lundi au vendredi de 8h à 5h Temps de réponse : 5 secondes max Fréquence : jusqu'à 10 commandes par minute Volume : indéfini Cycle de versioning : annonce aux utilisateurs 1 mois avant tout changement
Sécurité	

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
retailerId	Int	Identifiant unique du détaillant appelant le service	Définis au chapitre 3 à la page 5.
orderItems	Int[][]	Composé d'un code de produit et d'une quantité (nombre de bouteilles)	Définis au chapitre 4 à la page 5. Commande max par produit : 1'000 unités.

Retour		
Type	**Description**	**Valeurs possibles**
Int	Identifiant unique de la commande chez l'encaveur	0 : Erreur > 0 : Commande acceptée

5.1.3 Service confirmTransport – Confirmation de transport

Signature du service	confirmTransport(int convoyerId, int transportRequestId, int retailerId, Date deliveryDate, int transportId) : int
Description du service	Ce service permet l'envoi d'une confirmation de transport auprès de l'encaveur par le transporteur
SLA	Disponibilité : 24h/24h Temps de réponse : 5 secondes max Fréquence : jusqu'à 10 appels par minute Volume : indéfini Cycle de versioning : annonce aux utilisateurs 1 mois avant tout changement
Sécurité	

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
convoyerId	Int	Identifiant unique du transporteur	Défini au chapitre 3 à la page 5
transportRequestId	Int	Numéro identifiant de manière unique la demande de transport	1-1'000'000'000
retailerId	Int	Identifiant unique du détaillant	Défini au chapitre 3 à la page 5
deliveryDate	Date	Date de livraison prévue	Date (dans le futur)
transportId	int	Identifiant unique du transporteur	1-1'000'000'000

Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité de la confirmation de transport	0 : valide 1 : transportRequestId invalide 2 : convoyerId invalide 3 : retailerId invalide 4 : deliveryDate < date du jour 5 : service web hors-service

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition des interfaces techniques

5.1.4 Service confirmAoR – Confirmation de prise en charge

Signature du service	confirmAoR(int convoyerId, int transportId, int orderId, Date aorDate) : int
Description du service	Ce service permet l'envoi d'une confirmation de prise en charge du chargement auprès de l'encaveur par le transporteur
SLA	Disponibilité : 24h/24h Temps de réponse : 5 secondes max Fréquence : jusqu'à 10 appels par minute Volume : indéfini Cycle de versioning : annonce aux utilisateurs 1 mois avant tout changement
Sécurité	

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
convoyerId	Int	Identifiant unique du Transporteur	Défini au chapitre 3 à la page 5
transportId	Int	Identifiant unique du numéro de transport chez le Transporteur	1-1'000'000'000
orderId	Int	Identifiant unique du numéro de commande chez l'Encaveur	1-1'000'000'000
aorTmstp	Timestamp	Date et heure de la prise en charge	Timestamp unique

Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité de la confirmation de prise en charge	0 : valide 1 : transportId invalide 2 : convoyerId invalide 3 : orderId invalide 4 : service web hors-service

Simon Martin
simon.martin@hevs.ch
☏+41 27 606 90 33
🖷 +41 27 606 90 00

Titre : BPMS-4-SME Définition des interfaces techniques
Doc-ID : BPMS-4-SME.030
Version 0.1

Institut Informatique de Gestion
HES-SO Valais
TechnoArk 3
3960 Sierre

5.1.5 Service confirmAoD – Accusé de réception

Signature du service	confirmAoD(int retailerId, int orderId, Date loadingReceptionDate) : int
Description du service	Ce service permet l'envoi d'un accusé de réception auprès de l'encaveur par le détaillant
SLA	Disponibilité : 24h/24h Temps de réponse : 5 secondes max Fréquence : jusqu'à 10 appels par minute Volume : indéfini Cycle de versioning : annonce aux utilisateurs 1 mois avant tout changement
Sécurité	

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
retailerId	Int	Identifiant unique du Détaillant	Défini au chapitre 3 à la page 5
orderId	Int	Identifiant unique de l'ordre chez l'Encaveur	1-1'000'000'000
loadingReceptionTmstp	Timestamp	Date de réception du chargement	Timestamp

Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité de l'accusé de réception	0 : valide 1 : orderId invalide 2 : retailerId invalide 3 : service web hors-service

5.2 Interface du détaillant

5.2.1 Classe WSRetailer

WSRetailer
loadingLeft(int convoyerId, int encavorId, int orderId, Date loadingLeftDate, int[][] orderItems) : int noticeOfDelivery(int encavorId, Date deliveryDate, int transportId, int orderId) : int

5.2.2 Service loadingLeft – Chargement parti

Signature du service	loadingLeft(int convoyerId, int encavorId, int orderId, Date loadingLeftDate, int[][] orderItems) : int
Description du service	Ce service permet l'envoi d'informations relatives au chargement auprès du détaillant par le transporteur
SLA	Disponibilité : 24h/24h Temps de réponse : 5 secondes max Fréquence : jusqu'à 10 appels par minute Volume : indéfini Cycle de versioning : annonce aux utilisateurs 1 mois avant tout changement
Sécurité	

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
convoyerId	Int	Identifiant unique du Transporteur	Définis au chapitre 3 à la page 5.
encavorId	Int	Identifiant unique de l'Encaveur	Définis au chapitre 3 à la page 5.
orderId	int	Identifiant unique du numéro de commander chez l'Encaveur	1-1'000'000'000
loadingLeftTmsp	Timestamp	Date et heure de départ du chargement	Timestamp
orderItems	Int[][]	Composé d'un code de produit et d'une quantité (nombre de bouteilles)	Définis au chapitre 4 à la page 5. Commande max par produit : 1'000 unités.

Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité des informations de chargement	0 : valide 1 : orderId invalide 2 : orderItems invalides 3 : convoyerId invalide 4 : encavorId invalide 5 : service web hors-service

5.2.3 Service noticeOfDelivery – Avis de livraison

Signature du service	noticeOfDelivery(int encavorId, Date deliveryDate, int transportId, int orderId): int
Description du service	Ce service permet l'envoi d'un avis de livraison auprès du détaillant par l'encaveur
SLA	Disponibilité : 24h/24h Temps de réponse : 5 secondes max Fréquence : jusqu'à 10 appels par minute Volume : indéfini Cycle de versioning : annonce aux utilisateurs 1 mois avant tout changement
Sécurité	

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
encavorId	Int	Identifiant unique de l'Encaveur	Défini au chapitre 3 à la page 5
deliveryDate	Date	Date de livraison prévue	Date (dans le futur)
transportId	int	Identifiant unique du Transporteur	1-1'000'000'000
orderId	int	Identifiant unique de la commande chez l'Encaveur	1-1'000'000'000

Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité de l'avis de livraison	0 : valide 1 : transportId invalide 2 : orderId invalide 3 : encavorId invalide 4 : deliveryDate < date du jour 5 : service web hors-service

5.3 Interface du Tansporteur

5.3.1 Classe WSConvoyer

WSConvoyer
requestTransport (int encavorId, int retailerId, int orderId, Date deliveryDate) : int loadingConditioned(int encavorId, int retailerId, int orderId, int[][] orderItems) : int confirmDelivery(int retailerId, int transportId, int orderId, Date deliveryDate) : int

5.3.2 Service requestTransport – Demande de transport

Signature du service	requestTransport (int encavorId, int retailerId, int orderId, Date deliveryDate) : int
Description du service	Ce service permet l'envoi d'une demande de transport auprès du transporteur par l'encaveur

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
encavorId	Int	Identifiant unique	Défini au chapitre 3 à la page 5
retailerId	Int	Identifiant unique	Défini au chapitre 3 à la page 5
orderId	int	Identifiant unique	1-1'000'000'000
deliveryDate	Date	Date de livraison souhaitée	Date (dans le futur)

Retour		
Type	**Description**	**Valeurs possibles**
Int	Retourne l'identifiant de la demande de transport	0 : Erreur > 0 : Demande acceptée

5.3.3 Service loadingConditioned – Chargement conditionné

Signature du service	loadingConditioned(int encavorId, int retailerId, int orderId, int[][] orderItems) : int
Description du service	Ce service permet l'envoi d'un avis auprès du transporteur par l'encaveur lui annonçant que le chargement est conditionné

Paramètres d'input			
Paramètre	**Type**	**Description**	**Valeurs possibles**
encavorId	Int	Identifiant unique	Défini au chapitre 3 à la page 5
retailerId	Int	Identifiant unique	Défini au chapitre 3 à la page

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

BPMS-4-SME
Définition des interfaces techniques

			5
orderId	Int	Identifiant unique	1-1'000'000'000
orderItems	Int[][]	Composé d'un code de produit et d'une quantité (nombre de bouteilles)	Définis au chapitre 4 à la page 5. Commande max par produit : 1'000 unités.

Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité de l'annonce	0 : valide 1 : orderId invalide 2 : orderItems invalides 3 : encavorId invalide 4 : retailerId 5 : service web hors-service

5.3.4 Service confirmDelivery – Confirmation de livraison

Signature du service	confirmDelivery(int retailerId, int transportId, int orderId, Date deliveryDate) : int
Description du service	Ce service permet l'envoi d'une confirmation de livraison auprès du transporteur par le détaillant

Paramètres d'input		
Paramètre	**Type**	**Description**	**Valeurs possibles**
retailerId	Int	Identifiant unique du Détaillant	Défini au chapitre 3 à la page 5
transportId	Int	Identifiant unique du numéro de transport chez le Transporteur	1-1'000'000'000
orderId	Int	Identifiant unique de la commande chez l'Encaveur	1-1'000'000'000
deliveryTmstp	Timestamp	Date et heure de livraison effective	Timestamp

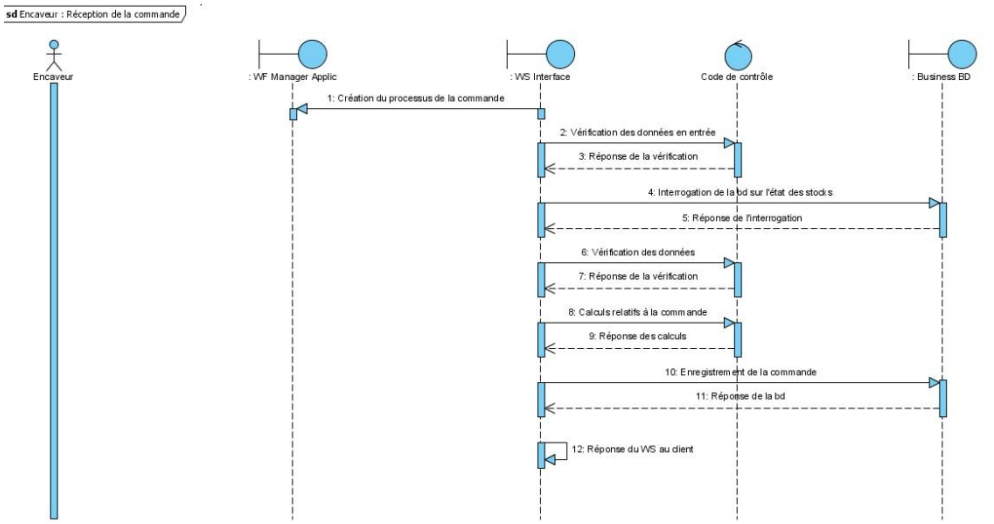
Retour		
Type	**Description**	**Valeurs possibles**
Int	Validité de la confirmation	0 : valide 1 : transportId invalide 2 : orderId invalide 3 : retailerId invalide 4 : service web hors-service

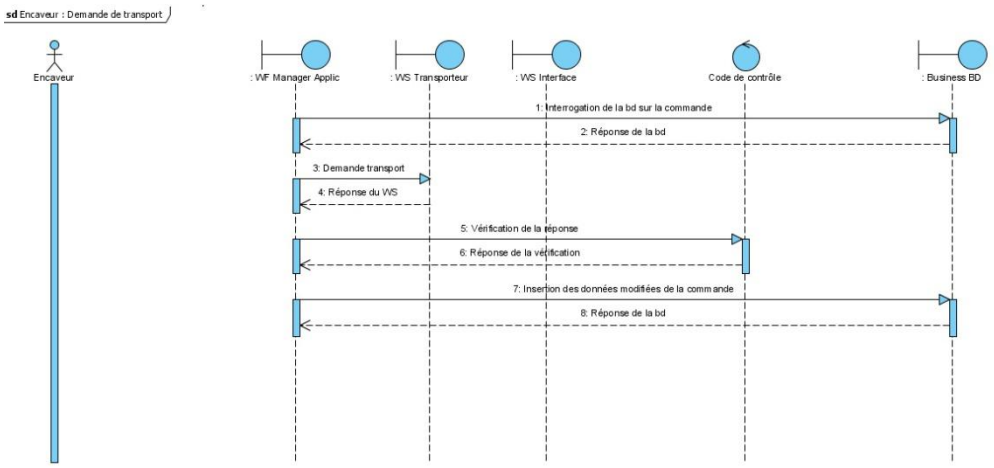
Simon Martin
simon.martin@hevs.ch
+41 27 606 90 33
+41 27 606 90 00

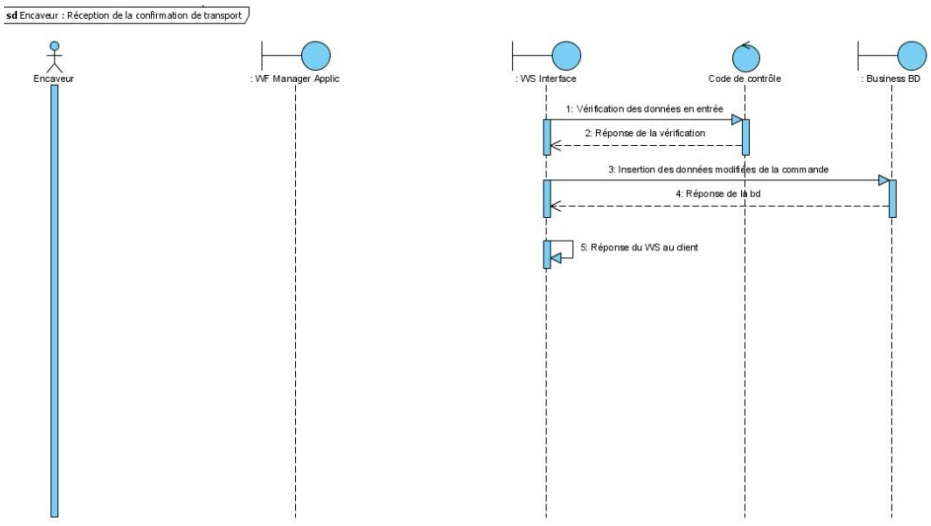
Diagrammes de séquence

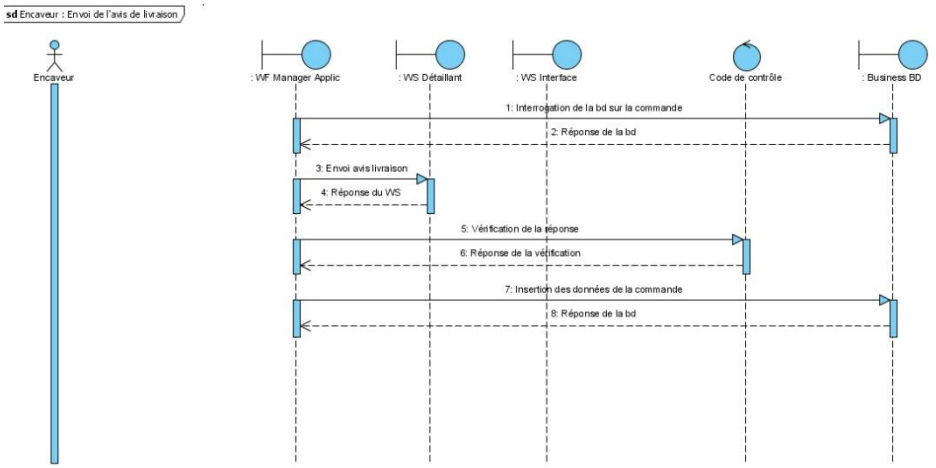
Description :

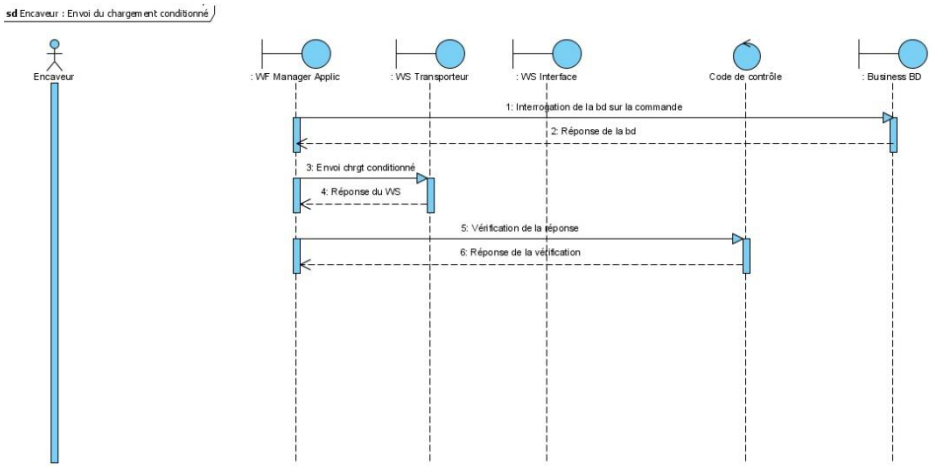
Diagrammes de séquence établis pour chacun des Use Case des acteurs durant la phase d'analyse du travail.

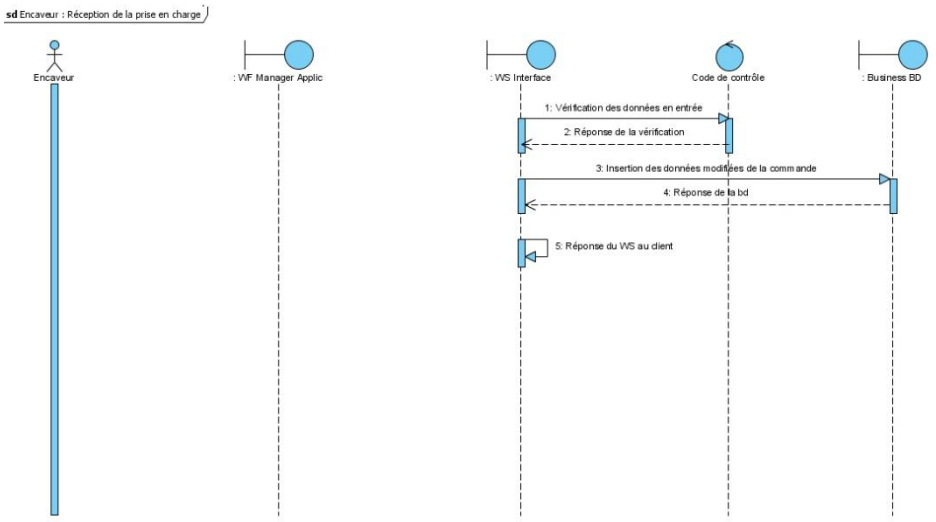

sd Encaveur : Réception de la commande
Encaveur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Création du processus de la commande
2: Vérification des données en entrée
3: Réponse de la vérification
4: Interrogation de la bd sur l'état des stocks
5: Réponse de l'interrogation
6: Vérification des données
7: Réponse de la vérification
8: Calculs relatifs à la commande
9: Réponse des calculs
10: Enregistrement de la commande
11: Réponse de la bd
12: Réponse du WS au client

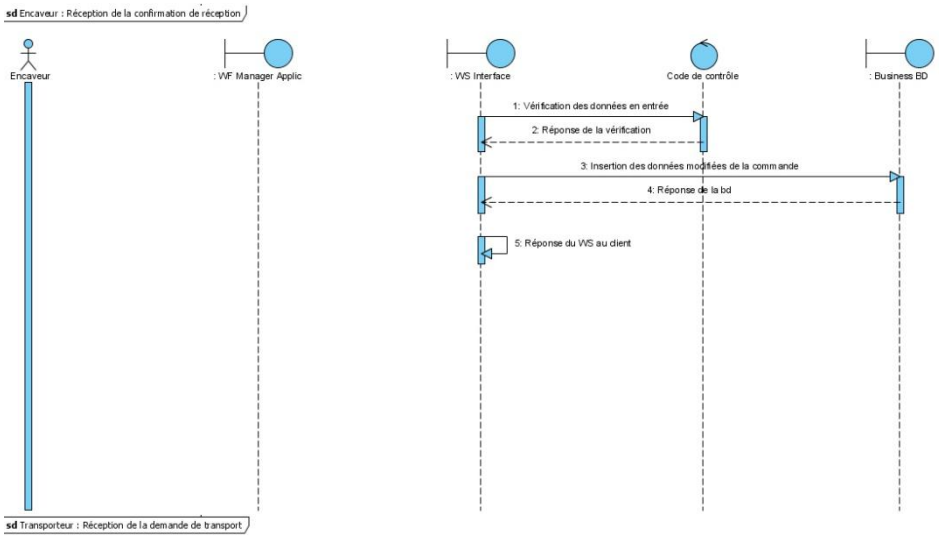

sd Encaveur : Demande de transport
Encaveur
: WF Manager Applic
: WS Transporteur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur la commande
2: Réponse de la bd
3: Demande transport
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données modifiées de la commande
8: Réponse de la bd

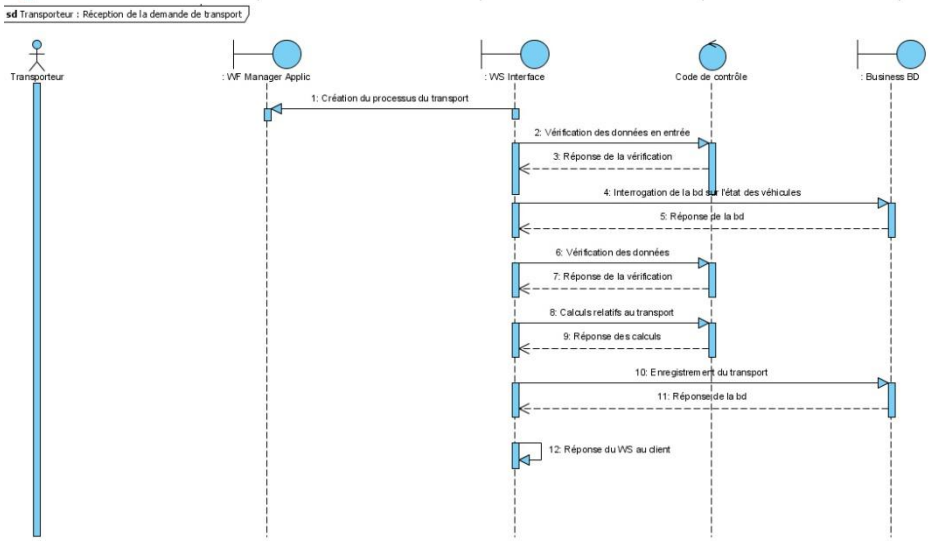

sd Encaveur : Réception de la confirmation de transport
Encaveur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées de la commande
4: Réponse de la bd
5: Réponse du WS au client

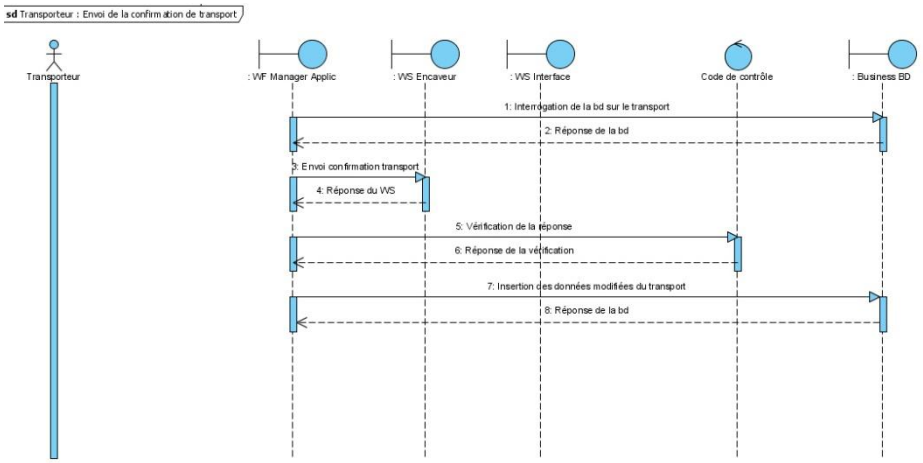

sd Encaveur : Envoi de l'avis de livraison
Encaveur
: WF Manager Applic
: WS Détaillant
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur la commande
2: Réponse de la bd
3: Envoi avis livraison
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données de la commande
8: Réponse de la bd

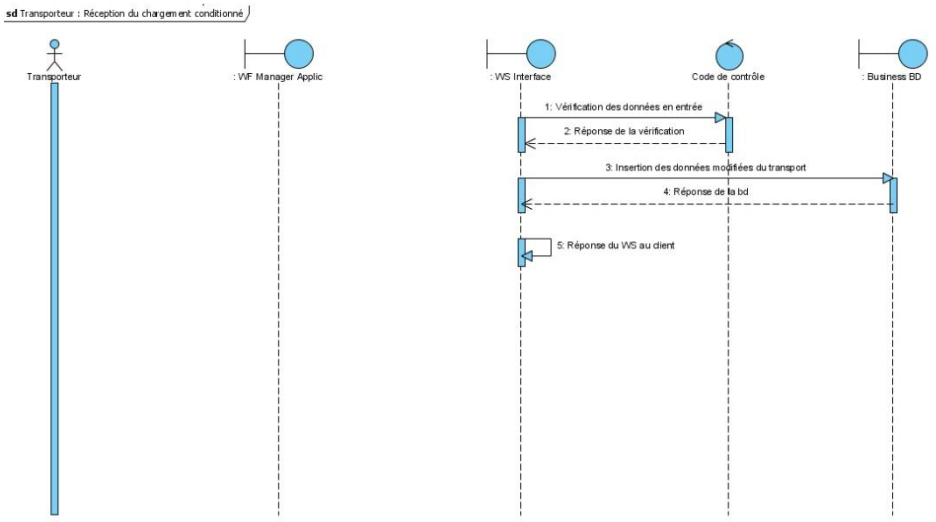

sd Encaveur : Envoi du chargement conditionné
Encaveur
: WF Manager Applic
: WS Transporteur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur la commande
2: Réponse de la bd
3: Envoi chrgt conditionné
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification

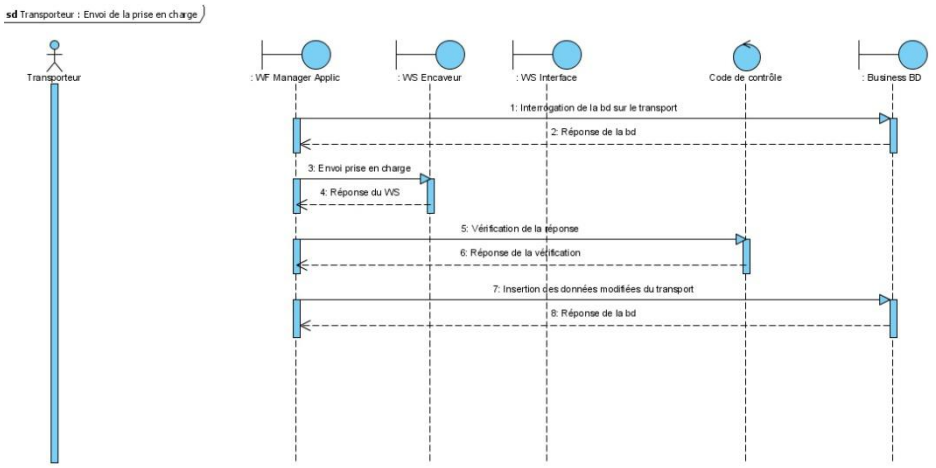

sd Encaveur : Réception de la prise en charge
Encaveur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées de la commande
4: Réponse de la bd
5: Réponse du WS au client

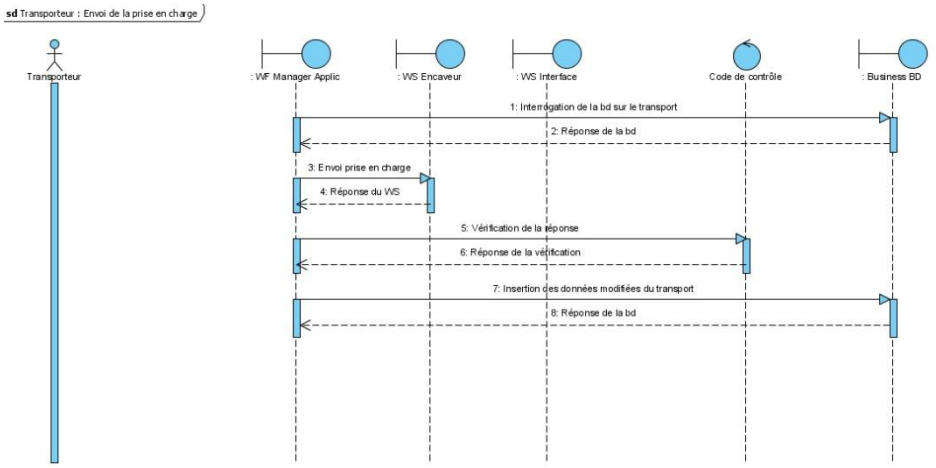

sd Encaveur : Réception de la confirmation de réception
Encaveur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées de la commande
4: Réponse de la bd
5: Réponse du WS au client

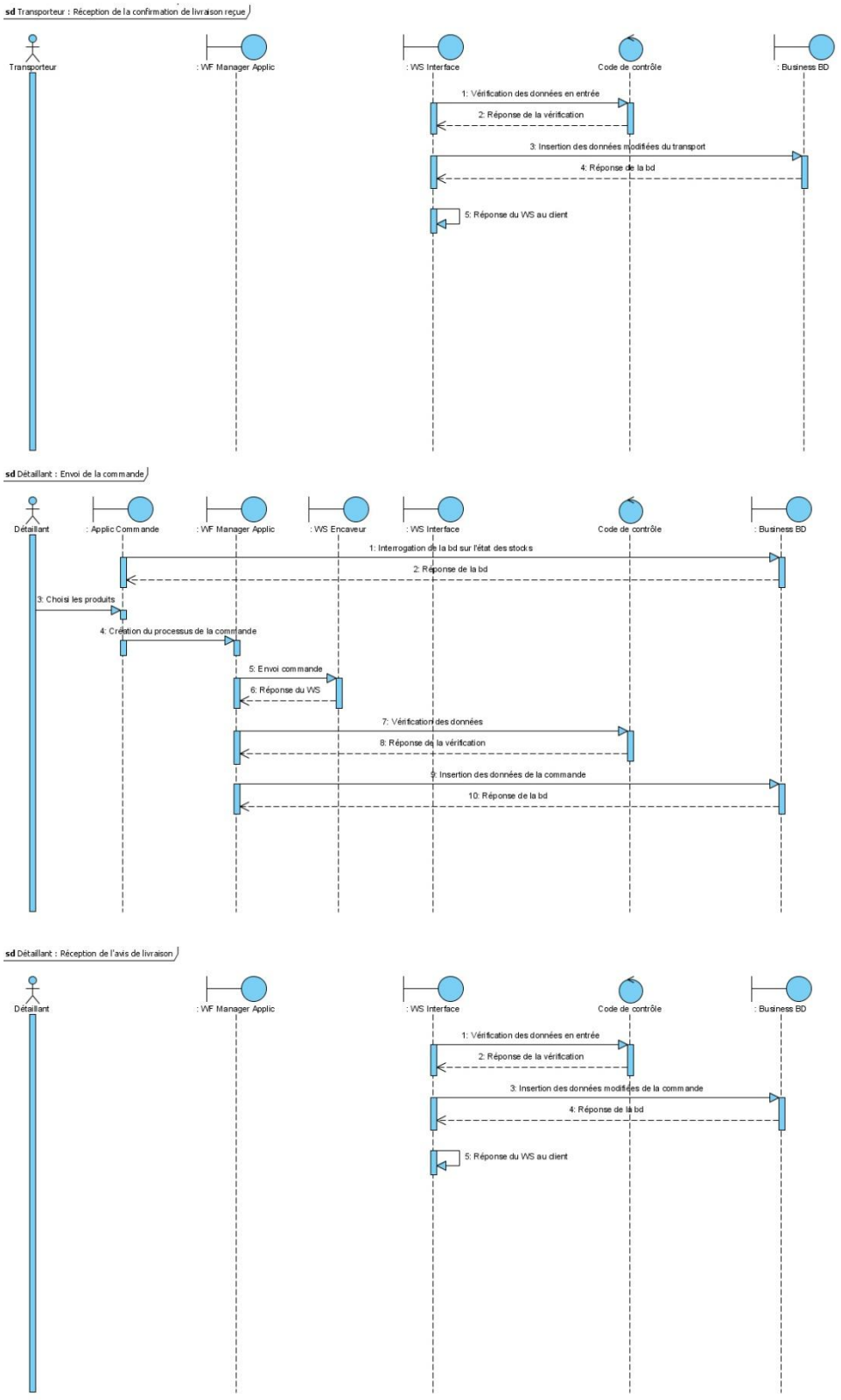

sd Transporteur : Réception de la demande de transport
Transporteur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Création du processus du transport
2: Vérification des données en entrée
3: Réponse de la vérification
4: Interrogation de la bd sur l'état des véhicules
5: Réponse de la bd
6: Vérification des données
7: Réponse de la vérification
8: Calculs relatifs au transport
9: Réponse des calculs
10: Enregistrement du transport
11: Réponse de la bd
12: Réponse du WS au client

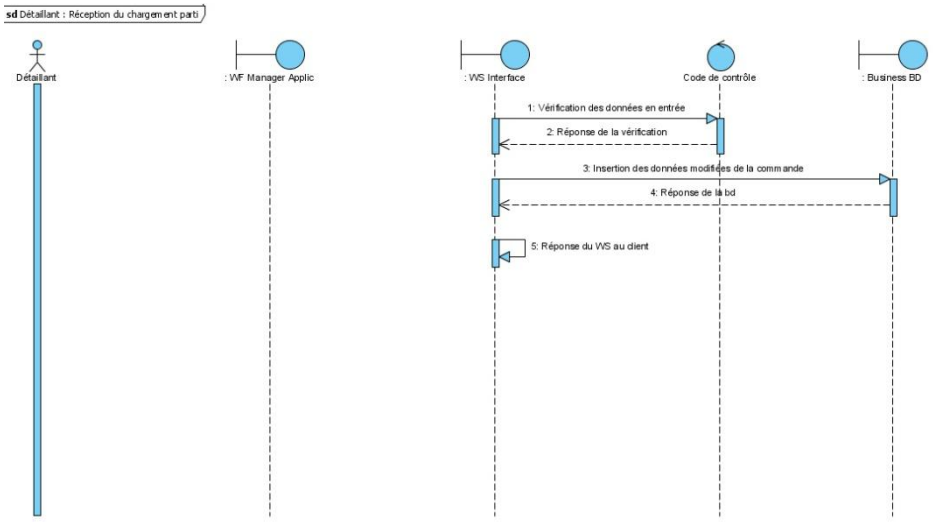

sd Transporteur : Envoi de la confirmation de transport
Transporteur
: WF Manager Applic
: WS Encaveur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur le transport
2: Réponse de la bd
3: Envoi confirmation transport
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données modifiées du transport
8: Réponse de la bd

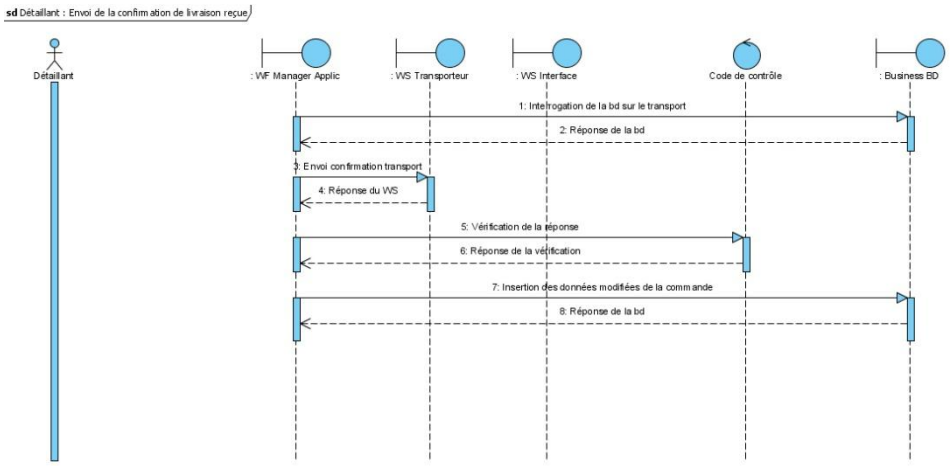

sd Transporteur : Réception du chargement conditionné
Transporteur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées du transport
4: Réponse de la bd
5: Réponse du WS au client

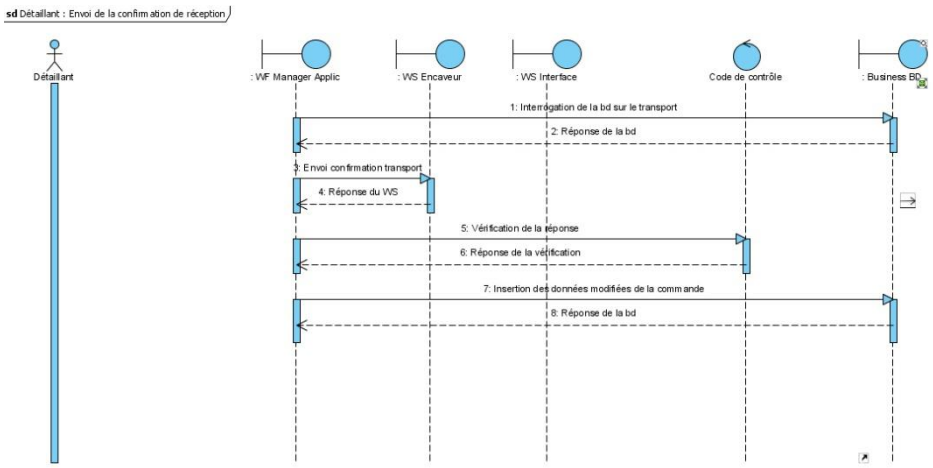

sd Transporteur : Envoi de la prise en charge
Transporteur
: WF Manager Applic
: WS Encaveur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur le transport
2: Réponse de la bd
3: Envoi prise en charge
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données modifiées du transport
8: Réponse de la bd


sd Transporteur : Envoi de la prise en charge
Transporteur
: WF Manager Applic
: WS Encaveur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur le transport
2: Réponse de la bd
3: Envoi prise en charge
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données modifiées du transport
8: Réponse de la bd


sd Transporteur : Réception de la confirmation de livraison reçue
Transporteur
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées du transport
4: Réponse de la bd
5: Réponse du WS au client
sd Détaillant : Envoi de la commande
Détaillant
: Applic Commande
: WF Manager Applic
: WS Encaveur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur l'état des stocks
2: Réponse de la bd
3: Choisi les produits
4: Création du processus de la commande
5: Envoi commande
6: Réponse du WS
7: Vérification des données
8: Réponse de la vérification
9: Insertion des données de la commande
10: Réponse de la bd
sd Détaillant : Réception de l'avis de livraison
Détaillant
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées de la commande
4: Réponse de la bd
5: Réponse du WS au client


sd Détaillant : Réception du chargement parti
Détaillant
: WF Manager Applic
: WS Interface
Code de contrôle
: Business BD
1: Vérification des données en entrée
2: Réponse de la vérification
3: Insertion des données modifiées de la commande
4: Réponse de la bd
5: Réponse du WS au client


sd Détaillant : Envoi de la confirmation de livraison reçue
Détaillant
: WF Manager Applic
: WS Transporteur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur le transport
2: Réponse de la bd
3: Envoi confirmation transport
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données modifiées de la commande
8: Réponse de la bd


sd Détaillant : Envoi de la confirmation de réception
Détaillant
: WF Manager Applic
: WS Encaveur
: WS Interface
Code de contrôle
: Business BD
1: Interrogation de la bd sur le transport
2: Réponse de la bd
3: Envoi confirmation transport
4: Réponse du WS
5: Vérification de la réponse
6: Réponse de la vérification
7: Insertion des données modifiées de la commande
8: Réponse de la bd

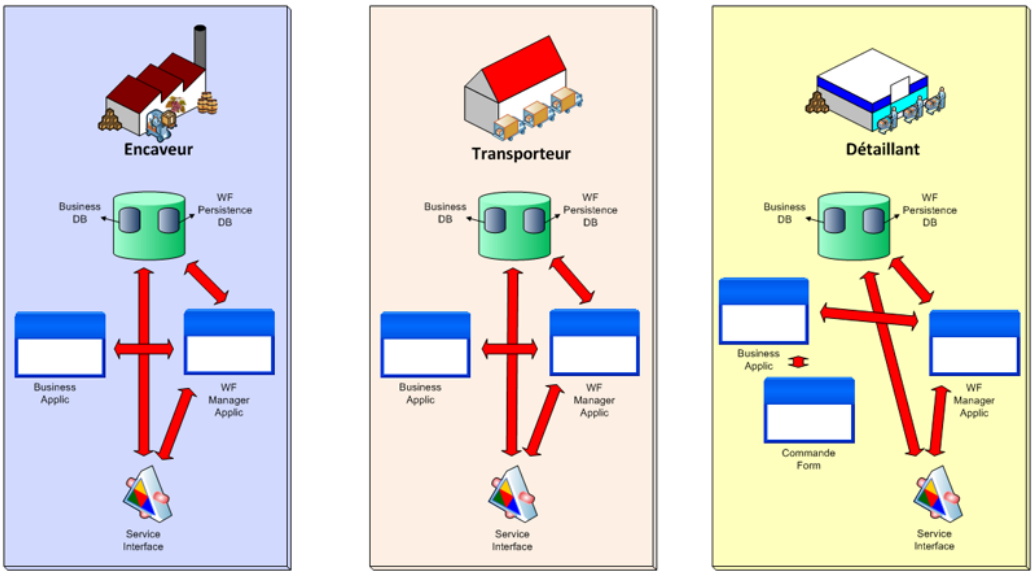
Diagrammes de l'architecture

Description :

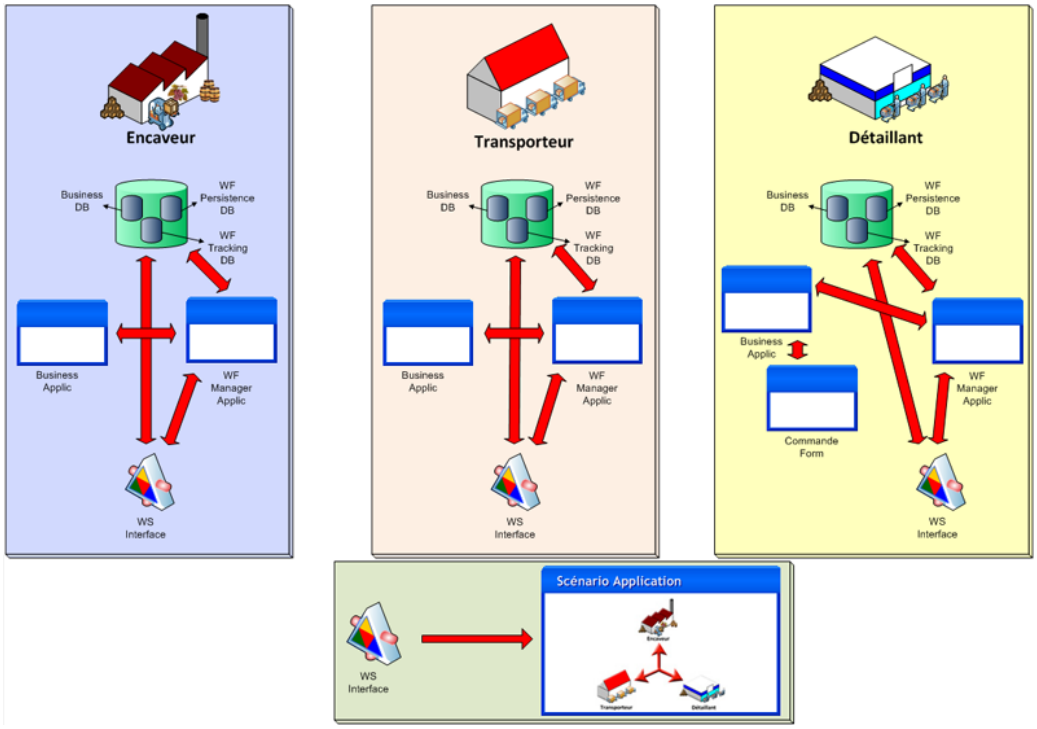
Diagrammes de l'architecture et des communications entre les applications établis durant la phase d'analyse.

Architecture Must Have
Architecture Nice to Have
Intercommunication Must Have
Intercommunication Nice to Have

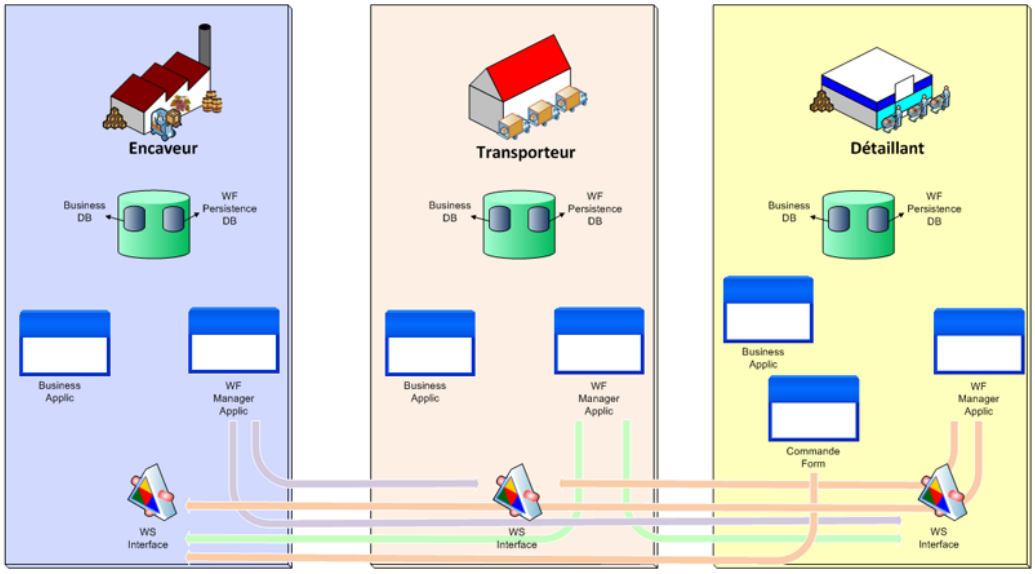
Architecture Must Have



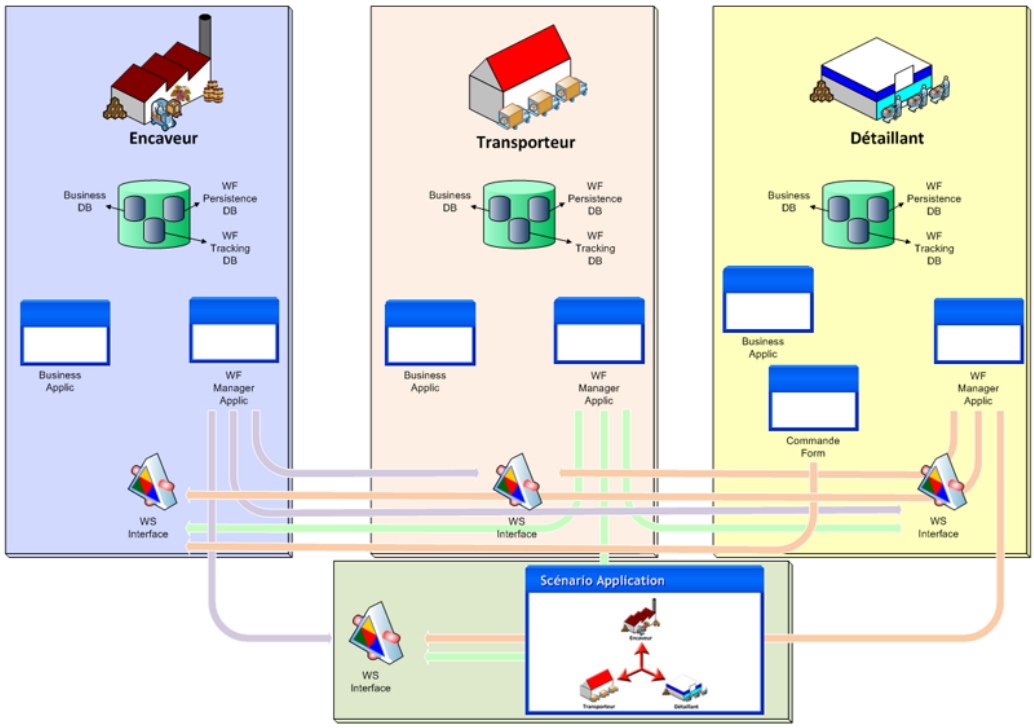
Architecture Nice to Have



Intercommunication Must Have



Intercommunication Nice to Have

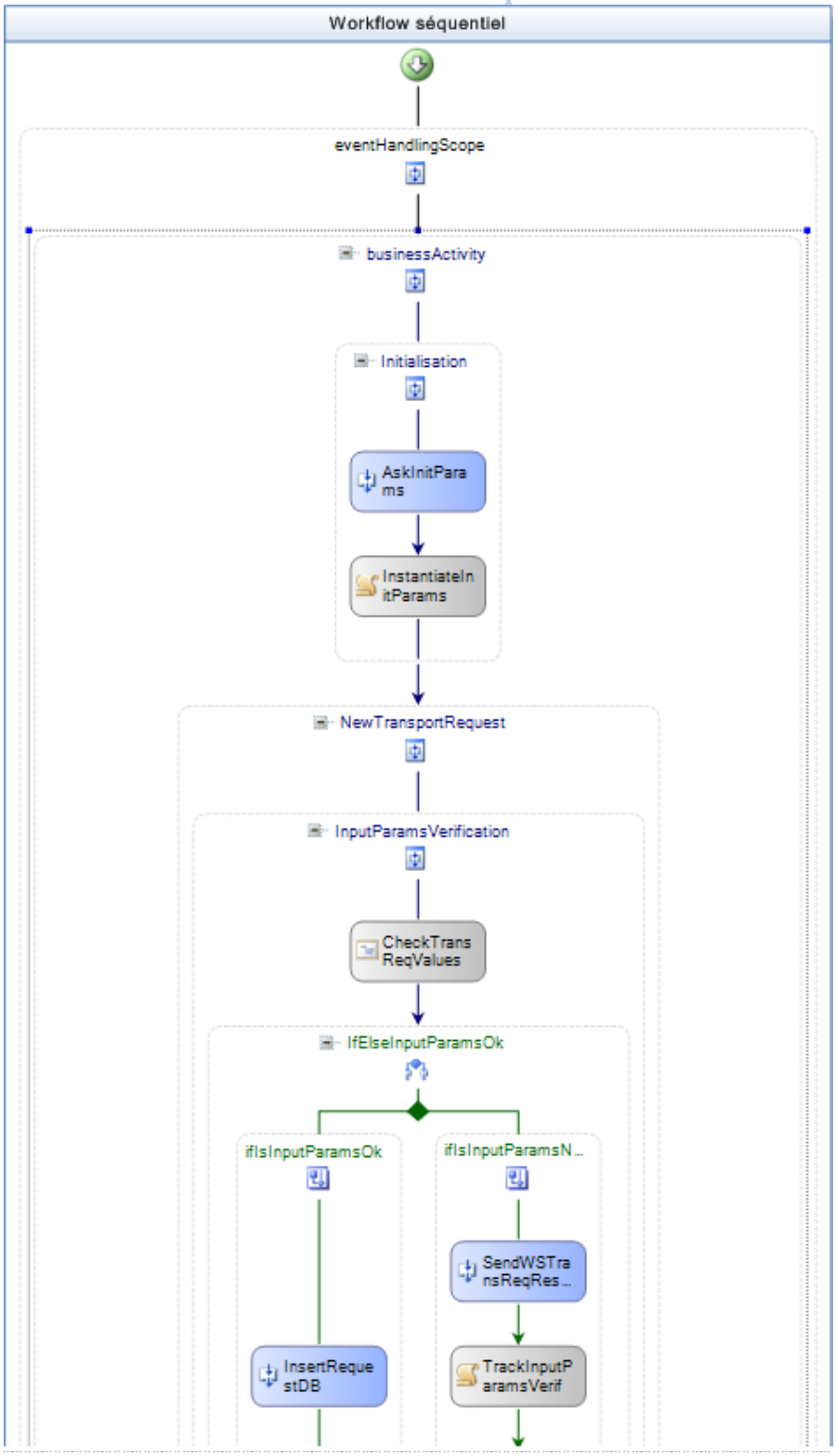


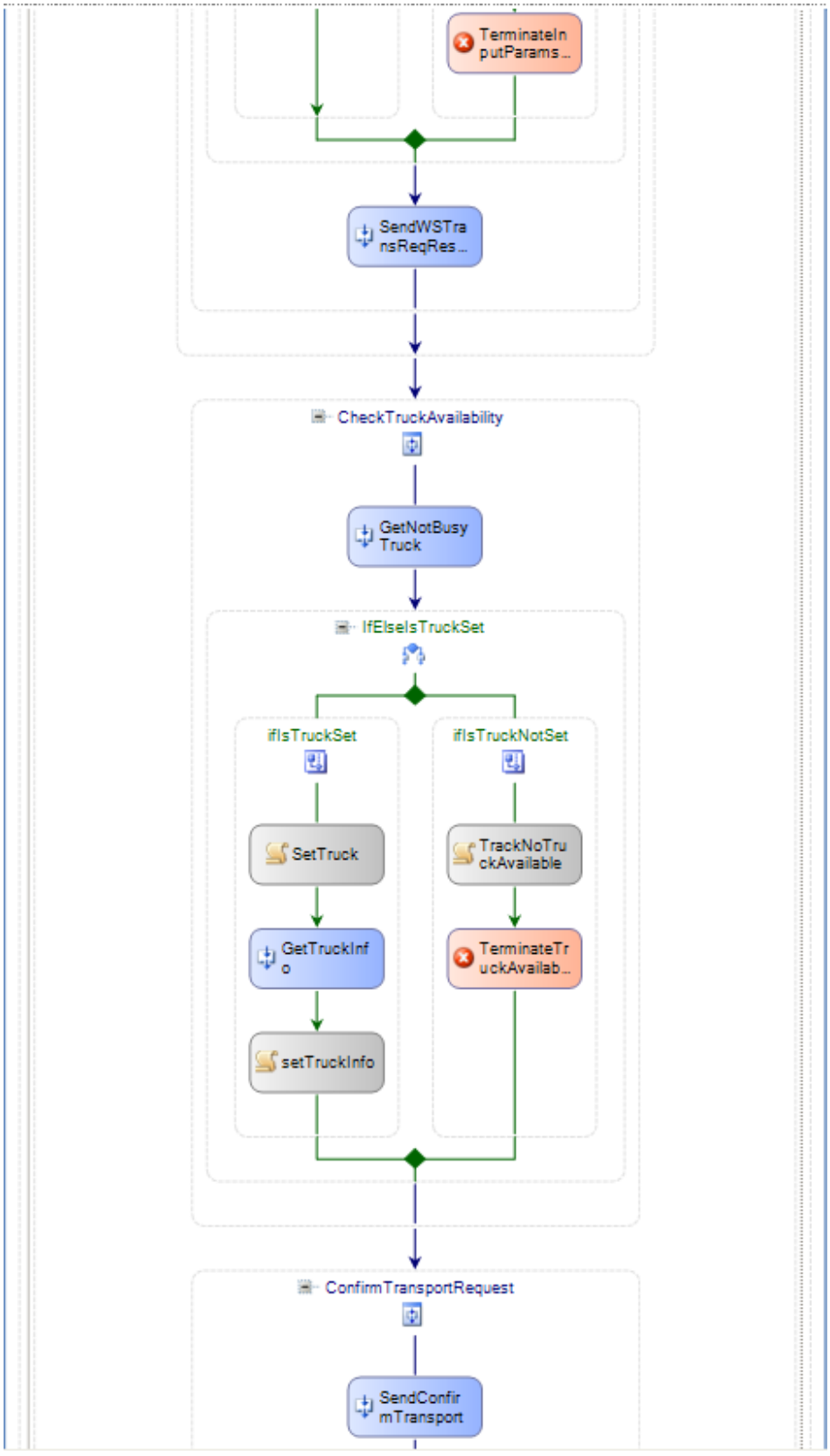
Diagrammes des processus

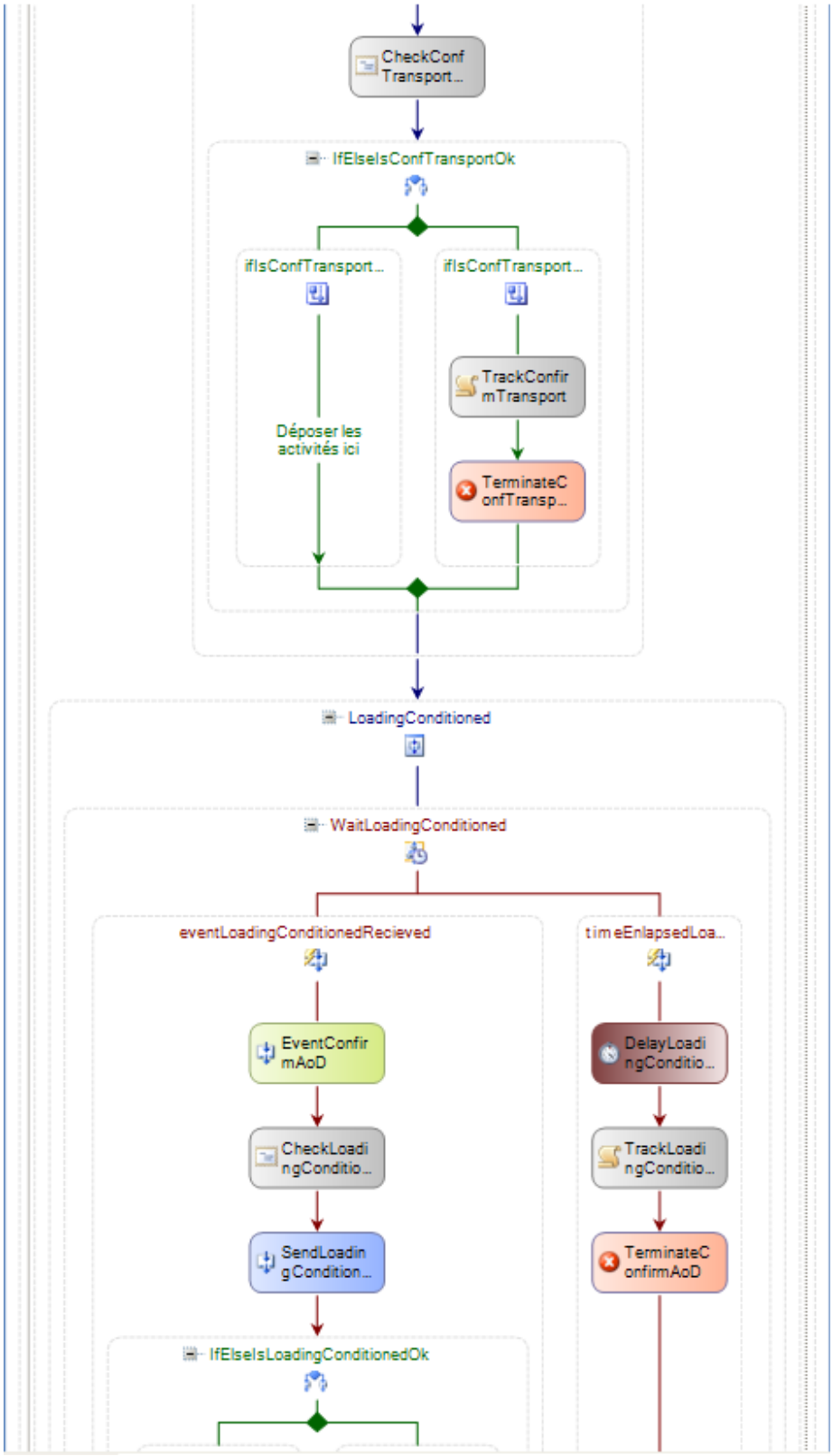
Description :

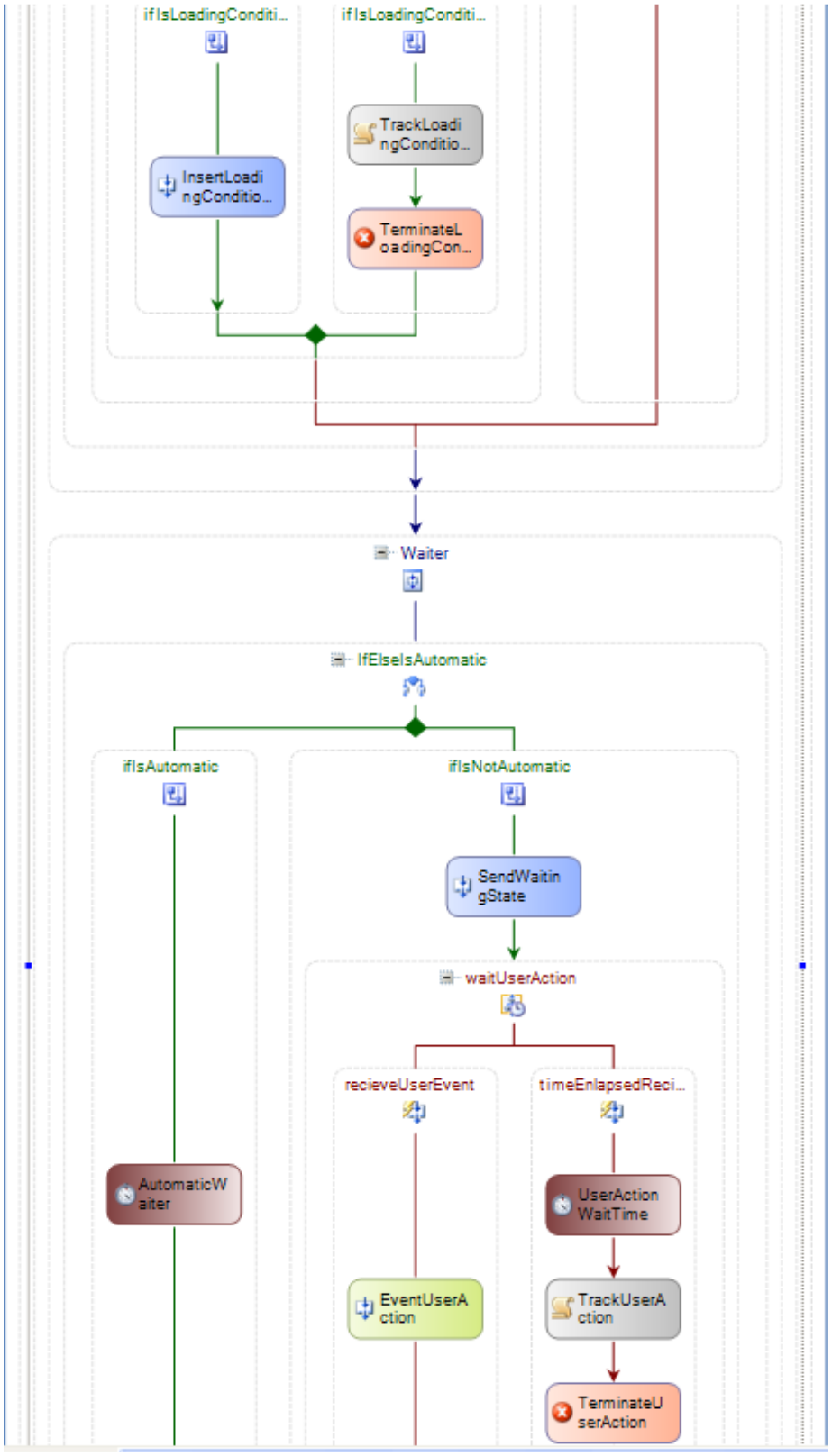
Processus de la logique métier de l'encaveur
Processus de la logique métier du transporteur
Processus de la logique métier du détaillant

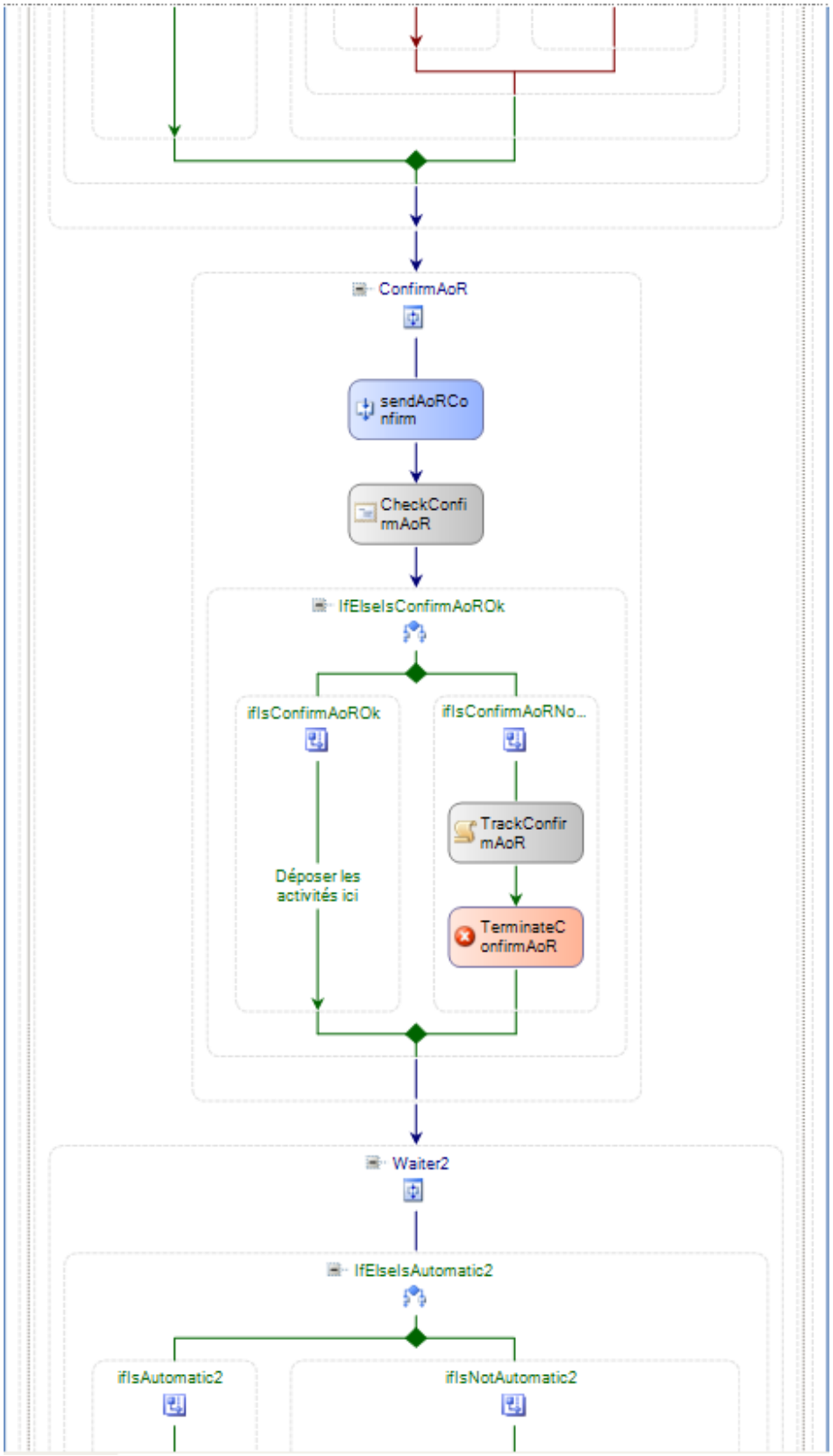
Processus du transporteur

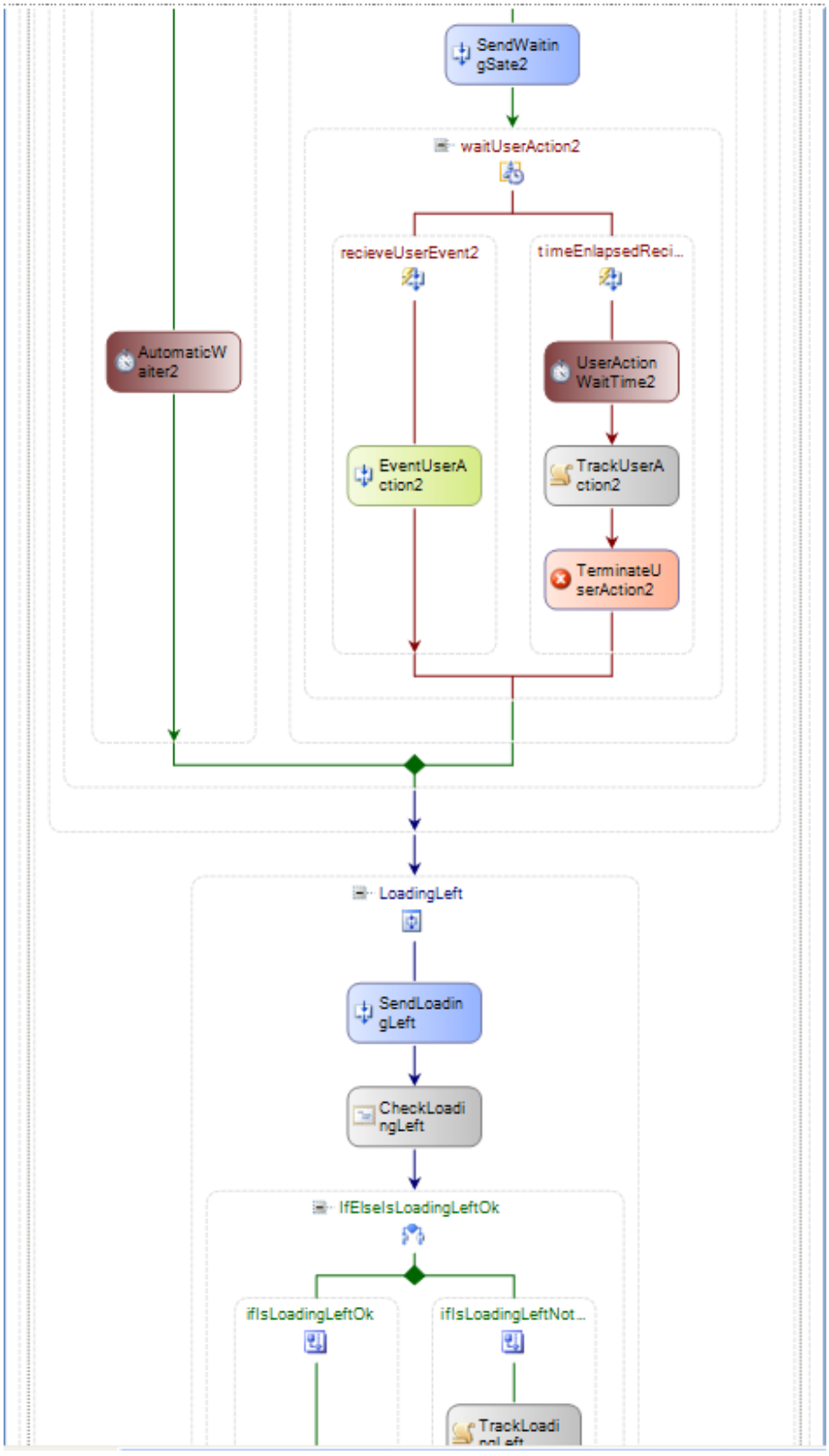


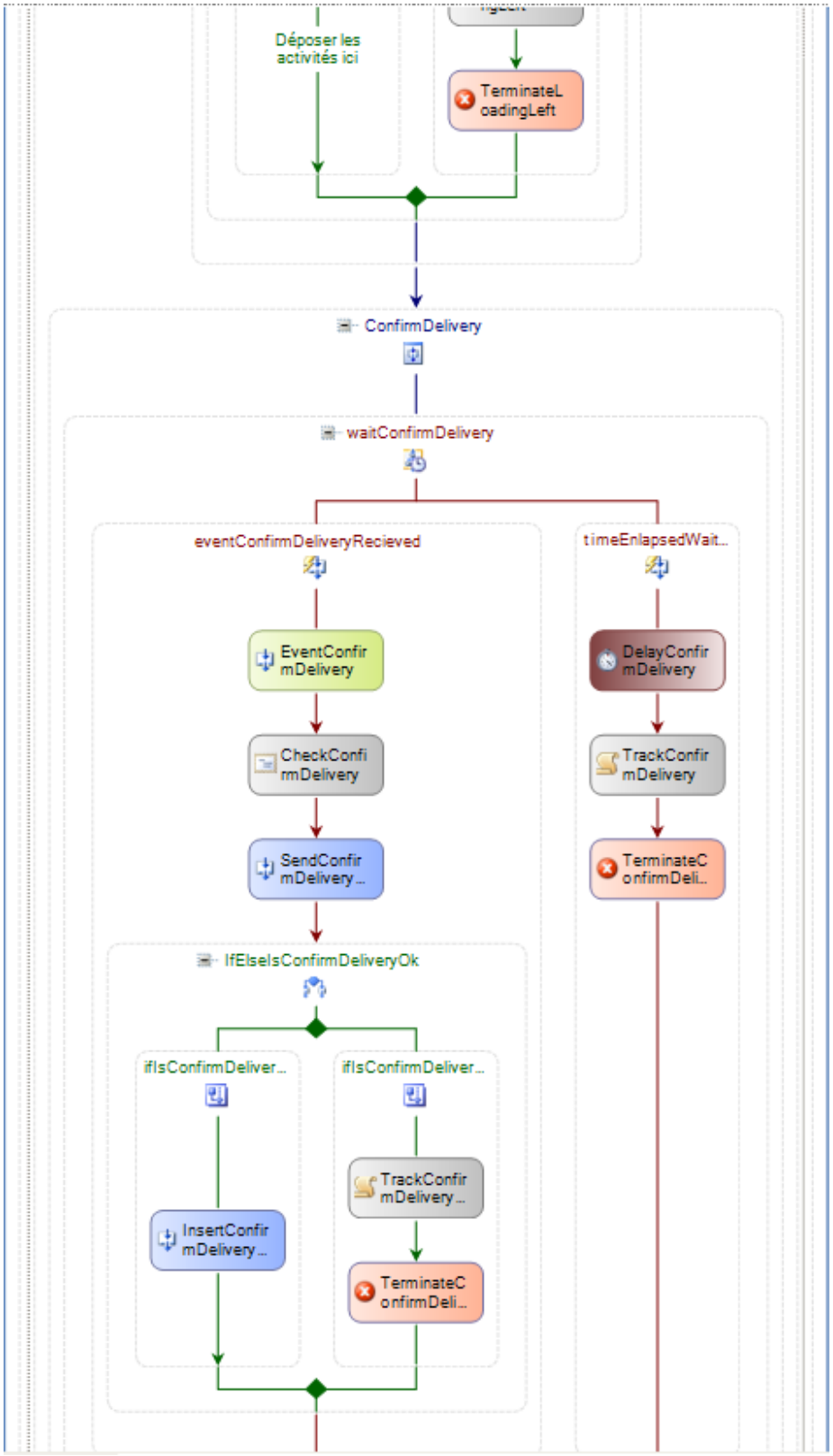

TerminateInputParams...
SendWSTransReqRes...
CheckTruckAvailability
GetNotBusyTruck
IfElseIsTruckSet
ifIsTruckSet
ifIsTruckNotSet
SetTruck
TrackNoTruckAvailable
GetTruckInfo
TerminateTruckAvailab...
setTruckInfo
ConfirmTransportRequest
SendConfirmTransport

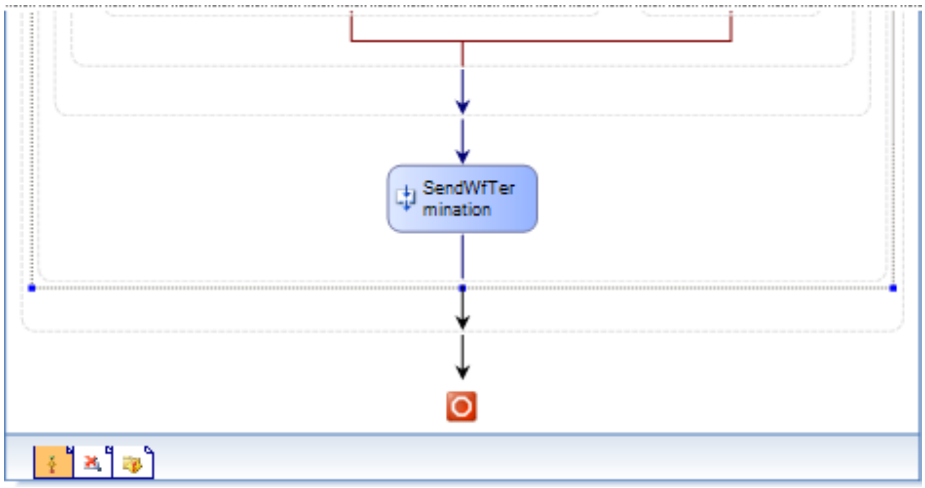

CheckConf Transport...
IfElseIsConfTransportOk
ifIsConfTransport...
ifIsConfTransport...
Déposer les activités ici
TrackConfir mTransport
TerminateC onfTransp...
LoadingConditioned
WaitLoadingConditioned
eventLoadingConditionedRecieved
timeEnlapsedLoa...
EventConfir mAoD
DelayLoadi ngConditio...
CheckLoadi ngConditio...
TrackLoadi ngConditio...
SendLoadin gCondition...
TerminateC onfirmAoD
IfElseIsLoadingConditionedOk


ifIsLoadingConditi...
ifIsLoadingConditi...
InsertLoadingConditio...
TrackLoadingConditio...
TerminateLoadingCon...
Waiter
IfElseIsAutomatic
ifIsAutomatic
ifIsNotAutomatic
SendWaitingState
waitUserAction
recieveUserEvent
timeEnlapsedReci...
AutomaticWaiter
UserActionWaitTime
EventUserAction
TrackUserAction
TerminateUserAction

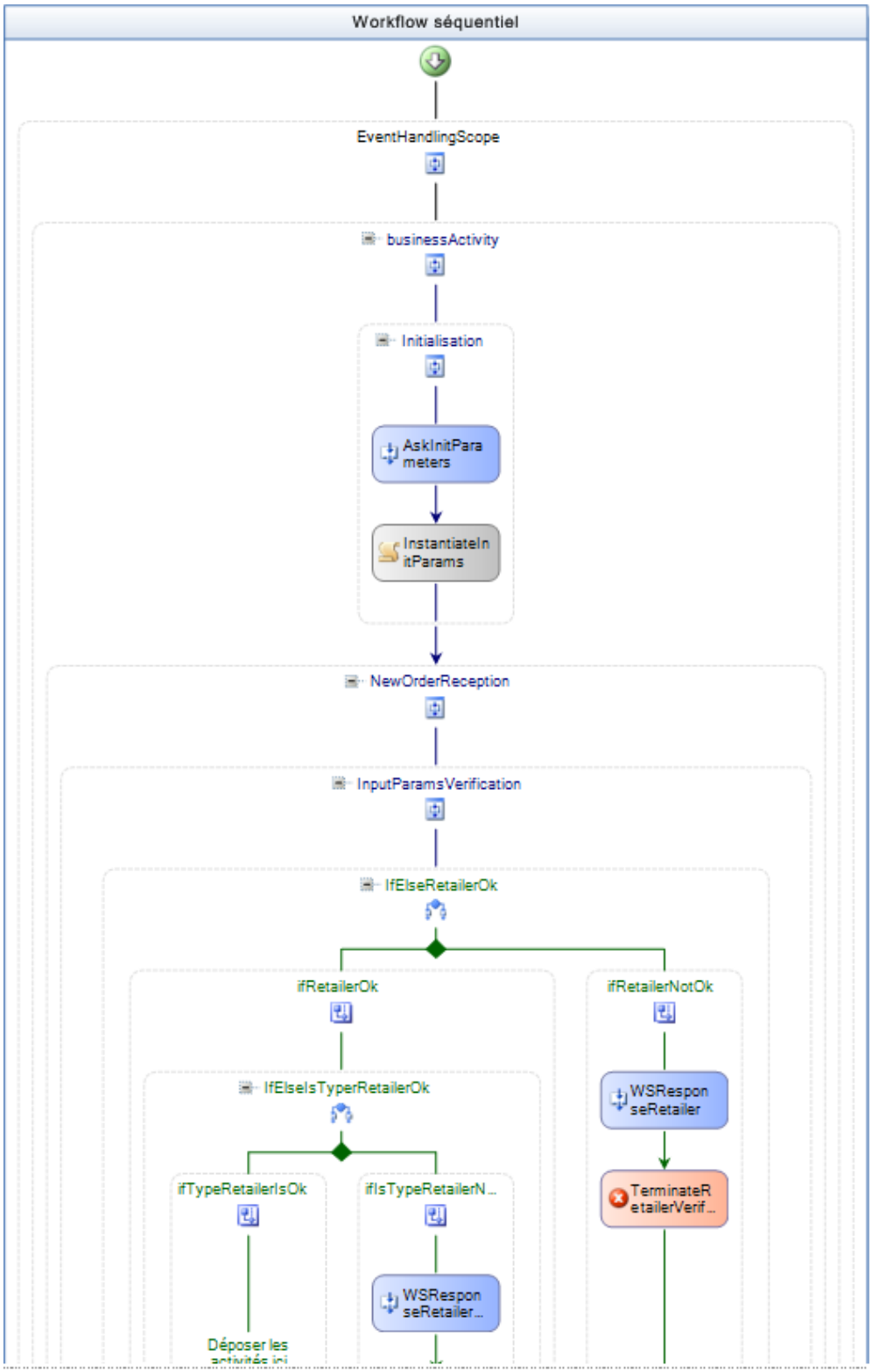

ConfirmAoR
sendAoRConfirm
CheckConfirmAoR
IfElseIsConfirmAoROk
ifIsConfirmAoROk
ifIsConfirmAoRNo...
Déposer les activités ici
TrackConfirmAoR
TerminateConfirmAoR
Waiter2
IfElseIsAutomatic2
ifIsAutomatic2
ifIsNotAutomatic2

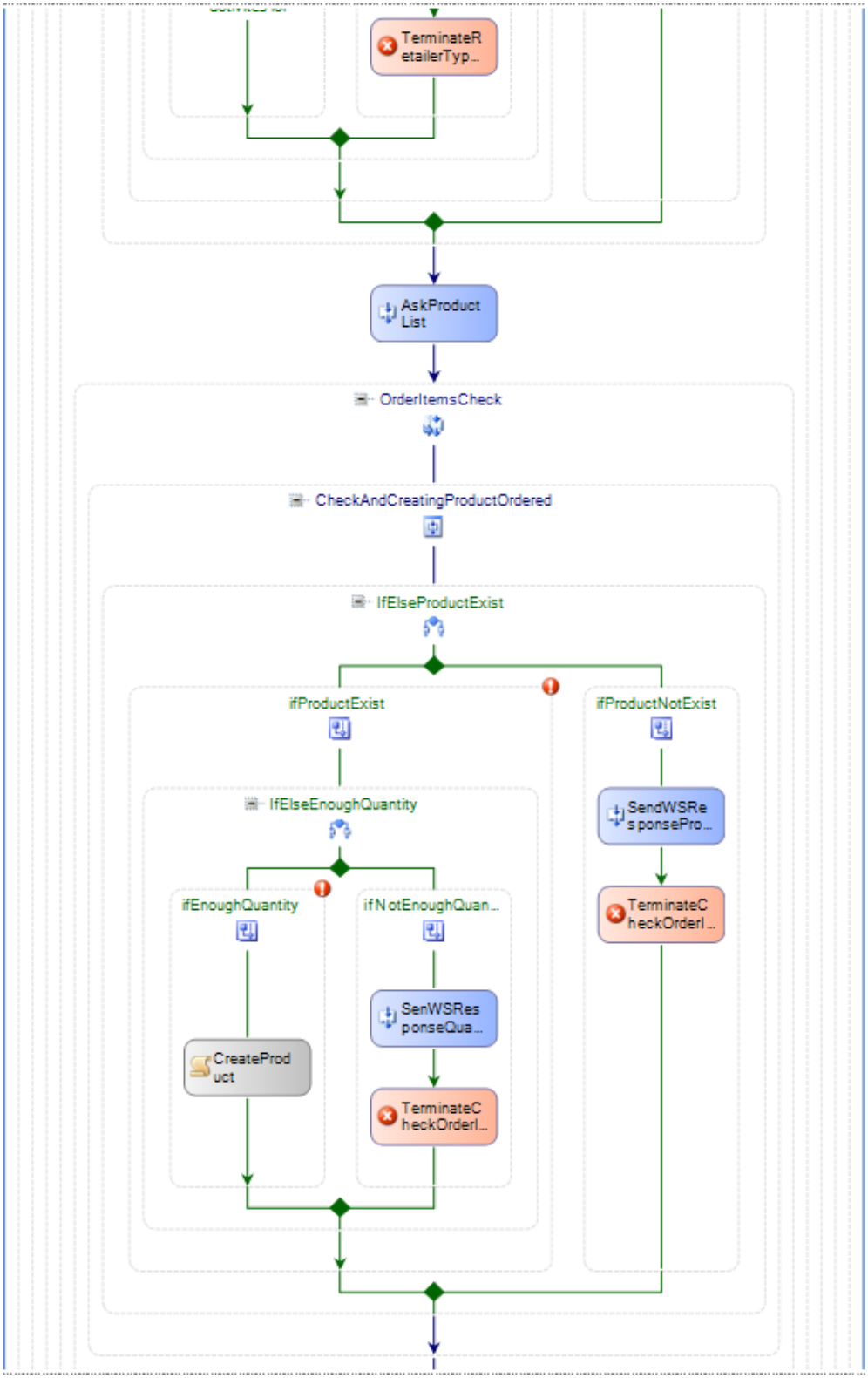

SendWaitingSate2
waitUserAction2
recieveUserEvent2
timeEnlapsedReci...
AutomaticWaiter2
UserActionWaitTime2
EventUserAction2
TrackUserAction2
TerminateUserAction2
LoadingLeft
SendLoadingLeft
CheckLoadingLeft
IfElseIsLoadingLeftOk
ifIsLoadingLeftOk
ifIsLoadingLeftNot...
TrackLoadingLeft

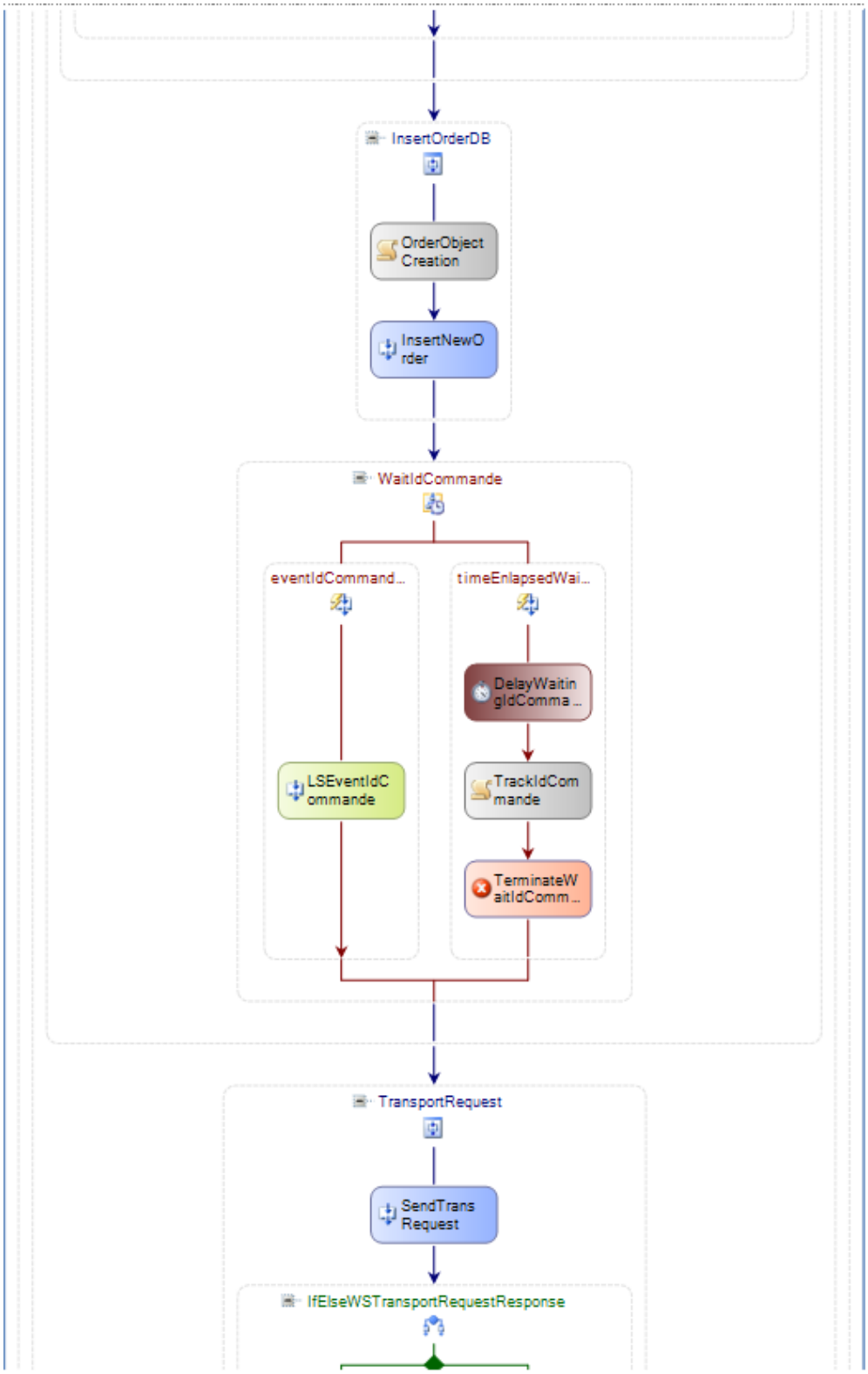

Déposer les activités ici
TerminateLoadingLeft
ConfirmDelivery
waitConfirmDelivery
eventConfirmDeliveryRecieved
timeEnlapsedWait...
EventConfirmDelivery
DelayConfirmDelivery
CheckConfirmDelivery
TrackConfirmDelivery
SendConfirmDelivery...
TerminateConfirmDeli...
IfElseIsConfirmDeliveryOk
ifIsConfirmDeliver...
ifIsConfirmDeliver...
TrackConfirmDelivery...
InsertConfirmDelivery...
TerminateConfirmDeli...

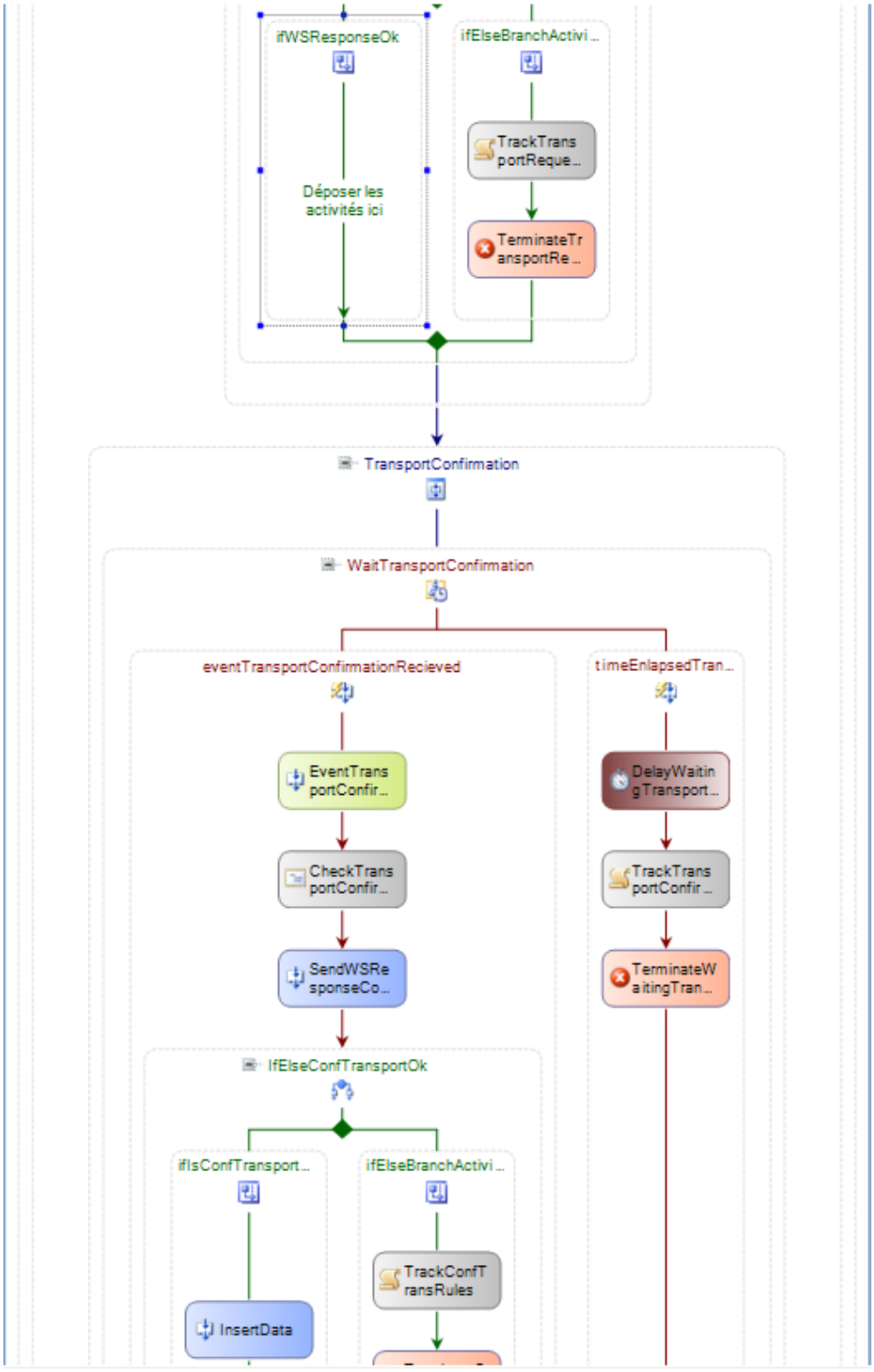

SendWfTer
mination

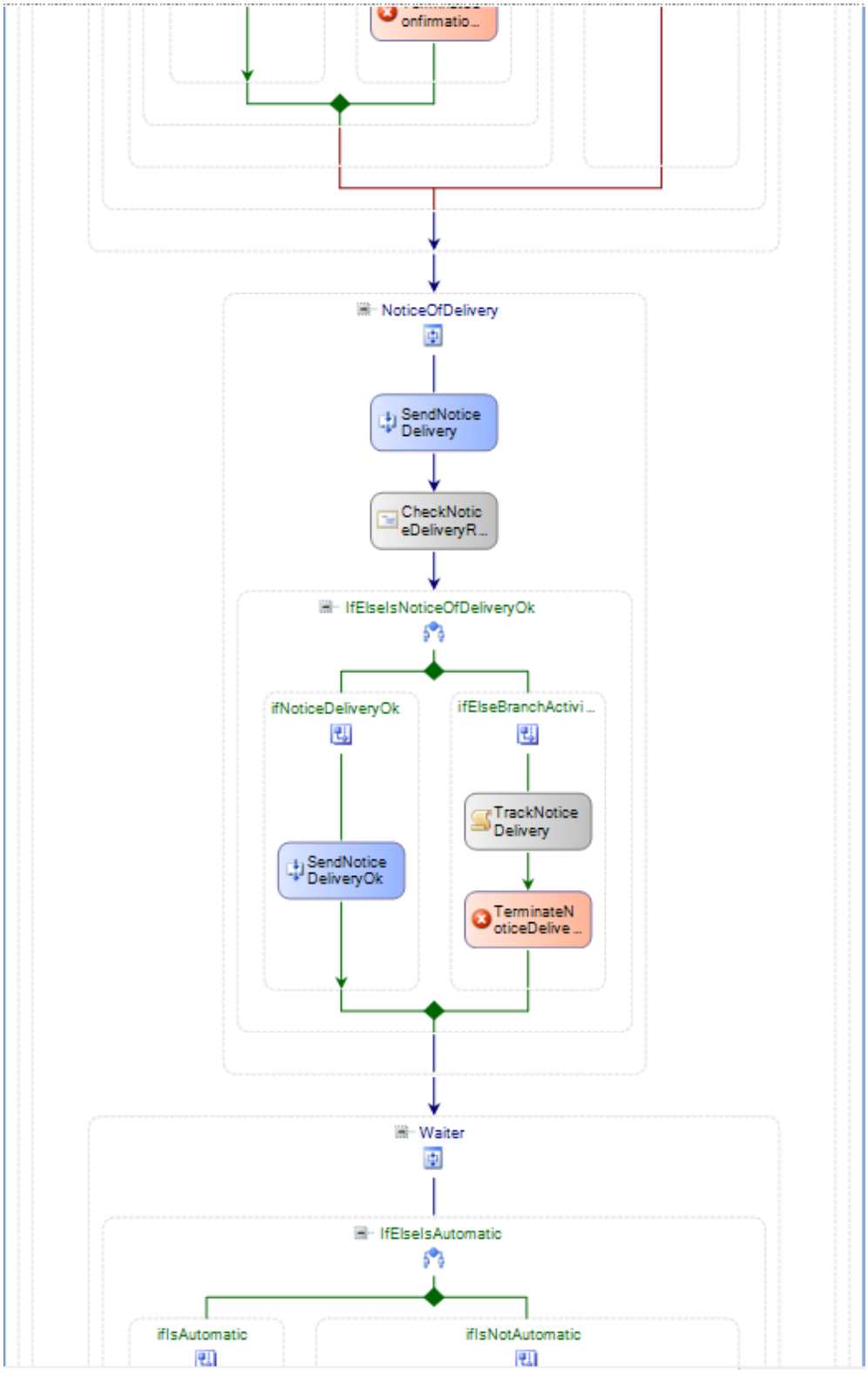
Processus de l'encaveur

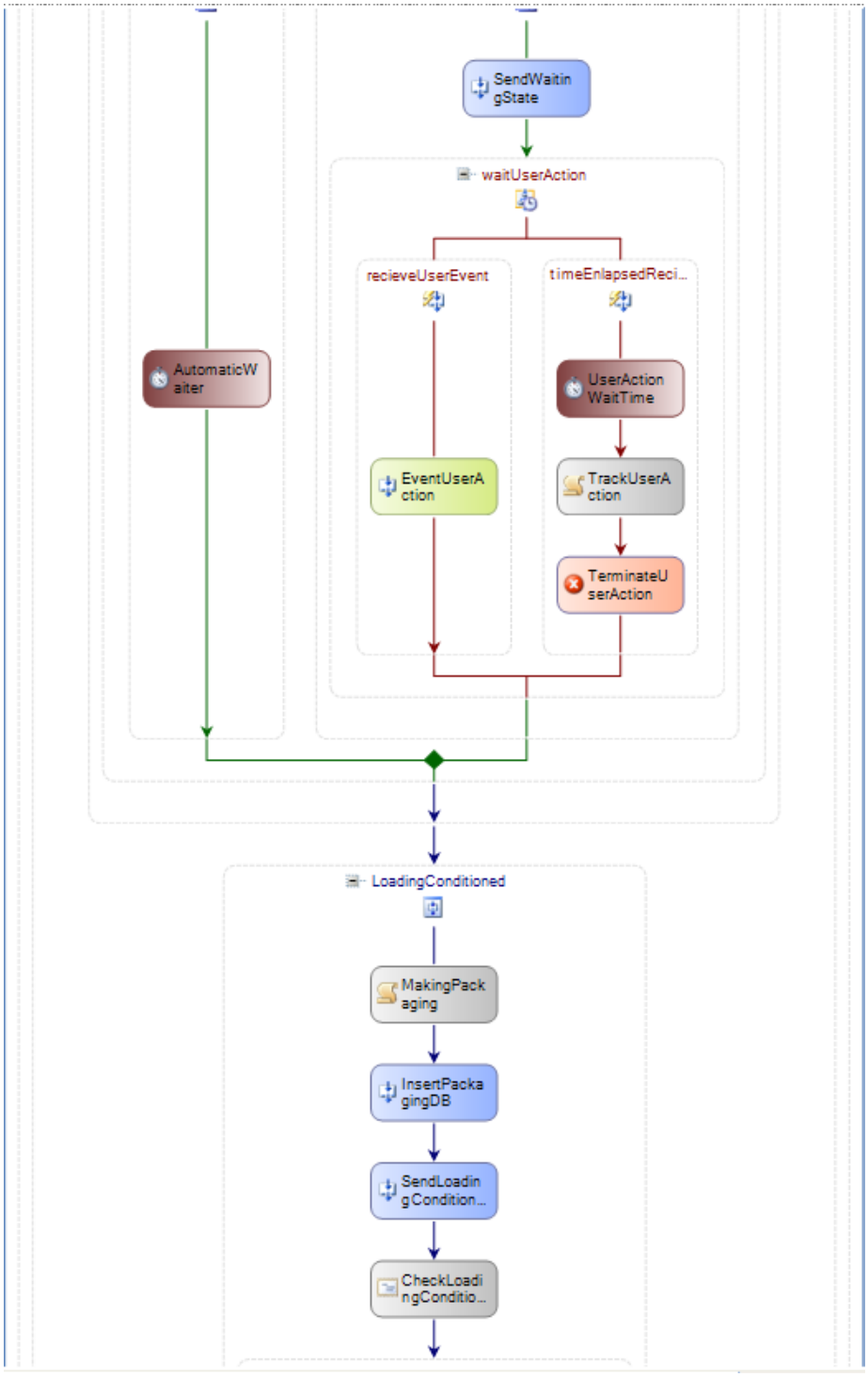


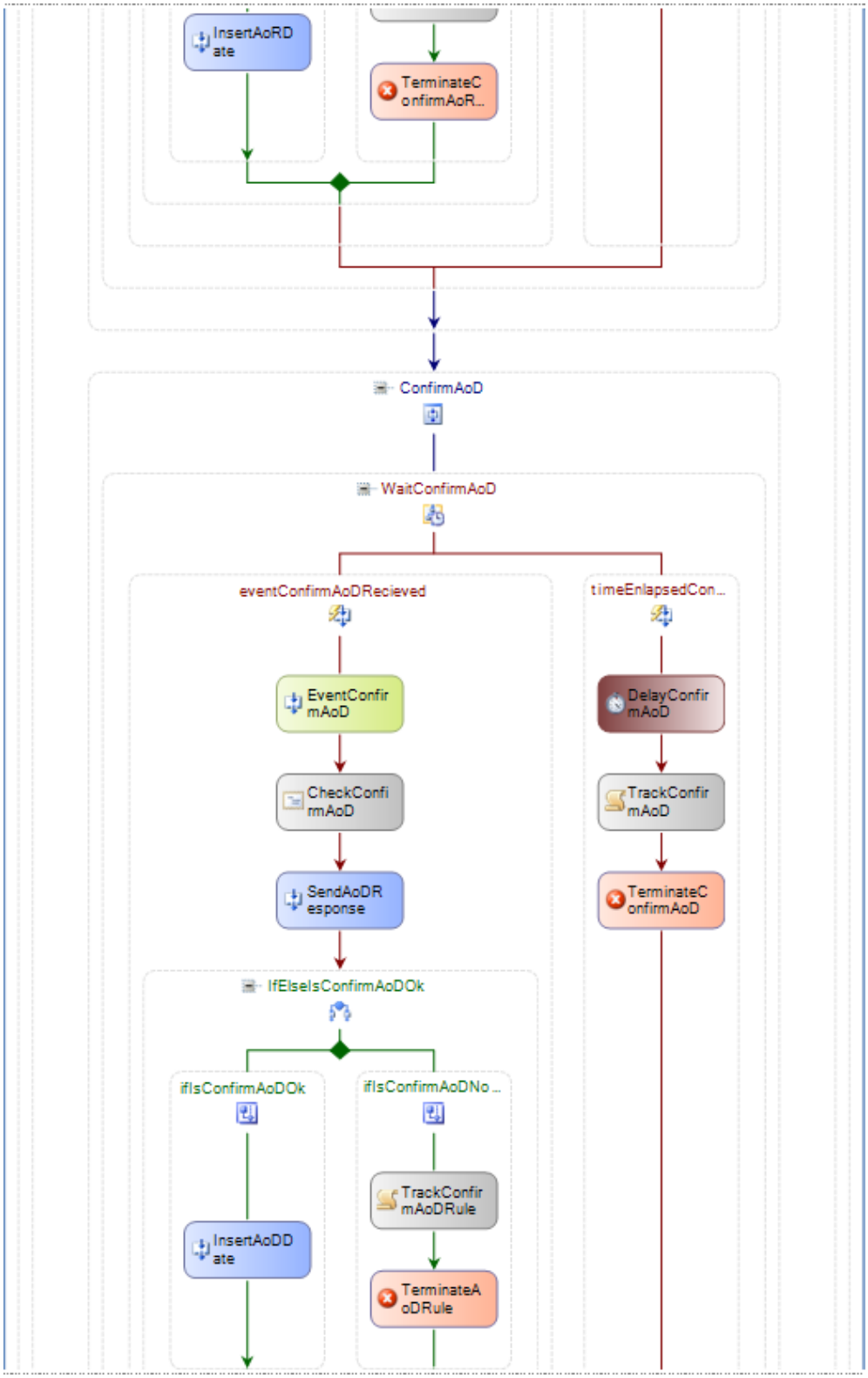

TerminateRetailerTyp...
AskProductList
OrderItemsCheck
CheckAndCreatingProductOrdered
IfElseProductExist
ifProductExist
ifProductNotExist
IfElseEnoughQuantity
SendWSResponsePro...
ifEnoughQuantity
ifNotEnoughQuan...
TerminateCheckOrderI...
SenWSResponseQua...
CreateProduct
TerminateCheckOrderI...

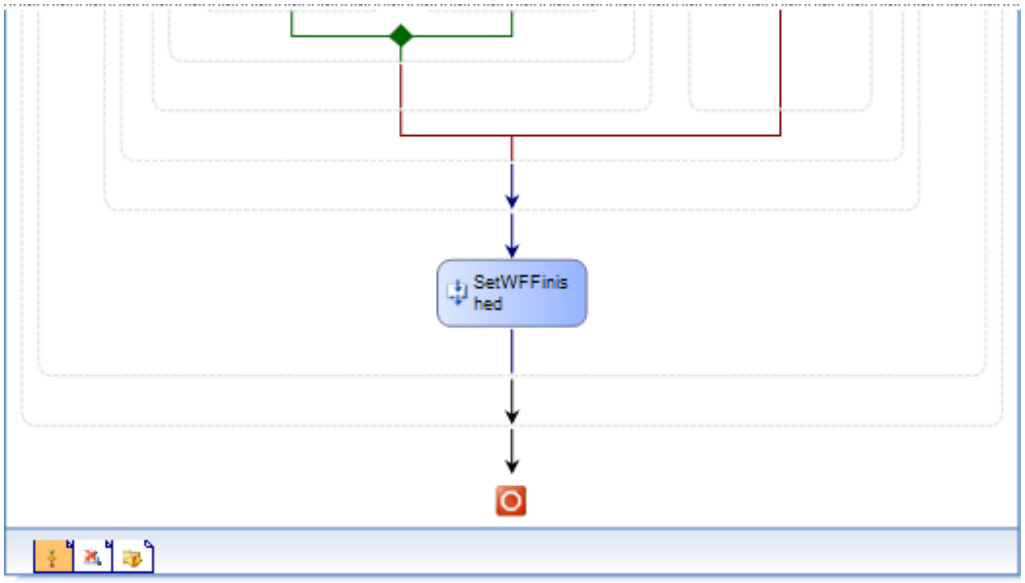

InsertOrderDB
OrderObject Creation
InsertNewO rder
WaitIdCommande
eventIdCommand...
timeEnlapsedWai...
DelayWaitin gIdComma...
LSEventIdC ommande
TrackIdCom mande
TerminateW aitIdComm...
TransportRequest
SendTrans Request
IfElseWSTransportRequestResponse


ifWSResponseOk
Déposer les activités ici
ifElseBranchActivi...
TrackTransportReque...
TerminateTransportRe...
TransportConfirmation
WaitTransportConfirmation
eventTransportConfirmationRecieved
EventTransportConfir...
CheckTransportConfir...
SendWSResponseCo...
timeEnlapsedTran...
DelayWaitingTransport...
TrackTransportConfir...
TerminateWaitingTran...
IfElseConfTransportOk
ifIsConfTransport...
InsertData
ifElseBranchActivi...
TrackConfTransRules

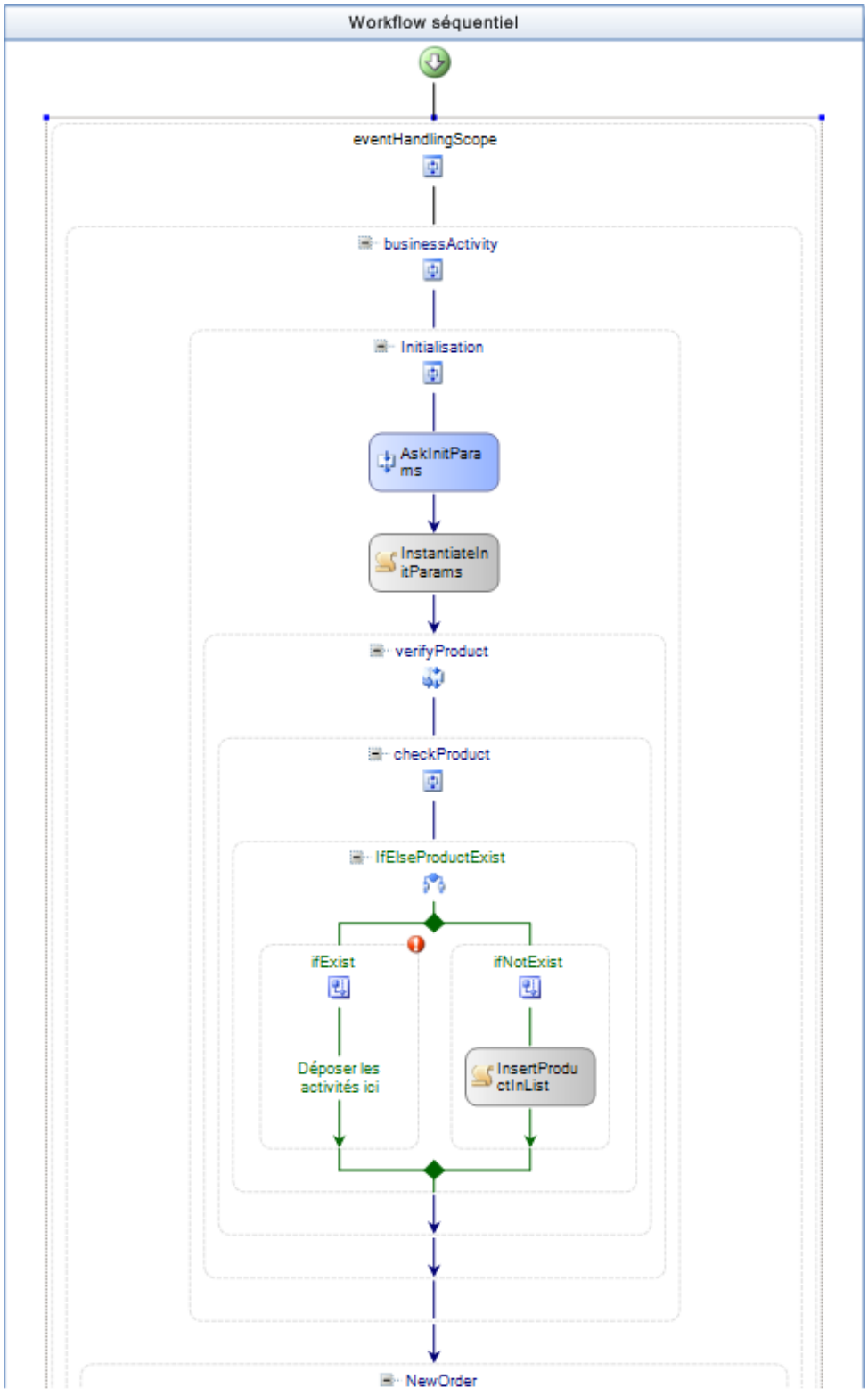

onfirmatio...
NoticeOfDelivery
SendNotice Delivery
CheckNotic eDeliveryR...
IfElseIsNoticeOfDeliveryOk
ifNoticeDeliveryOk
ifElseBranchActivi...
SendNotice DeliveryOk
TrackNotice Delivery
TerminateN oticeDelive...
Waiter
IfElseIsAutomatic
ifIsAutomatic
ifIsNotAutomatic

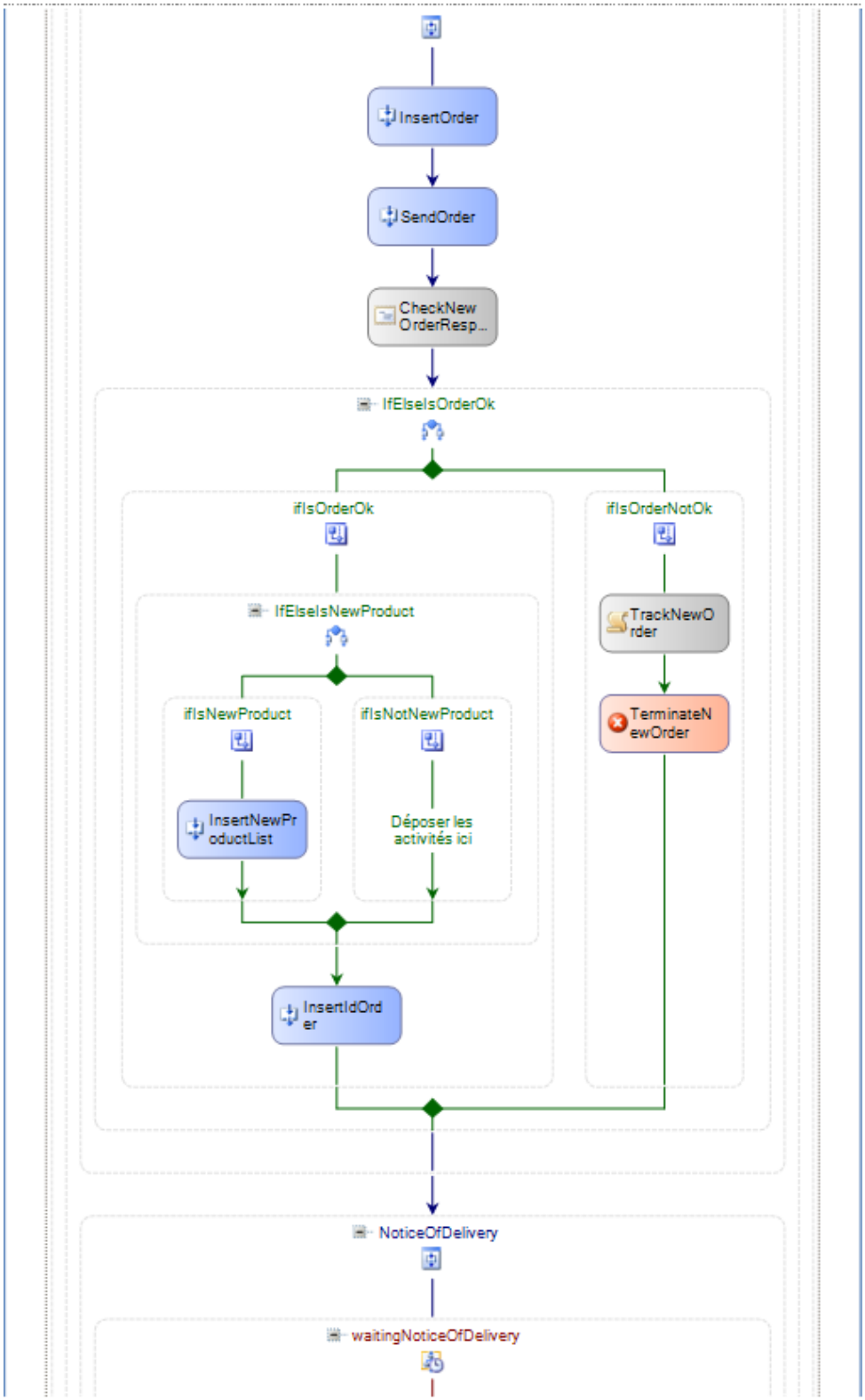

SendWaitin gState
waitUserAction
recieveUserEvent
timeEnlapsedReci...
AutomaticW aiter
UserAction WaitTime
EventUserA ction
TrackUserA ction
TerminateU serAction
LoadingConditioned
MakingPack aging
InsertPacka gingDB
SendLoadin gCondition...
CheckLoadi ngConditio...

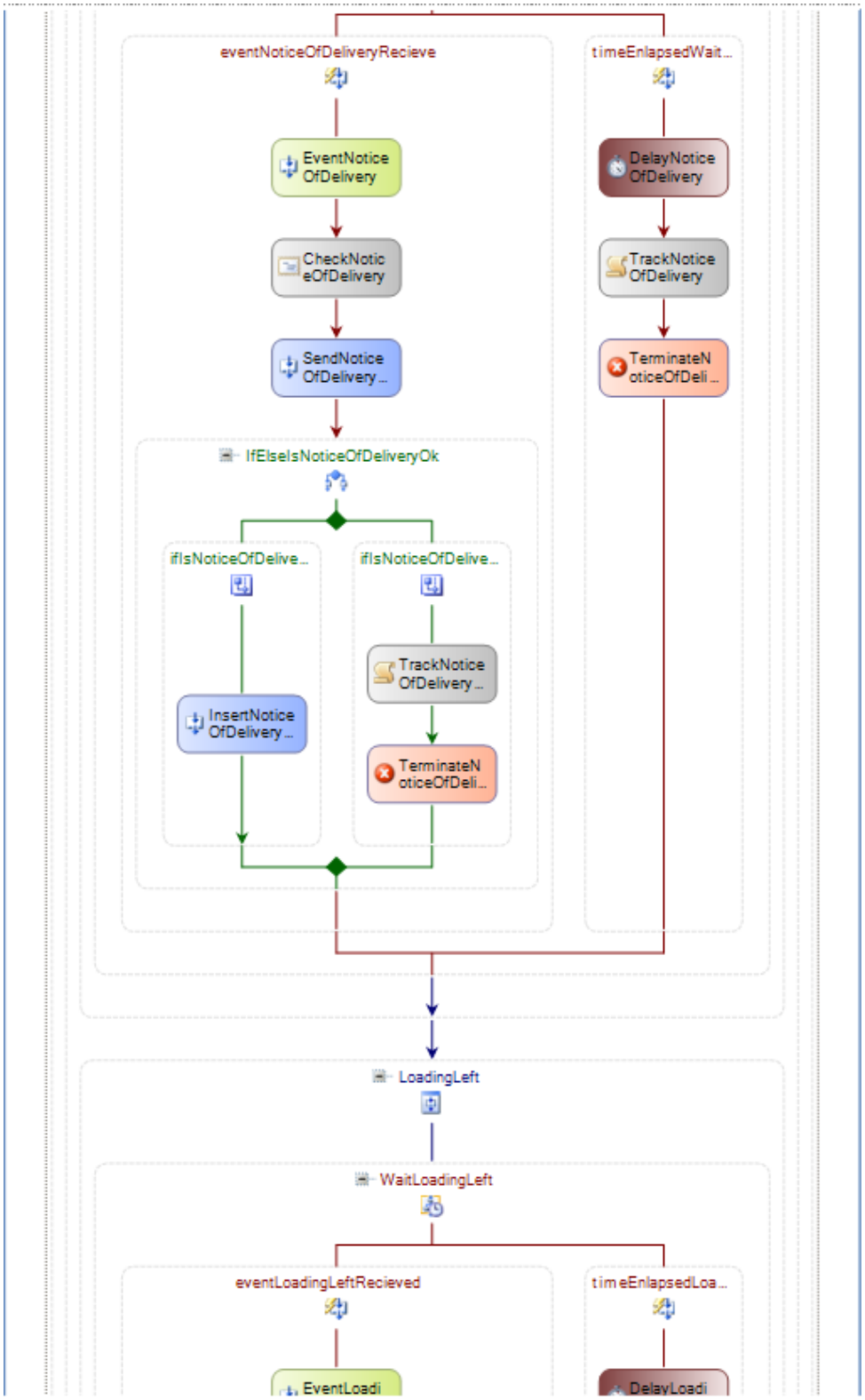

InsertAoRD ate
TerminateC onfirmAoR...
ConfirmAoD
WaitConfirmAoD
eventConfirmAoDRecieved
timeEnlapsedCon...
EventConfir mAoD
DelayConfir mAoD
CheckConfi rmAoD
TrackConfir mAoD
SendAoDR esponse
TerminateC onfirmAoD
IfElseIsConfirmAoDOk
ifIsConfirmAoDOk
ifIsConfirmAoDNo...
InsertAoDD ate
TrackConfir mAoDRule
TerminateA oDRule

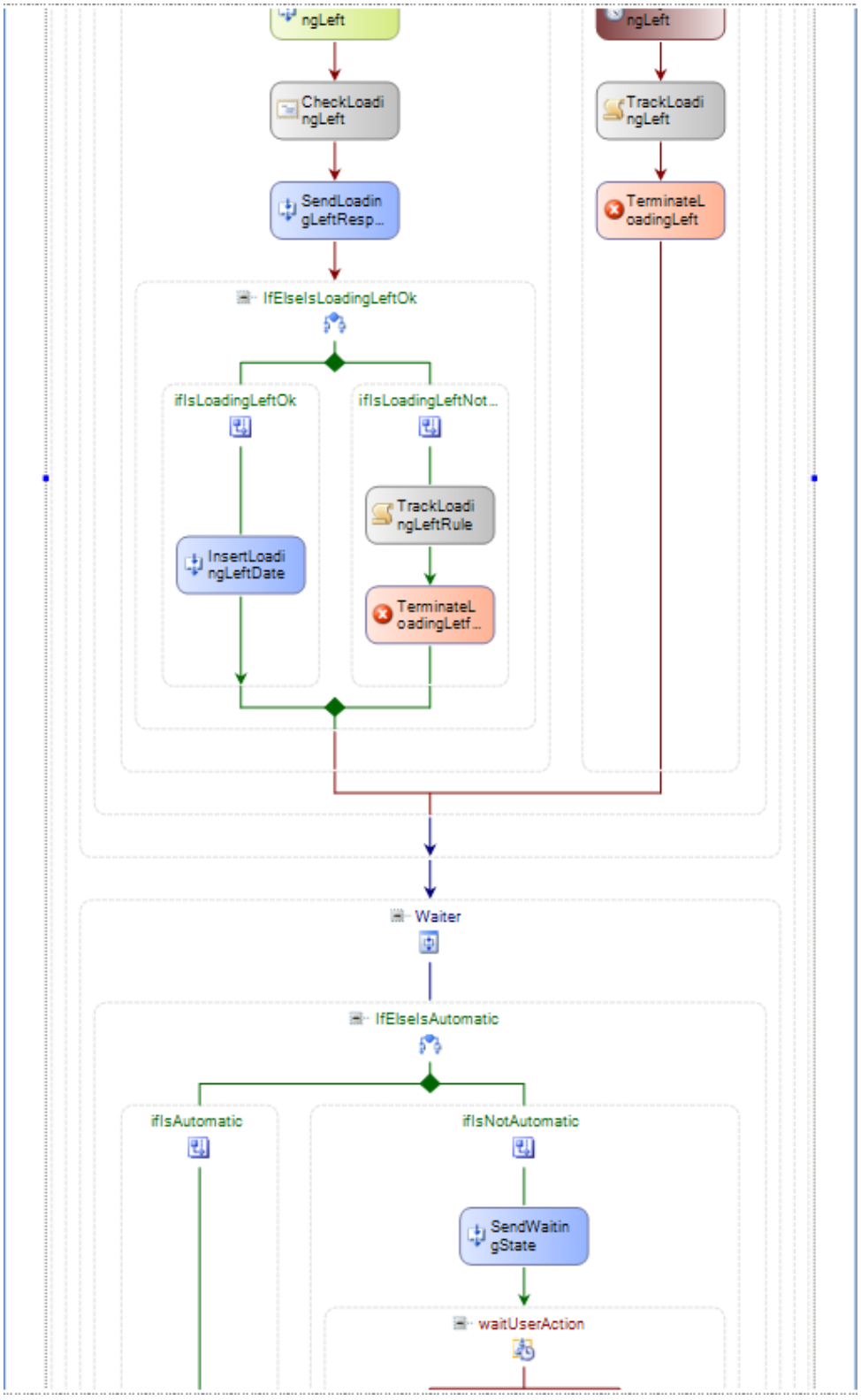

SetWFFinis
hed

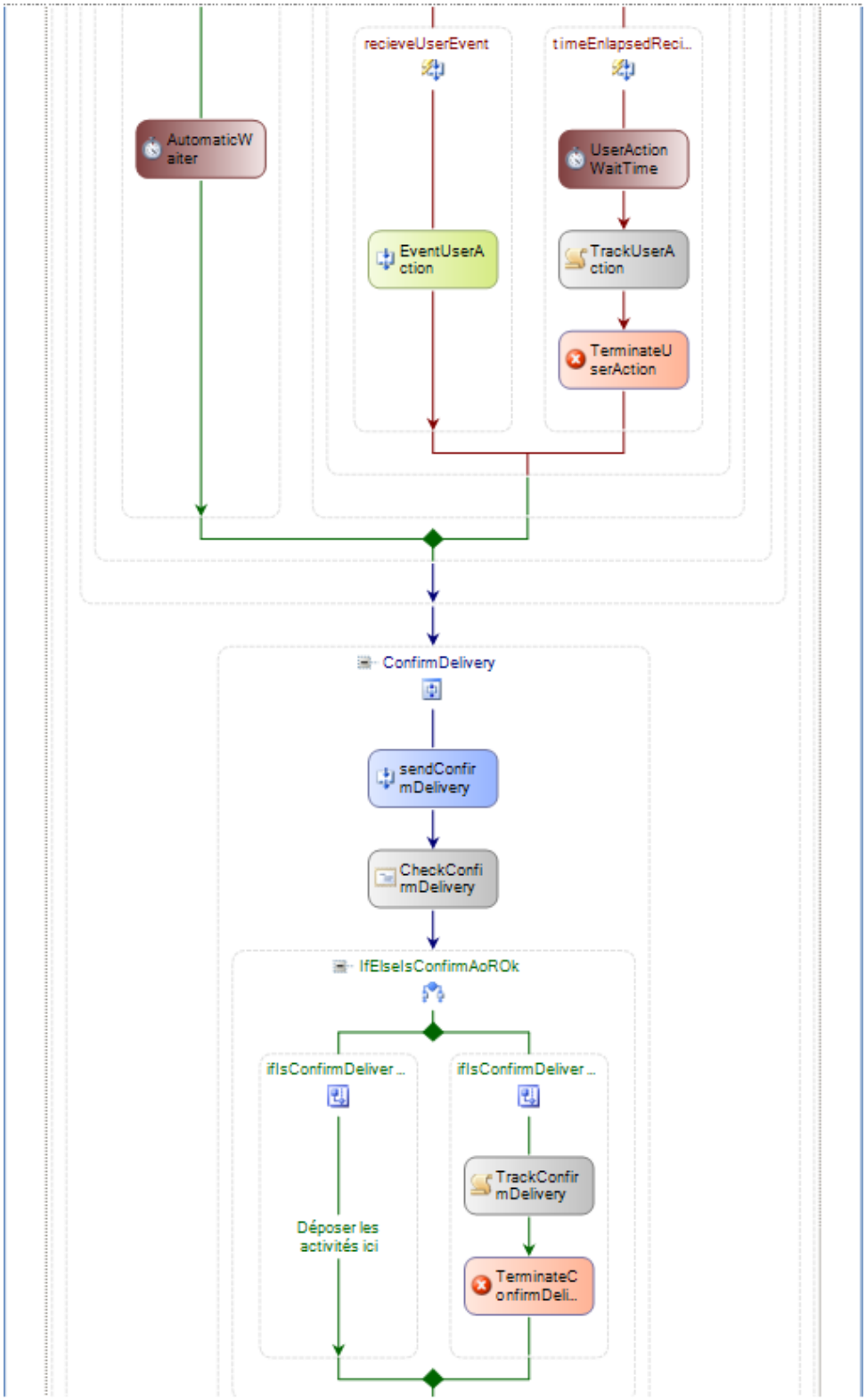
Processus du détaillant

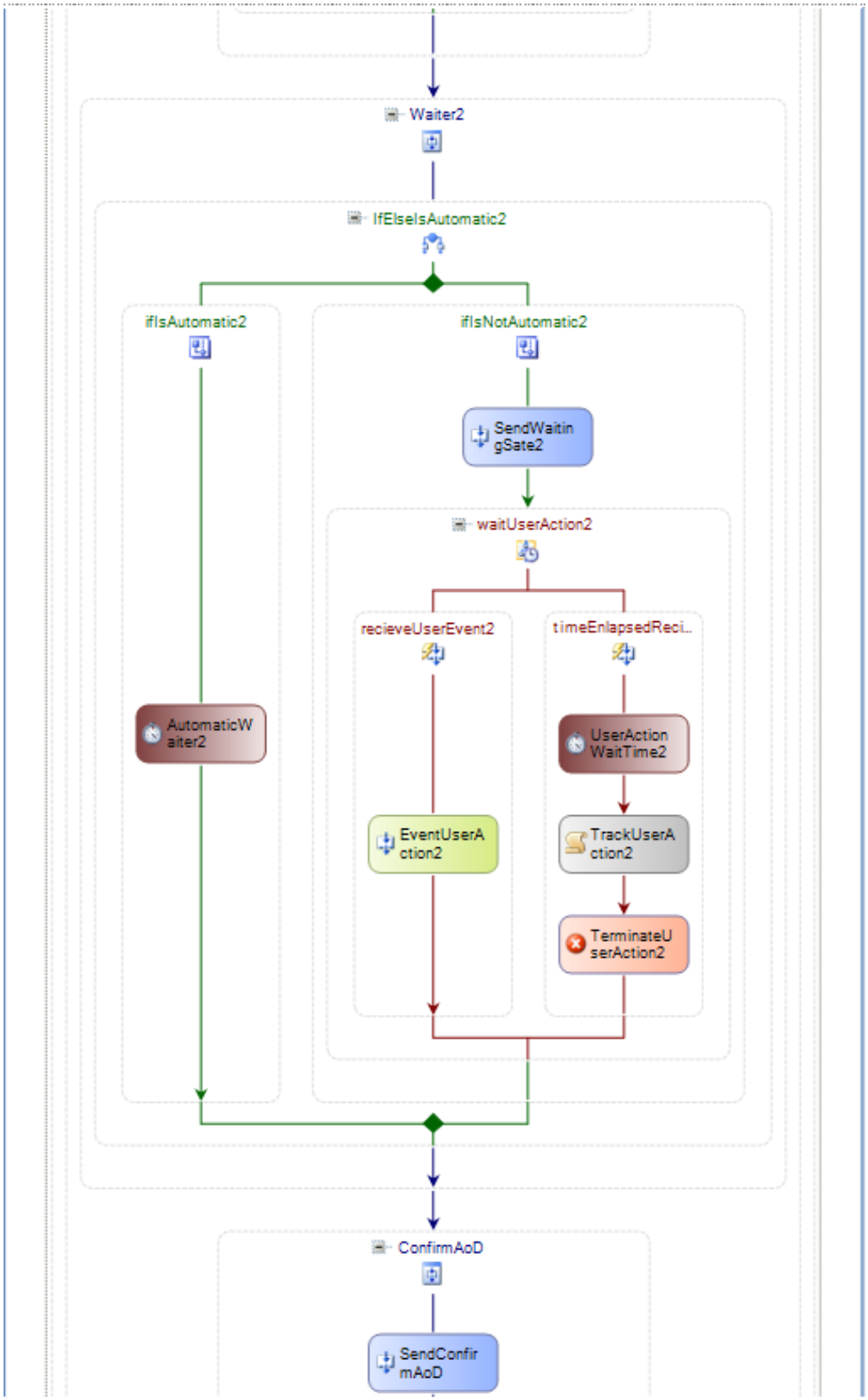


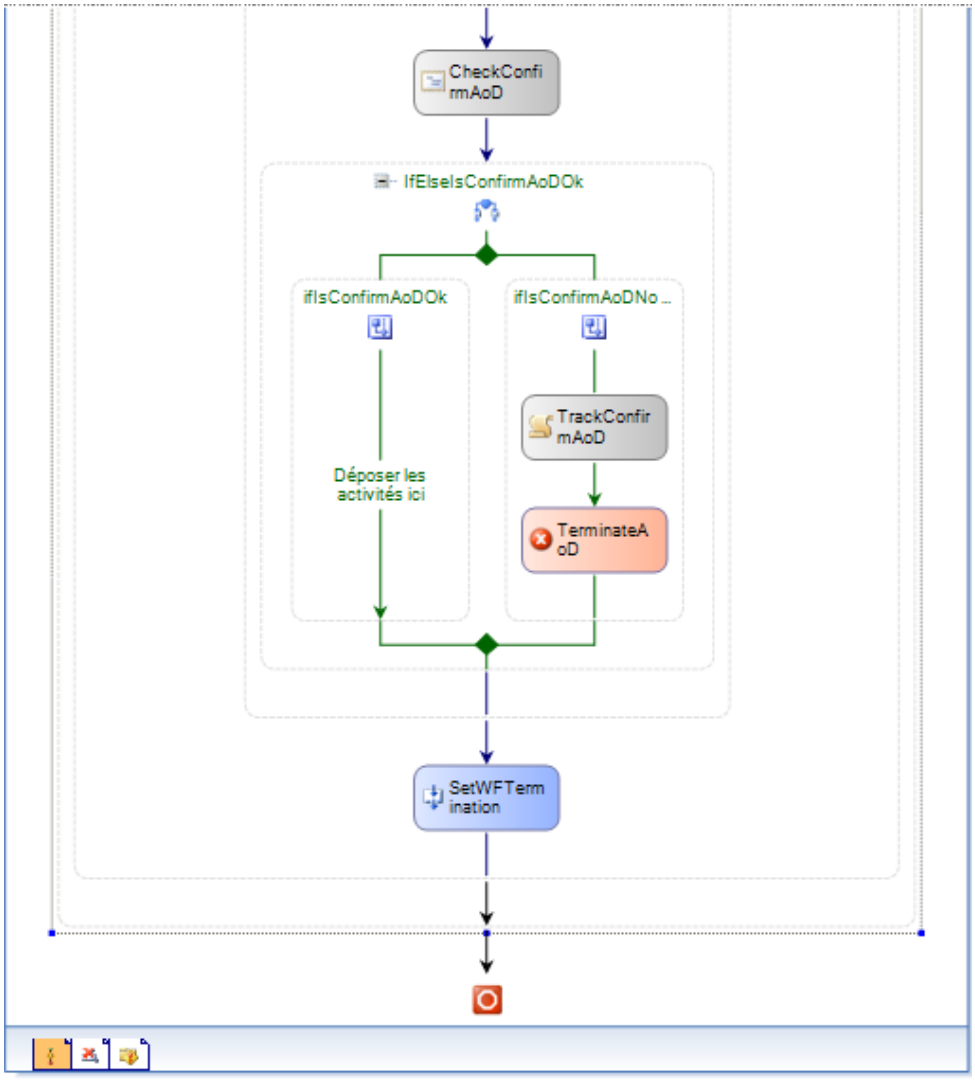

InsertOrder
SendOrder
CheckNew OrderResp...
IfElseIsOrderOk
ifIsOrderOk
ifIsOrderNotOk
IfElseIsNewProduct
TrackNewO rder
ifIsNewProduct
ifIsNotNewProduct
TerminateN ewOrder
InsertNewPr oductList
Déposer les activités ici
InsertIdOrd er
NoticeOfDelivery
waitingNoticeOfDelivery


eventNoticeOfDeliveryRecieve
timeEnlapsedWait...
EventNotice OfDelivery
DelayNotice OfDelivery
CheckNotic eOfDelivery
TrackNotice OfDelivery
SendNotice OfDelivery...
TerminateN oticeOfDeli...
IfElseIsNoticeOfDeliveryOk
ifIsNoticeOfDelive...
ifIsNoticeOfDelive...
TrackNotice OfDelivery...
InsertNotice OfDelivery...
TerminateN oticeOfDeli...
LoadingLeft
WaitLoadingLeft
eventLoadingLeftRecieved
timeEnlapsedLoa...
EventLoadi
DelayLoadi


ngLeft
CheckLoadingLeft
SendLoadingLeftResp...
IfElseIsLoadingLeftOk
ifIsLoadingLeftOk
InsertLoadingLeftDate
ifIsLoadingLeftNot...
TrackLoadingLeftRule
TerminateLoadingLetf...
ngLeft
TrackLoadingLeft
TerminateLoadingLeft
Waiter
IfElseIsAutomatic
ifIsAutomatic
ifIsNotAutomatic
SendWaitingState
waitUserAction


recieveUserEvent
timeEnlapsedReci...
AutomaticW aiter
UserAction WaitTime
EventUserA ction
TrackUserA ction
TerminateU serAction
ConfirmDelivery
sendConfir mDelivery
CheckConfi rmDelivery
IfElseIsConfirmAoROk
ifIsConfirmDeliver ...
ifIsConfirmDeliver ...
TrackConfir mDelivery
Déposer les activités ici
TerminateC onfirmDeli...


Waiter2
IfElseIsAutomatic2
ifIsAutomatic2
ifIsNotAutomatic2
SendWaitingSate2
waitUserAction2
recieveUserEvent2
timeEnlapsedReci..
AutomaticWaiter2
UserActionWaitTime2
EventUserAction2
TrackUserAction2
TerminateUserAction2
ConfirmAoD
SendConfirmAoD

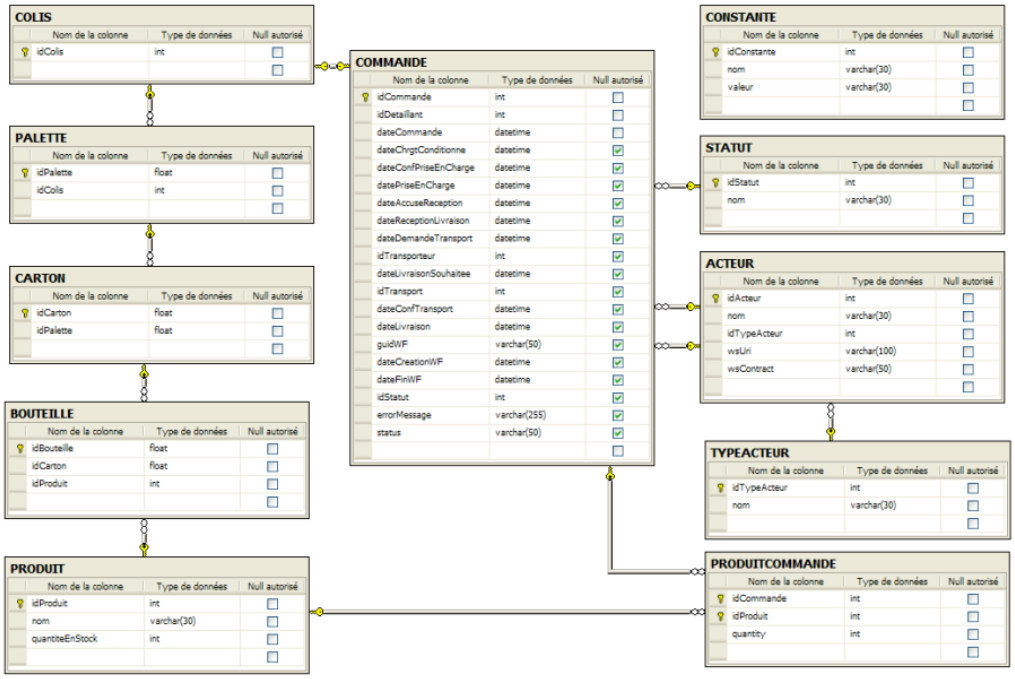

CheckConfirmAoD
IfElseIsConfirmAoDOk
ifIsConfirmAoDOk
ifIsConfirmAoDNo...
Déposer les activités ici
TrackConfirmAoD
TerminateAoD
SetWFTermination

Diagrammes des bases de données

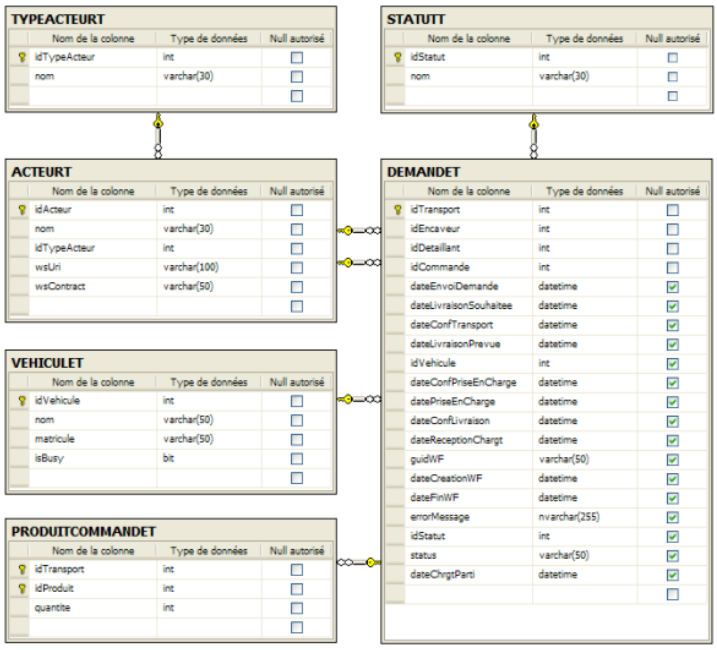
Description :

Modèle physique de la base de données métier de l'encaveur, du transporteur et du détaillant, ainsi que le modèle physique de la base de données gérant la persistance et le suivi des processus

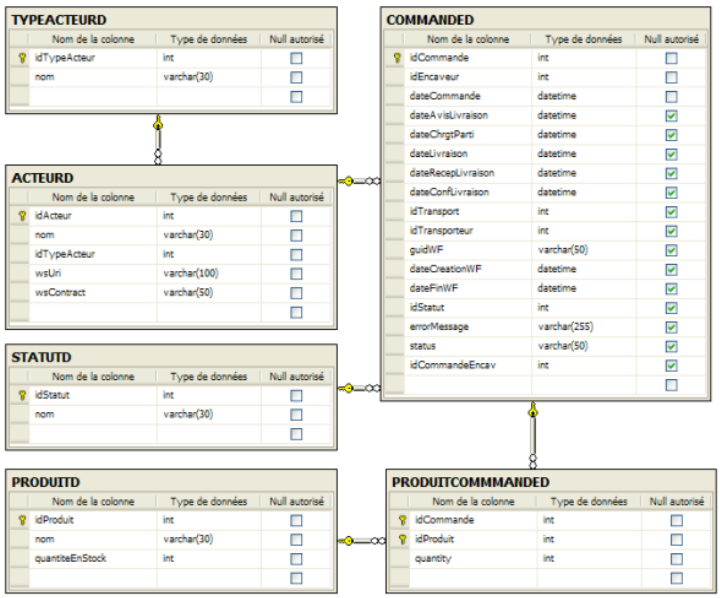
Base de données métier de l'encaveur



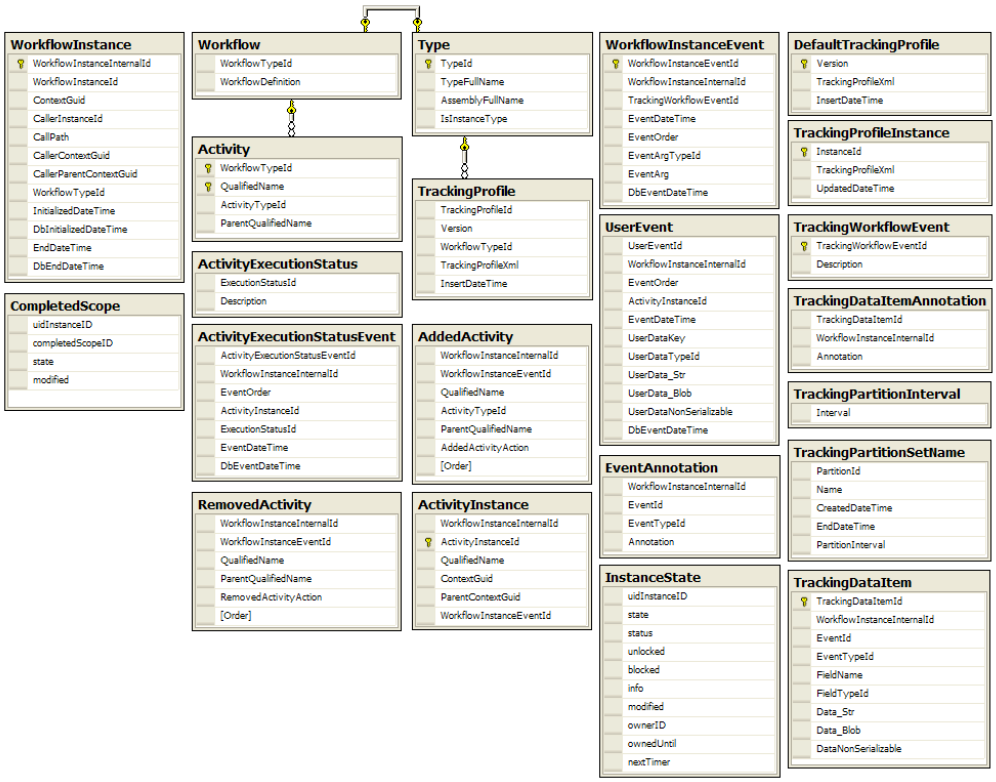
Base de données métier du transporteur



Base de données métier du détaillant



Base de données pour la gestion de la persistance et du tracking

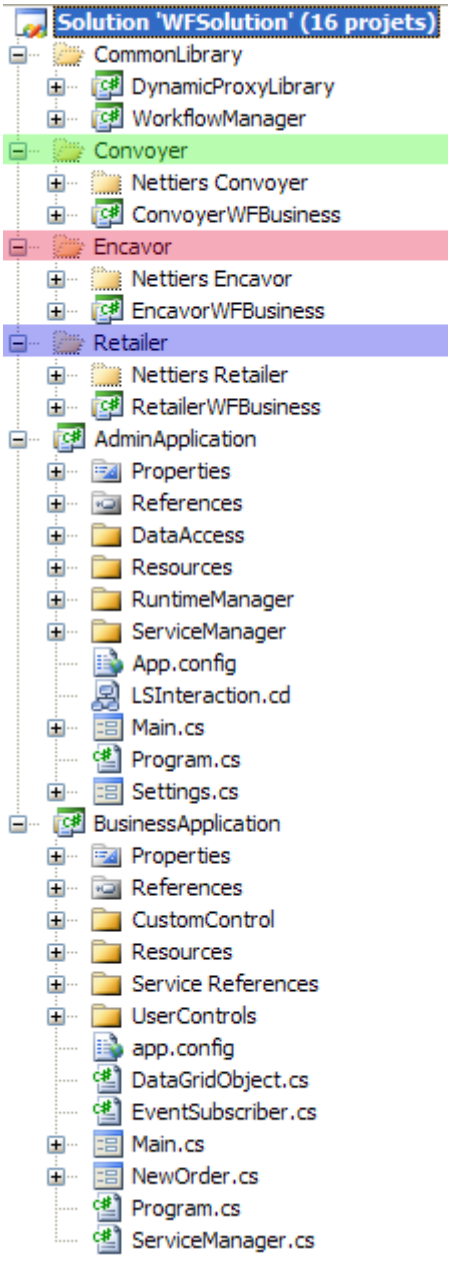


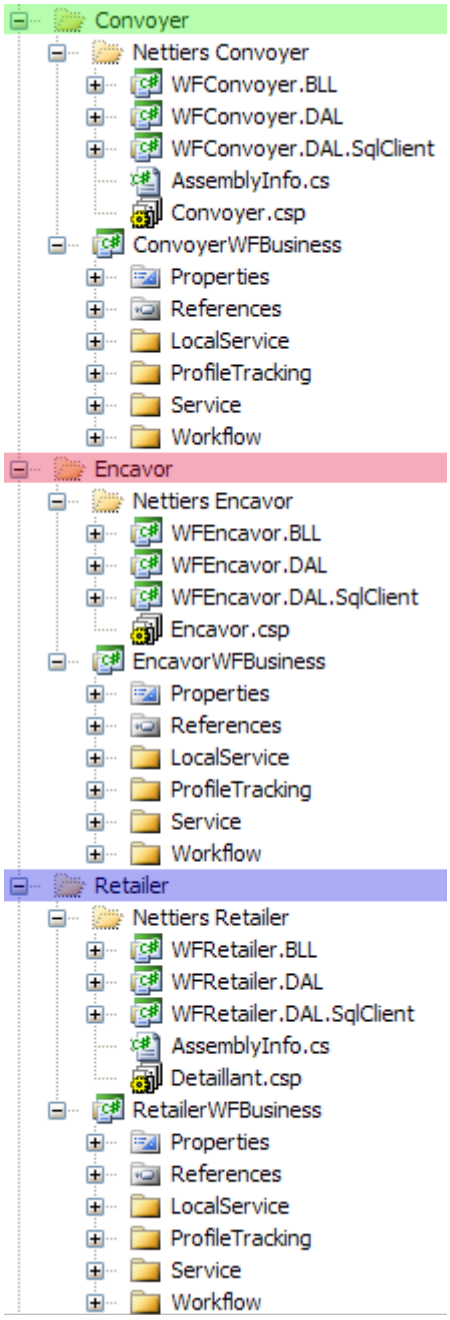
Solution Visual Studio

Description :

Solution Visual Studio de l'implémentation de la chaîne logistique. Présentation des projets ainsi que des classes de la solution.

Le premier niveau des nœuds de l'arborescence de la solution présente quatre dossiers et deux projets :





1 Dossier CommonLibrary

Comme son nom l'indique, ce dossier contient des librairies de classes communes pour l'ensemble de la solution. Ces classes sont divisées en deux projets distincts.

DynamicProxy

Ce projet contient les classes nécessaires à la génération dynamique des « proxy » des services web des acteurs de la chaîne logistique. Ces classes sont utilisées à chaque fois qu'un acteur de la chaîne doit contacter le service web d'un autre acteur. Les deux classes appelées dans les autres projets sont « DynamicProxyFactory » et « DynamicProxy ». L'utilisation de celles-ci est abordée plus bas au point 3.3.2 Communication à la page 73.

WorkflowManager

Ce projet contient les classes communes permettant la gestion des processus dans les différents projets.

CustomTrackingChannel : Cette classe est le « channel » donné par le *CustomTrackingService* au moteur d'exécution des processus. Cet objet formate les informations pour le CustomTrackingService.

CustomTrackingService : Cette classe contient le service personnalisé de suivi référencé dans le moteur d'exécution des processus permettant d'afficher en tant réel les informations de suivi.

Info : Cette classe représente le formulaire utilisé en tant que *MessageBox* pour l'affichage d'information à l'utilisateur.

LSException : Cette classe est une exception personnalisée qui est utilisée au sein des services locaux des processus de chaque acteur. Lorsqu'elle est levée, l'instance de processus la réceptionne et se termine. Elle contient le message informant de l'erreur survenue causant l'arrêt de l'instance du processus.

MyEventArgs : Cette classe est de type « *EventArgs* » et permet de passer une collection d'objets aux évènements.

TrackingChannelEventArgs : C'est un objet de type *EventArgs* contenant les informations de suivi qui est créé à chaque fois que le service de suivi lève un évènement.

TrackingProfileHelper : C'est une classe outil permettant de manager les profils de suivi.

TrackingWriter : C'est une classe outil permettant d'aller rechercher les informations de suivi pour une instance ou pour une collection d'instances données et formate le résultat pour son affichage dans l'application d'administration.

WorkflowDataGrid : Dans l'application d'administration, c'est ce type d'objet qui est passé en tant que *datasource* du *datagridview* permettant l'affichage des processus. Il contient le *StringBuilder utilisé pour l'affichage des données du suivi des processus.*

WorkflowBDataGrid : Dans l'application business, c'est le type d'objet qui est passé en tant que *datasource* du *datagridview* permettant l'affichage des processus. Il contient l'objet principal de l'acteur déterminé au lancement de l'application. Cet objet référence toutes les informations nécessaires à l'affichage des données métiers d'une instance de processus.

WorkflowInstanceWrapper : Cet objet sert de conteneur pour une instance de processus. Il permet lorsque l'instance de processus est terminée, de garder, s'il y en a, les paramètres de sorties de cette instance de processus. Il contient également les données de l'exception si l'instance ne s'est pas terminée correctement.

WorkflowInstanceManager : Cet objet contient la référence du moteur d'exécution des processus, ainsi qu'un dictionnaire d'objets de type *WorkflowInstanceWrapper* représentant les instances de processus en cours. C'est le gestionnaire des processus.

2 Dossier Convoyer

Le dossier Convoyer contient tous les éléments permettant la gestion de l'activité du transporteur. Il est composé d'un dossier Nettiers Convoyer et d'un projet ConvoyerWFBusiness. Le détail de ce dossier est présenté au point 7 Les classes des acteurs à la page 167.

3 Dossier Encavor

Le dossier Encavor contient tous les éléments permettant la gestion de l'activité de l'encaveur. Il est composé d'un dossier Nettiers Encavor et d'un projet EncavorWFBusiness. Le détail de ce dossier est présenté au point 7 Les classes des acteurs à la page 167.

4 Dossier Retailer

Le dossier Retailer contient tous les éléments permettant la gestion de l'activité du détaillant. Il est composé d'un dossier Nettiers Retailer et d'un projet RetailerWFBusiness. Le détail de ce dossier est présenté au point 7 Les classes des acteurs à la page 167.

5 Projet AdminApplication

Ce projet est le logiciel représentant l'application d'administration. Il contient les classes de gestions des différents services, les classes du design de l'application et le fichier de configuration.

Dossier DataAccess

Ce dossier contient une classe pour chaque type d'acteur de la chaîne logistique. Chaque classe contient des méthodes statiques appelées par le service de communication pour l'application business. Ces méthodes sont utilisées pour l'interrogation de la base de données de l'acteur correspondant.

Dossier RuntimeManager

Ce dossier contient deux classes, la classe *RuntimeEventArgs* qui est l'objet transmis lorsqu'un évènement, servant à informer sur l'état du moteur d'exécution des processus, est levé. La classe *RuntimeHosting* est une classe singleton qui référence l'objet *WorkflowRuntimeManager. Cette classe contient les méthodes permettant de démarrer et d'arrêter le moteur d'exécution des processus.*

Dossier ServiceManager

Ce dossier contient les classes nécessaires aux services de communication entre l'application d'administration et l'application business.

EventPublisherForBusinessApplic : Cette classe implémente le service WCF pour l'envoi évènementiel d'informations à toutes les applications business en cours d'exécution.

IServiceForBusinessApplic : Cette classe est l'interface des services de communication WCF entre l'application d'administration et l'application business.

ServiceEventArgs : Cet objet de type *EventArgs* est utilisé pour donner le statut des services de communication.

ServiceForBusinessApplic : Cette classe implémente le service WCF utilisé par l'application business pour la demande d'informations des données métiers des processus en cours. Ce service utilisera alors les méthodes d'une des classes du dossier DataAccess pour ressortir les informations de la base de données. Ce service permet également d'informer les instances de processus d'une action utilisateur.

Service Hosting : Cette classe est une classe singelton qui héberge les instances des deux services de communication. Elle contient les méthodes permettant de démarrer et d'arrêter ces services de communication.

Fichiers

App.config : Ce fichier permet de configurer l'application d'administration. Il permet de configurer les informations des services de communication, le choix du type d'acteur pour cette application, ainsi que différents paramètres des informations contenues dans la base de données métiers, les paramètres pour la couche d'accès aux données Nettiers.

Main : C'est le formulaire d'affichage principal de l'application d'administration. Il contient le design de l'application.

Program : C'est la classe qui lance l'application d'administration.

Settings : C'est le formulaire appelé par l'application d'administration permettant de choisir le paramétrage des temps d'attentes.

6 Projet BusinessApplication

Ce projet est le logiciel représentant l'application business d'affichage des données métiers. Il comprend les classes de design de l'application, les classes de communication avec les services de l'application d'administration, ainsi que le fichier de configuration de l'application.

Dossier CustomControl

Ce dossier contient les classes personnalisées héritant des classes de base du namespace *System.Windows.Forms.* Elles permettent une personnalisation des contrôles du Framework 2.0.

Dossier Resources

Ce dossier contient les fichiers images servant au design de l'application.

Dossier Service References

Il contient les classes proxy des deux services de communication de l'application d'administration. Ces classes sont générées automatiquement par une fonctionnalité intégrée dans Visual Studio.

Dossier UserControls

Il contient les trois *UserControl* permettant d'afficher les informations métiers des trois différents types d'acteurs de la chaîne logistique.

Fichiers

App.config : Ce fichier permet de configurer l'application business. Il permet de spécifier le type d'acteur pour l'application permettant ainsi d'afficher les informations correspondantes à ce type. Il contient également les informations liées aux services de communication WCF de l'application d'administration.

DataGridObject : Cette classe représente l'objet utilisé en tant que *DataSource* pour le *DataGridView* du formulaire permettant au détaillant de passer une nouvelle commande.

EventSubscriber : Cette classe hérite de la classe de callback du service évènementiel de communication de l'application d'administration. C'est elle qui réceptionne les évènements reçus permettant ensuite d'afficher les informations correspondantes.

Main : C'est le formulaire d'affichage principal de l'application business. Il contient le design de l'application.

NewOrder : C'est le formulaire d'affichage permettant au détaillant de passer une nouvelle commande.

Program : C'est la classe qui lance l'application d'administration.

ServiceManager : Cette classe contient les méthodes permettant l'appel au service de communication de l'application d'administration. La création de l'appel au service se fait à l'aide de la classe proxy générée.

7 Les classes des acteurs

Ce point présente les projets et classes contenues dans les dossiers Convoyer, Encavor, Retailer à la racine de la solution. Ils n'ont pas été détaillés dans leur point respectif car ils contiennent les classes similaires pour les trois acteurs. Le nom des projets et des classes changent mais leur utilité est semblable quelque soit l'acteur.

Dossier Nettiers

Ce dossier fait référence à la solution générée par l'outil Codesmith à l'aide du fichier de projet Codesmith Encavor.csp. La solution générée est composée de trois projets permettant l'accès à la base de données, ainsi que l'utilisation d'objets représentant les entités de la base de données et contenant les valeurs sauvées dans celle-ci. Le dossier *Acteur*.BLL contient la représentation objet de toutes les entités de la base de données métiers de l'acteur. Les classes du projet *Acteur*.DAL et *Acteur*.DAL.SqlClient s'occupent quant à elles de l'accès à la base de données et aux requêtes demandées.

Projet *Acteur*WFBusiness

Ce projet contient le service web de l'acteur ainsi que toute la logique métier de l'activité de l'acteur.

Dossier LocalService

***Acteur*LocalService :** Cette classe est le service local de l'acteur. Elle sert de passerelle entre les instances de workflow et l'application d'administration. Elle contient les variables des paramètres du fichier de configuration. Les méthodes appelées par le service web de l'acteur, les méthodes appelées par l'application d'administration, ainsi que les méthodes appelées par les instances de processus. Elle contient également les évènements utilisés par le service local pour contacter une instance de workflow ainsi que les évènements utilisés par le service local pour interagir avec l'application d'administration.

***Acteur*LocalServiceDB :** Cette classe est également un service local de l'acteur. Elle est utilisée par le service local principal et également directement par les instances de processus. C'est elle qui gère les transactions avec la base de données métiers de l'acteur.

I*Acteur*LocalService : C'est l'interface du service local. Elle est obligatoire pour que le service puisse être enregistré auprès du moteur d'exécution des processus et puisse également être utilisée à l'intérieur de la classe de modélisation du processus.

I*Acteur*LocalServiceDB : C'est l'interface du service local de la base de données. De même que pour l'interface *IActeurLocalService* elle est obligatoire pour les mêmes raisons.

LSMsgEventArgs : Cet objet hérite de *ExternalDataEventArgs*. Il est utilisé par les services locaux afin d'interagir avec les instances de processus. Il permet de passer des objets en paramètres et des messages d'erreurs.

Dossier ProfileTracking

Il contient la classe ***Acteur*TrackingProfile** qui permet de spécifier au service de suivi un profil personnalisé pour l'acteur spécifié. Cette classe permet de modifier le profil de base pour ressortir les informations utiles à l'acteur.

Dossier Service

IWS*Acteur* : C'est l'interface du service web business de l'acteur. Elle contient toutes les méthodes qui peuvent être appelées par ce service.

WS*Acteur* : C'est le service web business de l'acteur. Il référence toutes les méthodes de l'interface et traite les demandes des clients. Il fait appel aux méthodes du local service pour interagir avec les instances de processus.

Dossier Workflow

Ce dossier contient le fichier de modélisation du processus de l'activité de l'acteur. Ce fichier est séparé en trois fichiers distincts. Le premier permet la modélisation graphique du processus, le deuxième comprend toute la logique métier liée au processus et finalement le troisième contient les règles utilisées dans le processus.

Manuel d'utilisation

Description :

Ce manuel décrit l'utilisation des applications prototypes développées pour le scénario de la chaîne logistique.

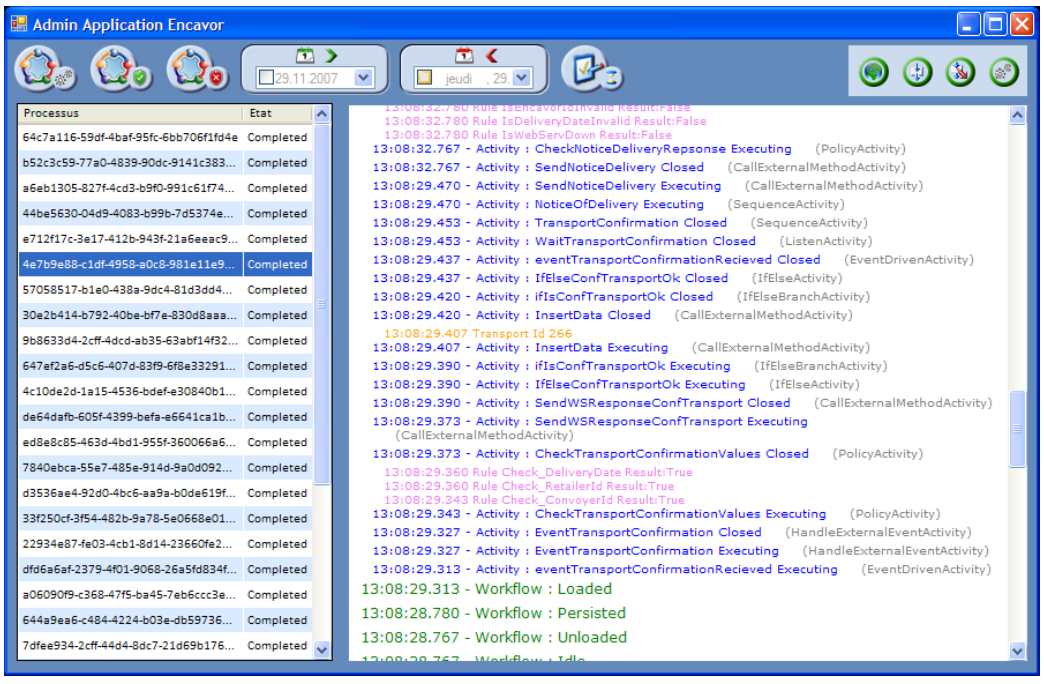
Application d'administration
Application business

1 Application d'administration

1.1 Menu

1	Affiche les processus en cours
2	Affiche les processus terminés avec succès
3	Affiche les processus terminés par erreur durant leur exécution
4	Sélectionne les processus à afficher qui se sont terminés après cette date
5	Sélectionne les processus à afficher qui se sont terminés avant cette date
6	Affiche la fenêtre qui détermine le mode de reprise des processus
7	Affiche le statut du service business
8	Affiche le statut du service utilisé pour l'application business
9	Affiche le statut du service évènementiel utilisé pour l'application business
10	Affiche le statut du moteur d'exécution des processus

1.2 Interface



L'application est séparée en trois parties ; la partie supérieure contient le menu, la partie de gauche affiche les processus choisis et la partie de droite affiche les données relatives au processus sélectionné.

Les données affichées dans la partie de droite représentent le suivi du processus. Les couleurs déterminent le type d'information de suivi.

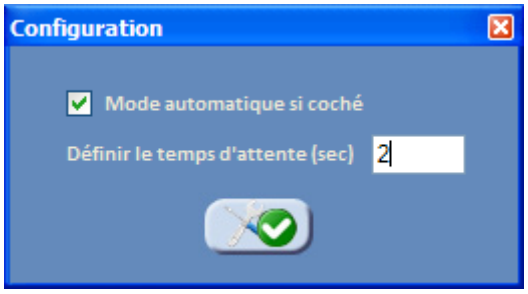
Vert	Affichage des données d'évènement processus
Bleu	Affichage des données d'évènement activité
Rose	Affichage des données d'évènement de règles
Orange	Affichage des données d'évènement utilisateur
Brun	Affichage des données du type d'activité

Les quatre icones affichent l'état des services.Elles sont vertes si le service respectif a pu démarrer correctement, au cas contraire elles sont rouges. Il faut dès lors vérifier les configurations du service dans le fichier de configuration de l'application.

Le choix de la période d'affichage des processus fonctionne uniquement pour les processus terminés et non pour les processus en cours d'exécution.

La fenêtre de sélection du mode d'exécution des processus permet de sélectionner le mode manuel ou alors de définir un laps de temps en secondes pour la reprise des activités nécessitant un délai.

Si aucun processus n'est en cours d'exécution, le mode est sauvegardé dans l'application pour les processus à venir. Si le « runtime » contient des processus en cours d'exécution, ils sont reconfigurés pour passer dans le mode choisi.



2 Application business

2.1 Menu

1	Affiche les processus en cours
2	Affiche les processus terminés avec succès
3	Affiche les processus terminés par erreur durant leur exécution
4	Sélectionne les processus à afficher qui se sont terminés après cette date
5	Sélectionne les processus à afficher qui se sont terminés avant cette date
6	Sélectionne le processus représentant le numéro de commande inscrit dans la zone de texte
7	Permet de faire passer le processus à l'activité suivante
8	Affiche la fenêtre permettant d'effectuer une nouvelle commande (Uniquement pour l'application du détaillant)

2.2 Interface

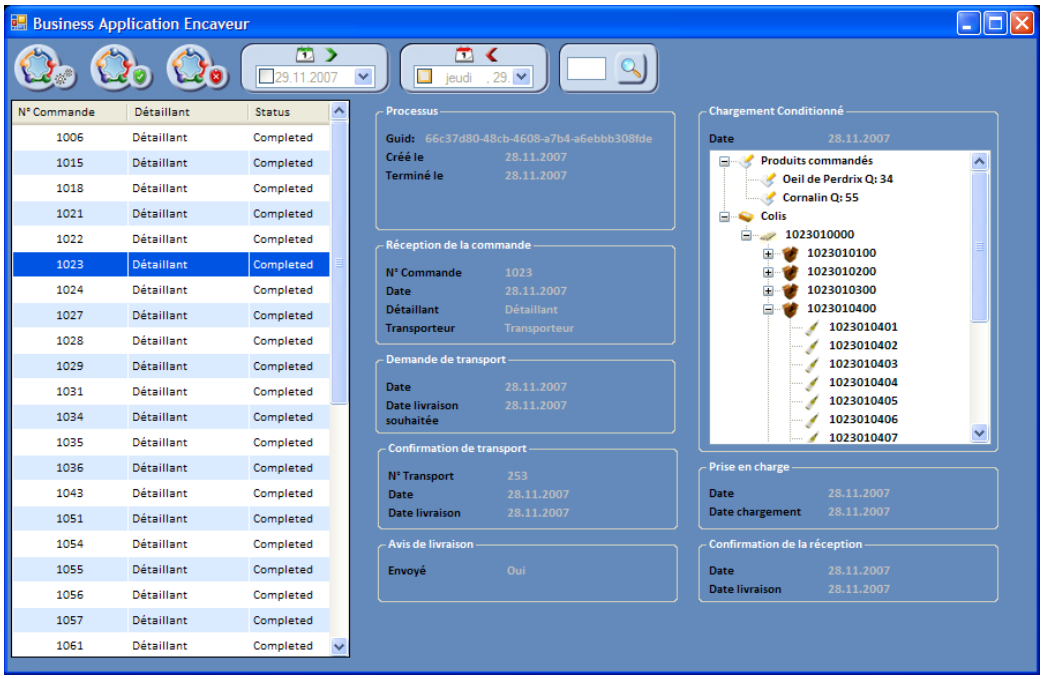
L'application est séparée en trois parties ; la partie supérieure contient le menu, la partie de gauche affiche les processus choisis et la zone de partie affiche les données relatives au processus sélectionné.

Le choix de la période d'affichage des processus fonctionne uniquement pour les processus terminés et non pour les processus en cours d'exécution.

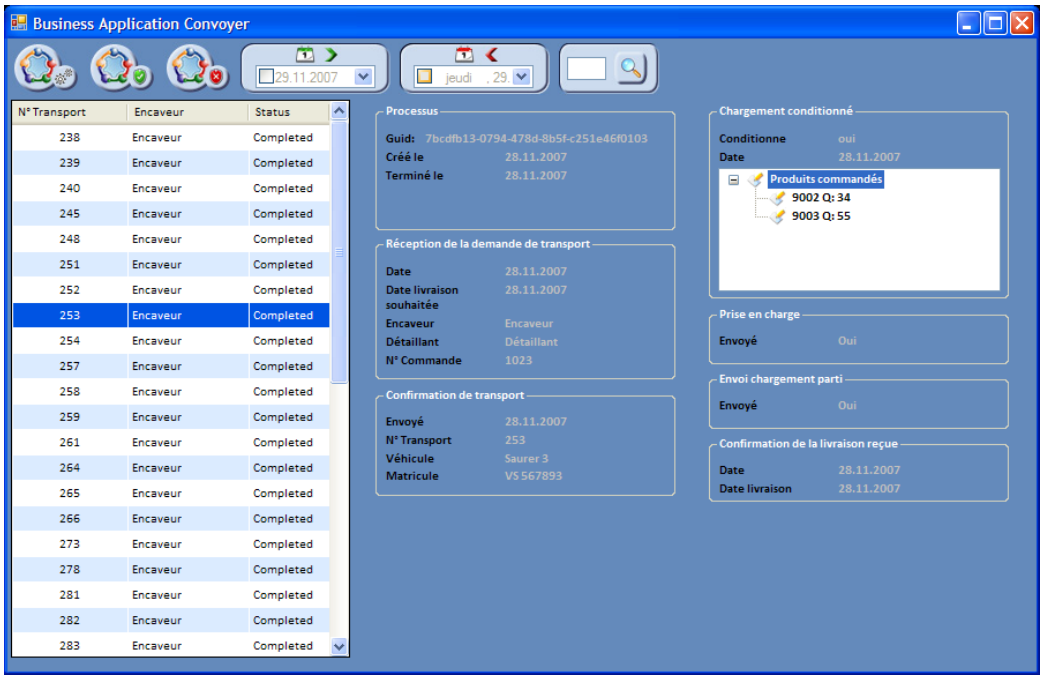
Lorsqu'un processus entre en mode d'attente d'un évènement utilisateur, le bouton n° 7 apparaît et permet de faire passer le processus à l'activité suivante.

Les données présentées dans la partie de gauche affichent les informations relatives au processus sélectionné. Ces informations sont regroupées par les Use Case déterminant la logique métier de chaque acteur.

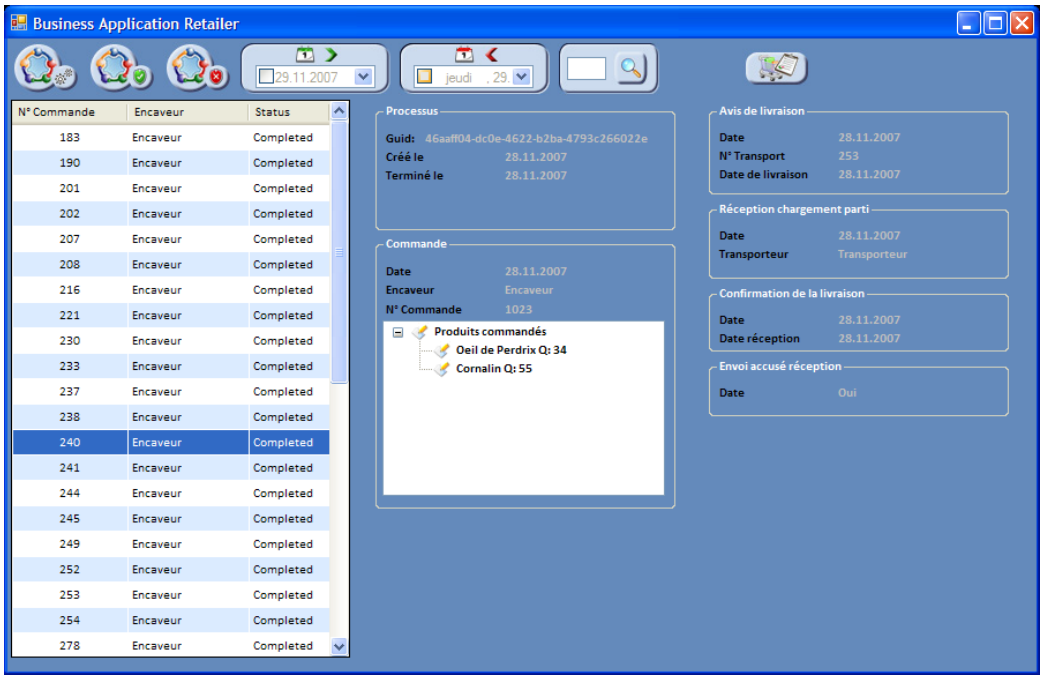
Interface Encaveur

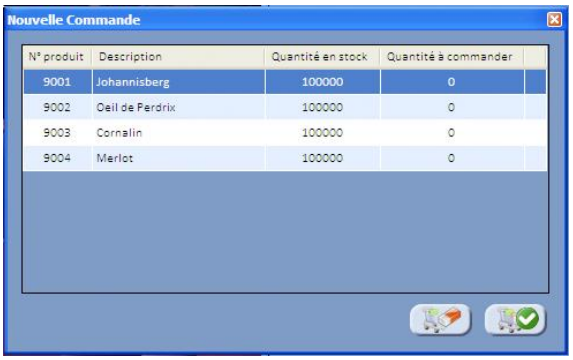


Interface Transporteur



Interface Détaillant





Le tableau pour une nouvelle commande affiche tous les produits actuellement en stock chez l'encaveur. La colonne quantité en stock représente le nombre de ces produits dans le stock du détaillant. Afin de passer une commande il suffit de choisir la quantité désirée dans la colonne quantité à commander.

Manuel d'installation

Description :

Ce manuel décrit les étapes à réaliser pour exécuter les applications prototypes.

Ce manuel présente l'installation à effectuer pour pouvoir exécuter les applications prototypes au sein de la même machine.

Pré-requis

- Microsoft SQL Server 2005
- Framework 3.0

Installation des bases de données

Le scénario comporte six bases de données, chaque acteur dispose d'une base de données métier et d'une base de données pour la gestion de la persistance et du suivi des processus.

Les bases de données sont fournies sous leur représentation fichier, c'est-à-dire pour chaque base il faut disposer du fichier « .mdf » ainsi que du fichier « .ldf ». Afin d'installer toutes les bases vous devez avoir 12 fichiers en votre possession.

Pour installer les bases il faut les attacher à SQL Server, voici les étapes à suivre.

- Se connecter au moteur de base de données
- Clic droit sur le dossier « Base de données » et sélectionner « Joindre ».
- Dans la nouvelle fenêtre clic sur ajouter et sélectionner le fichier « .mdf » de la base à installer.

Répéter ces étapes pour chacune des bases de données du scénario.

Au cas ou la configuration du « connectionString » des bases n'est pas faite en mode « Integrated Security », créer un utilisateur SQL et affecter le rôle pour se connecter aux bases de données.

Utilisateur SQL :

- Développer le dossier « Sécurité » puis clic droit sur « Connexions » sélectionner « Nouvelle connexion »
- Choisir authentification SQL Server et renseigner le nom et le mot de passe.
- Dans l'onglet « Mappage de l'utilisateur » sélectionner les bases de données du scénario et sélectionner comme mappage au rôle « public » et « db_owner ».
- Pour chaque base de données ensuite on va le paramétrer en temps que propriétaire de la base. Clic droit sur la base « Propriétés », dans l'onglet « Fichier » sélectionner ce nouvel utilisateur comme Propriétaire.

Configuration des bases dans le fichier « .config »

Maintenant que les bases sont installées, il faut configurer l'application administration de manière à ce qu'elle puisse se connecter à ces bases.

Etape à effectuer pour les trois applications d'administration :

Dans le dossier de l'application d'administration de l'encaveur, ouvrir le fichier « AdminApplication.exe.config ».

A la fin de la section « appSettings » se trouve la clé « key="EncavorPersTrack" ». Modifier la valeur « Data Source » afin de correspondre à la machine, il est possible de modifier la valeur de « Initial Catalog » si vous l'avez modifiée dans le moteur de base de données lors de l'ajout de la base :

```
Initial Catalog=EncaveurPersTrack;Data Source= « nomdelamachine » ;Integrated Security=true;
```

Si vous ne voulez pas ou que vous n'avez pas les droits pour la connexion aux bases de données avec le login Windows, modifier comme suit :

```
Initial Catalog=EncaveurPersTrack;Data Source= « nomdelamachine » ;User Id = utilisateurCrée ; Password = PwdUtilisateur;
```

Le champ « User Id » correspond à l'utilisateur SQL créé précédemment.

La base de données de l'encaveur pour la gestion de la persistance et du suivi et configurée. Il reste encore à configurer la base de données métier.

Dans la section « connectionStrings » modifier comme suit la valeur de la propriété « connectionString » pour la clé « netTiersConnectionStringEncavor ».

```
Initial Catalog=EncaveurDB; Source= « nomdelamachine » ;Integrated Security=true;
```

Ou alors :

```
Initial Catalog= EncaveurDB;Data Source= « nomdelamachine » ;User Id = utilisateurCrée ; Password = PwdUtilisateur;
```

Effectuer ces changements dans les trois dossiers des applications administration.

Configuration des services

Les services se configurent également dans le fichier « .config » de l'application d'administration. Les services utilisent des ports différents pour communiquer, donc il faut que le firewall de l'ordinateur accepte les communications à travers ceux-ci. (Port TCP : 48081, 48082, 48083, 47081, 47082, 47083, 46081, 46082, 46083).

La modification à effectuer est le changement des adresses des services par rapport à la machine hôte. Modifier les informations surlignées dans l'image ci-dessous. Remplacer par le nom de la machine hôte.

```
<system.serviceModel>
    <services>

        <service name="EncavorWFBusiness.Service.WSEncavor" behaviorConfiguration="myBehavior">
            <host>
                <baseAddresses>
                    <add baseAddress="http://MACBOOKPRO-SP:48081/EncavorService/"/>
                </baseAddresses>
            </host>
            <endpoint address="" binding="wsHttpBinding" bindingConfiguration="wsHttp" contract="EncavorWFBusiness.Service.IWSEncavor"/>
            <endpoint address="" binding="wsHttpBinding" bindingConfiguration="wsHttp" contract="EncavorWFBusiness.Service.IWSEncavor2"/>
            <endpoint address="http://MACBOOKPRO-SP:48081/EncavorService/MEX2" binding="mexHttpBinding" contract="IMetadataExchange"/>
        </service>

        <service name="ServiceManager.ServiceForBusinessApplic" behaviorConfiguration="myBehaviorTcp">
            <host>
                <baseAddresses>
                    <add baseAddress="net.tcp://MACBOOKPRO-SP:48082/ServiceForBusinessApplic/"/>
                </baseAddresses>
            </host>
            <endpoint address="MEX" binding="mexTcpBinding" contract="IMetadataExchange"/>
            <endpoint address="" binding="netTcpBinding" bindingConfiguration="netTcp" contract="ServiceManager.IServiceForBusinessApplic"/>
        </service>

        <service behaviorConfiguration="myBehaviorTcp" name="ServiceManager.EventPublisherForBusinessApplic">
            <host>
                <baseAddresses>
                    <add baseAddress="net.tcp://MACBOOKPRO-SP:48083/EventPublisherForBusinessApplic"/>
                </baseAddresses>
            </host>
            <endpoint address="" binding="netTcpBinding" bindingConfiguration="netTcp" contract="ServiceManager.IServiceForBusinessApplicEvent"/>
            <endpoint address="MEX" binding="mexTcpBinding" contract="IMetadataExchange"/>
        </service>

    </services>

    <behaviors>...
    <bindings>...
</system.serviceModel>
```

Effectuer cette modification pour chaque application d'administration du scénario

La dernière étape de configuration des services consiste à renseigner dans les bases de données métier des acteurs l'adresse des services web respectifs.

Pour ce faire, il faut aller modifier les valeurs de la table ACTEUR de chacune des bases de données métier des acteurs. Le champ « wsUri » contient la référence au service web.

- Clic droit sur la table ACTEUR de la base de données « EncaveurDB » et sélectionner « ouvrir la table ».
- Modifier la valeur du champ « wsUri » pour chacun des acteurs insérés dans la base. Modifier uniquement le nom de la machine par le nom de la machine hôte comme effectué précédemment dans les fichiers de configuration.

Renouveler cette étape pour les deux autres bases ; DetaillantDB, TransporteurDB.

Configuration Application Business

De la même manière que pour la configuration des services il faut changer l'adresse des services utilisés pour correspondre avec la machine hôte. Voici les étapes : aller dans le dossier de l'application business de l'encaveur et ouvrir le fichier « BusinessApplication.exe.config ». Modifier les informations surlignées dans l'image ci-dessous.

```
<system.serviceModel>
    <bindings>...
    <client>
        <endpoint address="net.tcp://macbookpro-sp:8082/ServiceForBusinessApplic/"
            binding="netTcpBinding" bindingConfiguration="NetTcpBinding_ServiceForBusinessApplic"
            contract="BusinessApplication.SrvBusiApplic.ServiceForBusinessApplic"
            name="NetTcpBinding_ServiceForBusinessApplic" />
        <endpoint address="net.tcp://macbookpro-sp:8083/EventPublisherForBusinessApplic"
            binding="netTcpBinding" bindingConfiguration="NetTcpBinding_WFEventService"
            contract="BusinessApplication.SrvEventBusiApplic.WFEventService"
            name="NetTcpBinding_WFEventService" />
    </client>
</system.serviceModel>
```

Effectuer cette modification pour chaque application business du scénario

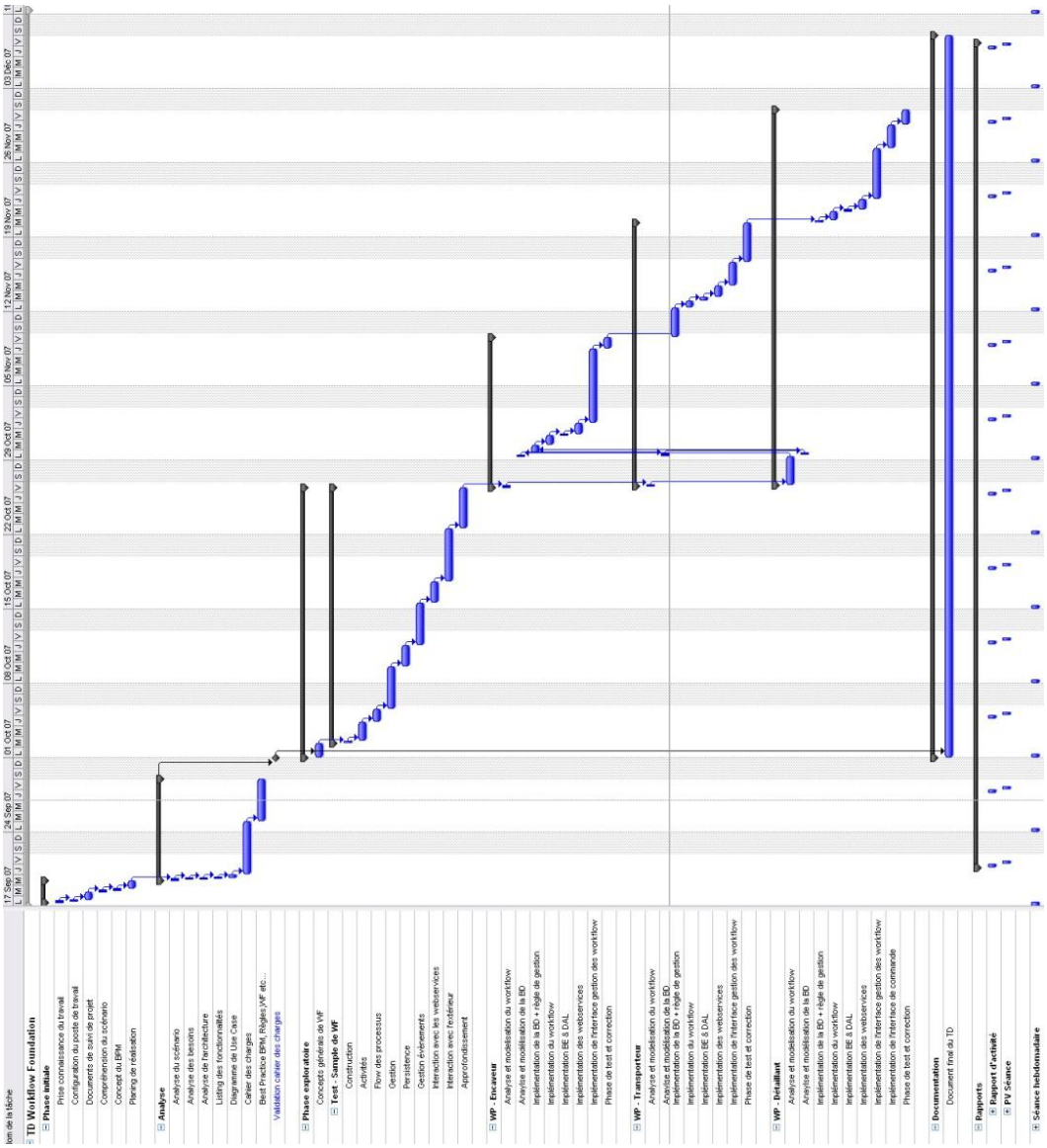
Pour l'application du détaillant il faut encore modifier l'url pour atteindre le service web permettant la passation d'une nouvelle commande. La clé se trouve dans le nœud « appSettings ».

```
    <!--    Url Web Service Encavor2 using to retrieve product that can be ordered-->
    <add key="WSEncavor2" value="http://MACBOOKPRO-SP:48081/EncavorService/?wsdl"/>
</appSettings>
```

Planification

Description :

Diagramme de Gantt de la planification initiale
Tableau du suivi de la planification


TD Workflow Foundation
Phase initiale
Prise connaissance du travail
Configuration du poste de travail
Documents de suivi de projet
Compréhension du scénario
Concept du BPM
Planing de réalisation
Analyse
Analyse du scénario
Analyse des besoins
Analyse de l'architecture
Listing des fonctionnalités
Diagramme de Use Case
Cahier des charges
Best Practice BPM, Règles,WF etc...
Validation cahier des charges
Phase exploratoire
Concepts généraux de WF
Test - Sample de WF
Construction
Activités
Flow des processus
Gestion
Persistence
Gestion événements
Interaction avec les webservices
Interaction avec l'extérieur
Approfondissement
WP - Encaveur
Analyse et modélisation du workflow
Analyse et modélisation de la BD
Implémentation de la BD + règle de gestion
Implémentation du workflow
Implémentation BE & DAL
Implémentation des webservices
Implémentation de l'interface gestion des workflow
Phase de test et correction
WP - Transporteur
Analyse et modélisation du workflow
Analyse et modélisation de la BD
Implémentation de la BD + règle de gestion
Implémentation du workflow
Implémentation BE & DAL
Implémentation des webservices
Implémentation de l'interface gestion des workflow
Phase de test et correction
WP - Détaillant
Analyse et modélisation du workflow
Analyse et modélisation de la BD
Implémentation de la BD + règle de gestion
Implémentation du workflow
Implémentation BE & DAL
Implémentation des webservices
Implémentation de l'interface gestion des workflow
Implémentation de l'interface de commande
Phase de test et correction
Documentation
Document final du TD
Rapports
Rapport d'activité
PV Séance
Séance hebdomadaire

Planification Initiale Microsoft Windows Worklow Foundation

Travail de diplôme 2007 : Probst Stéphane

TD Workflow Foundation	506 *heures*	17.09.2007 10:15 *date début*	10.12.2007 09:00 *date fin*
Phase initiale	**16.8**	**17.09.2007 10:15**	**19.09.2007 11:00**
✓ *Prise connaissance du travail*	*1.75*	*17.09.2007 10:15*	*17.09.2007 12:00*
✓ *Configuration du poste de travail*	*2*	*17.09.2007 13:00*	*17.09.2007 15:00*
✓ *Documents de suivi de projet*	*4*	*17.09.2007 15:00*	*18.09.2007 10:00*
✓ *Compréhension du scénario*	*2*	*18.09.2007 10:00*	*18.09.2007 12:00*
✓ *Concept du BPM*	*3*	*18.09.2007 13:00*	*18.09.2007 16:00*
✓ *Planing de réalisation*	*4*	*18.09.2007 16:00*	*19.09.2007 11:00*
Analyse	**61**	**19.09.2007 11:00**	**28.09.2007 17:00**
✓ *Analyse du scénario*	*2*	*19.09.2007 11:00*	*19.09.2007 14:00*
✓ *Analyse des besoins*	*1*	*19.09.2007 14:00*	*19.09.2007 15:00*
✓ *Analyse de l'architecture*	*1*	*19.09.2007 15:00*	*19.09.2007 16:00*
✓ *Listing des fonctionnalités*	*0.5*	*19.09.2007 16:00*	*19.09.2007 16:30*
✓ *Diagramme de Use Case*	*0.5*	*19.09.2007 16:30*	*19.09.2007 17:00*
✓ *Cahier des charges*	*24*	*20.09.2007 08:00*	*24.09.2007 17:00*
✓ *Best Practice BPM, Règles,WF etc...*	*32*	*25.09.2007 08:00*	*28.09.2007 17:00*
✓ *Validation cahier des charges*	*0*	*01.10.2007 08:00*	*01.10.2007 08:00*
Phase exploratoire	**155**	**01.10.2007 08:00**	**26.10.2007 11:00**
✓ *Concepts généraux de WF*	*10*	*01.10.2007 08:00*	*02.10.2007 10:00*
Test - Sample de WF	**145**	**02.10.2007 10:00**	**26.10.2007 11:00**
✓ *Construction*	*4*	*02.10.2007 10:00*	*02.10.2007 15:00*
✓ *Activités*	*12*	*02.10.2007 15:00*	*04.10.2007 10:00*
✓ *Flow des processus*	*12*	*04.10.2007 10:00*	*05.10.2007 15:00*
✓ *Gestion*	*16*	*05.10.2007 15:00*	*09.10.2007 15:00*
✓ *Persistence*	*16*	*09.10.2007 15:00*	*11.10.2007 15:00*
✓ *Gestion évènements*	*16*	*11.10.2007 15:00*	*15.10.2007 15:00*
✓ *Interaction avec les webservices*	*16*	*15.10.2007 15:00*	*17.10.2007 15:00*
✓ *Interaction avec l'extérieur*	*24*	*17.10.2007 15:00*	*22.10.2007 15:00*
! *Approfondissement*	*29*	*22.10.2007 15:00*	*26.10.2007 11:00*
WP - Encaveur	**75**	**26.10.2007 11:00**	**09.11.2007 15:00**
✓ *Analyse et modelisation du workflow*	*2*	*26.10.2007 11:00*	*26.10.2007 14:00*
✓ *Analyse et modélisation de la BD*	*2*	*29.10.2007 09:00*	*29.10.2007 11:00*
✓ *Implémentation de la BD + règle de gestion*	*4*	*29.10.2007 16:00*	*30.10.2007 11:00*
✓ *Implémentation du workflow*	*6*	*30.10.2007 11:00*	*31.10.2007 09:00*
✓ *Implémentation BE & DAL*	*2*	*31.10.2007 09:00*	*31.10.2007 11:00*
✓ *Implémentation des webservices*	*9*	*31.10.2007 11:00*	*01.11.2007 12:00*
✓ *Implémentation de l'interface gestion des workflow*	*40*	*01.11.2007 13:00*	*08.11.2007 12:00*
! *Phase de test et correction*	*10*	*08.11.2007 13:00*	*09.11.2007 15:00*
WP - Transporteur	**55**	**26.10.2007 14:00**	**20.11.2007 09:00**
Analyse et modelisation du workflow	*2*	*26.10.2007 14:00*	*26.10.2007 16:00*
✓ *Anaylse et modélisation de la BD*	*2*	*29.10.2007 11:00*	*29.10.2007 14:00*
✓ *Implémentation de la BD + règle de gestion*	*4*	*09.11.2007 15:00*	*12.11.2007 10:00*
✓ *Implémentation du workflow*	*6*	*12.11.2007 10:00*	*12.11.2007 17:00*
✓ *Implémentation BE & DAL*	*2*	*13.11.2007 08:00*	*13.11.2007 10:00*
✓ *Implémentation des webservices*	*9*	*13.11.2007 10:00*	*14.11.2007 11:00*
✓ *Implémentation de l'interface gestion des workflow*	*20*	*14.11.2007 11:00*	*16.11.2007 16:00*
! *Phase de test et correction*	*10*	*16.11.2007 16:00*	*20.11.2007 09:00*

WP - Détaillant		**75**	**26.10.2007 16:00**	**30.11.2007 17:00**
	Analyse et modelisation du workflow	*2*	*26.10.2007 16:00*	*29.10.2007 09:00*
✓	*Anaylse et modélisation de la BD*	*2*	*29.10.2007 14:00*	*29.10.2007 16:00*
✓	*Implémentation de la BD + règle de gestion*	*4*	*30.10.2007 09:00*	*30.10.2007 14:00*
✓	*Implémentation du workflow*	*6*	*30.10.2007 14:00*	*31.10.2007 11:00*
✓	*Implémentation BE & DAL*	*2*	*31.10.2007 11:00*	*31.10.2007 14:00*
✓	*Implémentation des webservices*	*9*	*31.10.2007 14:00*	*02.11.2007 15:00*
✓	*Implémentation de l'interface gestion des workflow*	*20*	*02.11.2007 15:00*	*07.11.2007 10:00*
✓	*Implémentation de l'interface de commande*	*20*	*07.11.2007 10:00*	*09.11.2007 15:00*
!	*Phase de test et correction*	*10*	*09.11.2007 15:00*	*30.11.2007 17:00*
Documentation		**40**	**01.10.2007 08:00**	**07.12.2007 17:00**
	Document final du TD	*40*	*01.10.2007 08:00*	*07.12.2007 17:00*
Rapports		**15**	**20.09.2007 16:15**	**07.12.2007 08:30**
	Rapport d'activité	**9**	**20.09.2007 16:15**	**06.12.2007 17:00**
✓	*Rapport d'activité 1*	*0.75*	*20.09.2007 16:15*	*20.09.2007 17:00*
✓	*Rapport d'activité 2*	*0.75*	*27.09.2007 16:15*	*27.09.2007 17:00*
✓	*Rapport d'activité 3*	*0.75*	*04.10.2007 16:15*	*04.10.2007 17:00*
✓	*Rapport d'activité 4*	*0.75*	*11.10.2007 16:15*	*11.10.2007 17:00*
✓	*Rapport d'activité 5*	*0.75*	*18.10.2007 16:15*	*18.10.2007 17:00*
✓	*Rapport d'activité 6*	*0.75*	*25.10.2007 16:15*	*25.10.2007 17:00*
✓	*Rapport d'activité 7*	*0.75*	*01.11.2007 16:15*	*01.11.2007 17:00*
✓	*Rapport d'activité 8*	*0.75*	*08.11.2007 16:15*	*08.11.2007 17:00*
✓	*Rapport d'activité 9*	*0.75*	*15.11.2007 16:15*	*15.11.2007 17:00*
✓	*Rapport d'activité 10*	*0.75*	*22.11.2007 16:15*	*22.11.2007 17:00*
✓	*Rapport d'activité 11*	*0.75*	*29.11.2007 16:15*	*29.11.2007 17:00*
✗	*Rapport d'activité 12*	*0.75*	*06.12.2007 16:15*	*06.12.2007 17:00*
	PV Séance	**6**	**21.09.2007 08:00**	**07.12.2007 08:30**
✓	*PV Séance 1*	*0.5*	*21.09.2007 08:00*	*21.09.2007 08:30*
✓	*PV Séance 2*	*0.5*	*28.09.2007 08:00*	*28.09.2007 08:30*
✓	*PV Séance 3*	*0.5*	*05.10.2007 08:00*	*05.10.2007 08:30*
✓	*PV Séance 4*	*0.5*	*12.10.2007 08:00*	*12.10.2007 08:30*
✗	*PV Séance 5*	*0.5*	*19.10.2007 08:00*	*19.10.2007 08:30*
✓	*PV Séance 6*	*0.5*	*26.10.2007 08:00*	*26.10.2007 08:30*
✓	*PV Séance 7*	*0.5*	*02.11.2007 08:00*	*02.11.2007 08:30*
✓	*PV Séance 8*	*0.5*	*09.11.2007 08:00*	*09.11.2007 08:30*
✓	*PV Séance 9*	*0.5*	*16.11.2007 08:00*	*16.11.2007 08:30*
✓	*PV Séance 10*	*0.5*	*23.11.2007 08:00*	*23.11.2007 08:30*
✓	*PV Séance 11*	*0.5*	*30.11.2007 08:00*	*30.11.2007 08:30*
✗	*PV Séance 12*	*0.5*	*07.12.2007 08:00*	*07.12.2007 08:30*
Séance hebdomadaire		**13**	**17.09.2007 08:00**	**10.12.2007 09:00**
✓	*Séance hebdomadaire 1*	*1*	*17.09.2007 08:00*	*17.09.2007 11:15*
✓	*Séance hebdomadaire 2*	*1*	*24.09.2007 08:00*	*24.09.2007 09:00*
✓	*Séance hebdomadaire 3*	*1*	*01.10.2007 08:00*	*01.10.2007 09:00*
✓	*Séance hebdomadaire 4*	*1*	*08.10.2007 08:00*	*08.10.2007 09:00*
✗	*Séance hebdomadaire 5*	*1*	*15.10.2007 08:00*	*15.10.2007 09:00*
✓	*Séance hebdomadaire 6*	*1*	*22.10.2007 08:00*	*22.10.2007 09:00*
✓	*Séance hebdomadaire 7*	*1*	*29.10.2007 08:00*	*29.10.2007 09:00*
✓	*Séance hebdomadaire 8*	*1*	*05.11.2007 08:00*	*05.11.2007 09:00*
✓	*Séance hebdomadaire 9*	*1*	*12.11.2007 08:00*	*12.11.2007 09:00*
✓	*Séance hebdomadaire 10*	*1*	*19.11.2007 08:00*	*19.11.2007 09:00*
✓	*Séance hebdomadaire 11*	*1*	*26.11.2007 08:00*	*26.11.2007 09:00*
✗	*Séance hebdomadaire 12*	*1*	*03.12.2007 08:00*	*03.12.2007 09:00*
✗	*Séance hebdomadaire 13*	*1*	*10.12.2007 08:00*	*10.12.2007 09:00*

Cette planification concerne seulement les fonctionnalités "Must Have" et pas les "Nice To Have"

Attestation d'authenticité

ATTESTATION

Je déclare, par ce document, que j'ai effectué le travail de diplôme ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de diplôme, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après : l'Institut Informatique de gestion de la HES-SO Valais, M. Laurent Bagnoud.

Stéphane Probst

Sierre, le 6 décembre 2007.